London Mathematical Society ISSN 1461–1570

SYMBOLIC COLLECTION USING DEEP THOUGHT

C. R. LEEDHAM-GREEN AND LEONARD H. SOICHER

Abstract

We describe the "Deep Thought" algorithm, which can, among other things, take a commutator presentation for a finitely generated torsion-free nilpotent group G, and produce explicit polynomials for the multiplication of elements of G. These polynomials were first shown to exist by Philip Hall, and allow for "symbolic collection" in finitely generated nilpotent groups. We discuss various practical issues in calculations in such groups, including the construction of a hybrid collector, making use of both the polynomials and ordinary collection from the left.

1. *Introduction*

Let G be a finitely generated torsion-free nilpotent group. Then G has a central series

$$G = G_1 > G_2 > \cdots > G_{n+1} = \{1\},$$

such that, for each $1 \leqslant r \leqslant n$, the central factor G_r/G_{r+1} is infinite cyclic, generated by $G_{r+1}a_r$, for some $a_r \in G_r$. Given these $a_1, \ldots, a_n$, each element $x \in G$ has a unique expression of the form

$$x = a_1^{x_1} \cdots a_n^{x_n},$$

with $x_1, \ldots, x_n \in \mathbb{Z}$. We call $(x_1, \ldots, x_n)$ the *vector of exponents* for x.

Philip Hall [3, Theorem 6.5] showed that there are rational polynomials $f_1, \ldots, f_n$ (in $2n$ variables) and $g_1, \ldots, g_n$ (in $n+1$ variables), which describe multiplication and powering in G. More precisely, suppose $t \in \mathbb{Z}$, and that $(x_1, \ldots, x_n)$, $(y_1, \ldots, y_n)$, $(z_1, \ldots, z_n)$, $(w_1, \ldots, w_n)$ are the respective vectors of exponents for $x, y, xy, x^t \in G$. Then, for $1 \leqslant r \leqslant n$,

$$z_r = f_r(x_1, \ldots, x_n, y_1, \ldots, y_n) \quad \text{and} \quad w_r = g_r(x_1, \ldots, x_n, t).$$

Now we know that

$$a_j a_i = a_i a_j \, a_{j+1}^{c_{i,j,j+1}} \cdots a_n^{c_{i,j,n}},$$

for certain integers $c_{i,j,k}$ $(1 \leqslant i < j < k \leqslant n)$. One thing we do in this paper is to show how to compute polynomials in variables corresponding to the x_i, y_i, $c_{i,j,k}$, such that these polynomials have the properties described above for Hall's polynomials f_r, when restricted to specific $c_{i,j,k}$. These polynomials are very complicated indeed, and in practice the construction and evaluation of the polynomials goes much faster when many of the $c_{i,j,k}$ are zero.

We acknowledge our great debt to the work of Philip Hall, who showed in [3] the existence of the polynomials of the form our "Deep Thought" algorithm calculates in the

Received 21st February 1997, revised 15th January 1998; *published 1st June 1998.*
1991 Mathematics Subject Classification 20F12, 20F18, 20F05

case of finitely generated torsion-free nilpotent groups. Charles Sims [8, pp. 441–445] has already described a method of constructing these polynomials of Hall (using interpolation and symbolic algebra), although we have not yet compared his approach to ours. An even more important influence is Philip Hall's earlier paper [2]. There, he introduces the process of collection, and the reader will see a very strong resemblance between our methods and the work of Hall on the calculation of $(xy)^m$ in the free group $F(x, y)$ modulo some term of its lower central series.

The natural setting for what we do is actually monoids, not groups. In the next section we discuss monoids M satisfying certain relations like those of a finitely generated nilpotent group, and we will develop the Deep Thought algorithm for such monoids.

In Section 7 we show that the polynomials produced by Deep Thought can be used for the multiplication of xy even when our monoid M is a finitely generated nilpotent group, and we allow the x_i, y_i, and $c_{i,j,k}$ to be arbitrary integers. We also show how to invert elements of such a group M.

In Section 8 we discuss efficiency issues related to the application of Deep Thought.

Deep Thought provides a form of symbolic collection for finitely generated nilpotent groups, and in Section 9 we discuss how to combine Deep Thought with ordinary collection from the left [4], to be able to compute effectively and efficiently in arbitrary finitely generated nilpotent groups.

2. *The monoid case*

Let M be a monoid which is generated by elements

$$a_1, \ldots, a_n,$$

such that the the following relations hold:

$$a_j a_i = a_i a_j \, a_{j+1}^{c_{i,j,j+1}} \cdots a_n^{c_{i,j,n}} \quad (1 \leqslant i < j \leqslant n), \tag{1}$$

for certain non-negative integers $c_{i,j,k}$ $(1 \leqslant i < j < k \leqslant n)$. The relations (1) allow us to rewrite an arbitrary word w in $a_1, \ldots, a_n$ to a reduced word v, such that $v = w$ in M. A *reduced* word is a word of the form

$$a_1^{x_1} \cdots a_n^{x_n},$$

where $x_1, \ldots, x_n$ are non-negative integers. The rewriting process we use is called *collection to the left*, introduced by Philip Hall [2], and detailed in Figure 1. (Note that since we have no power relations or negative powers to cancel generators during the collection process, collection to the left is probably the best strategy.)

Let

$$x = a_1^{x_1} \cdots a_n^{x_n}, \quad y = a_1^{y_1} \cdots a_n^{y_n}$$

be reduced words. We can use collection to the left to determine a reduced word

$$z = a_1^{z_1} \cdots a_n^{z_n},$$

such that $xy = z$ in M. We shall show how each exponent z_r $(1 \leqslant r \leqslant n)$ is the value of a certain polynomial f_r evaluated on the x_i, y_i, and $c_{i,j,k}$, by describing the algorithm, Deep Thought, which calculates these polynomials, given the relations (1).

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

while w is not a reduced word **do**

Let i^* be the least i such that w contains a subword of the form $a_j a_i$ with $i < j$.

Among the subwords of the form $a_j a_{i^*}$ with $i^* < j$, let $w^* = a_{j^*} a_{i^*}$ be the leftmost such subword.

Replace w^* in w by $a_{i^*} a_{j^*} a_{j^*+1}^{c_{i^*,j^*,j^*+1}} \cdots a_n^{c_{i^*,j^*,n}}$.

od;

Figure 1: Collection to the left, of w in M.

3. *Letters occurring in a collection*

Throughout this Section we keep the notation of Section 2.

In order to analyse the process of collection (to the left) of xy in M, we shall attach a unique label to each specific a_i which is introduced in the collection process. These labels come from the set of "letters", which are defined later. Each letter labelling a generator in the final reduced word is either a letter labelling a generator from the original left-hand word x, or a letter labelling a generator from the original right-hand word y, or is a letter labelling a generator from a word of the form $a_{j+1}^{c_{i,j,j+1}} \cdots a_n^{c_{i,j,n}}$, which is introduced when an $a_j a_i$ (with $i < j$) is collected. We emphasize that in a collection to the left in M, each generator (and its label) which is introduced will appear in the final collected reduced word, since there are no power relations or inverses to cancel generators.

3.1. *Atoms and non-atoms*

We now (recursively) define the concept of a letter, and show how letters are used to label uniquely the generators which are introduced in the collection of xy in M. (As usual, $x = a_1^{x_1} \cdots a_n^{x_n}$ and $y = a_1^{y_1} \cdots a_n^{y_n}$.)

A *letter* is either an atom or a non-atom. An *atom* α is a 3-tuple

$$(\mathrm{side}(\alpha), \mathrm{num}(\alpha), \mathrm{pos}(\alpha)),$$

such that $\mathrm{side}(\alpha) \in \{\mathsf{L}, \mathsf{R}\}$, and $\mathrm{num}(\alpha)$ and $\mathrm{pos}(\alpha)$ are positive integers. We use the notation

$$\mathrm{num}(\alpha)_{\mathrm{pos}(\alpha)}^{\mathrm{side}(\alpha)}$$

for an atom α.

If $x_i \geqslant j > 0$ then the atom i_j^{L} is used to label the j-th from the left a_i in x (the Left-hand word). Similarly, if $y_i \geqslant j > 0$ then i_j^{R} is used to label the j-th from the left a_i in y (the Right-hand word).

If a letter is not an atom then it is a non-atom. A *non-atom* α is a 4-tuple

$$(\mathrm{left}(\alpha), \mathrm{right}(\alpha), \mathrm{num}(\alpha), \mathrm{pos}(\alpha)),$$

such that $\mathrm{left}(\alpha)$ and $\mathrm{right}(\alpha)$ are letters, and $\mathrm{num}(\alpha)$ and $\mathrm{pos}(\alpha)$ are positive integers. We use the notation

$$[\,\mathrm{left}(\alpha), \mathrm{right}(\alpha)\,;\ \mathrm{num}(\alpha)_{\mathrm{pos}(\alpha)}\,]$$

for a non-atom α. We also require that this recursion terminates, so that all letters can be

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

described, using the above notation, as finite expressions involving only atoms. (Note that a letter is just a finite binary tree, with certain information attached to the nodes, and this is how we view a letter from a computational point of view. The leaves of these trees are atoms.)

Now suppose $i < j < k$ and $c_{i,j,k} \geqslant r > 0$, and that in a collection step we replace a specific $a_j a_i$ by $a_i a_j a_{j+1}^{c_{i,j,j+1}} \cdots a_n^{c_{i,j,n}}$. We may assume (inductively) that these specific a_j, a_i have respective labels δ, γ. They retain these labels, and we label the r-th from the left a_k which is introduced in this step by the letter $[\delta, \gamma\,;\, k_r]$.

We have thus defined letters, and shown how to label uniquely each specific generator introduced in the collection of xy. Indeed, we can now think of the collection process as collecting letters rather than generators. We say that a letter *occurs* in the collection of xy if it labels a generator introduced in this collection. Thus, a letter occurs in the collection of xy if and only if it labels a generator in the final collected reduced word.

Example 1. Define the monoid N to be generated by a_1, a_2, a_3, a_4, subject (only) to the relations

$$a_2 a_1 = a_1 a_2 a_3^2,\ a_3 a_1 = a_1 a_3 a_4,\ a_3 a_2 = a_2 a_3 a_4^5,\ \text{and } a_4 a_i = a_i a_4,\ (i = 1, 2, 3).$$

Let $x = a_1 a_2^2 a_3$ and $y = a_1^3 a_3 a_4$. Examples of atoms which occur in the collection of xy in N are 3_1^{L} and 1_2^{R}. An example of a non-atom which occurs in the collection of xy is $[\,[\,2_2^{\mathsf{L}}, 1_1^{\mathsf{R}}\,;\,3_1\,]\,, 1_3^{\mathsf{R}}\,;\,4_1\,]$.

4. *Relations on letters*

We now define various relations on the set of letters, including the important equivalence relation, $\sim$. Given monoid relations of the form (1), the Deep Thought algorithm first determines the $\sim$-classes which can have elements occurring in a collection to the left. For each such equivalence class, Deep Thought also determines a polynomial which evaluates to the number of letters in that class that occur in a specified collection.

First we make clear what we mean for two letters α, β to be *equal* (written $\alpha = \beta$). Simply, they must both be atoms or both non-atoms, and must be equal as 3-tuples or 4-tuples, respectively. The conditions for letters α, β to be *almost equal* (written $\alpha \approx \beta$) are the same as for them being equal, except that we allow the possibility that $\mathrm{pos}(\alpha) \neq \mathrm{pos}(\beta)$. Any letter is both equal and almost equal to itself.

Example 2. $[\,[\,2_1^{\mathsf{L}}, 1_1^{\mathsf{R}}\,;\,3_1\,]\,, 1_2^{\mathsf{R}}\,;\,4_5\,]$ is almost equal to $[\,[\,2_1^{\mathsf{L}}, 1_1^{\mathsf{R}}\,;\,3_1\,]\,, 1_2^{\mathsf{R}}\,;\,4_3\,]$, but is not equal to it. On the other hand, $[\,[\,2_1^{\mathsf{L}}, 1_1^{\mathsf{R}}\,;\,3_1\,]\,, 1_2^{\mathsf{R}}\,;\,4_5\,]$ is not almost equal to $[\,[\,2_1^{\mathsf{L}}, 1_1^{\mathsf{R}}\,;\,3_2\,]\,, 1_2^{\mathsf{R}}\,;\,4_5\,]$. Also note that $1_2^{\mathsf{L}} \approx 1_1^{\mathsf{L}}$, but $1_2^{\mathsf{L}} \not\approx 1_2^{\mathsf{R}}$.

A letter α determines a finite sequence $\mathrm{Seq}(\alpha)$ of (not necessarily distinct) *subletters* of α, as follows. If α is an atom then $\mathrm{Seq}(\alpha) = (\alpha)$; otherwise

$$\mathrm{Seq}(\alpha) = \mathrm{Seq}(\mathrm{left}(\alpha)) \text{ concatenate } \mathrm{Seq}(\mathrm{right}(\alpha)) \text{ concatenate } (\alpha).$$

(In binary tree language, $\mathrm{Seq}(\alpha)$ is a postorder transversal of α.) Let $m = \mathrm{Length}(\mathrm{Seq}(\alpha))$. For $1 \leqslant i \leqslant m$, we denote the i-th element of $\mathrm{Seq}(\alpha)$ by $\mathrm{Seq}(\alpha, i)$, and let

$$\mathrm{Sub}(\alpha) = \{\mathrm{Seq}(\alpha, i) \mid \Delta \leqslant i \leqslant m\}$$

denote the set of subletters of α.

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

The first condition that must be satisfied to have $\alpha \sim \beta$ is that α and β have the same *structure*, which roughly means that α and β are exactly the same if we (recursively) ignore all pos attributes. More precisely, α and β have the same structure if the following are satisfied:

- α, β must both be atoms or both be non-atoms.
- If α, β are atoms they must be almost equal.
- If α, β are non-atoms, then $\text{num}(\alpha) = \text{num}(\beta)$, $\text{left}(\alpha)$ has the same structure as $\text{left}(\beta)$, and $\text{right}(\alpha)$ has the same structure as $\text{right}(\beta)$.

Note that if α and β have the same structure, we have $\text{Length}(\text{Seq}(\alpha)) = \text{Length}(\text{Seq}(\beta))$.

We are now in a position to define the relation $\sim$ on letters. We have $\alpha \sim \beta$ if each of the following is satisfied:

- α, β have the same structure.
- For $1 \leqslant i < j \leqslant \text{Length}(\text{Seq}(\alpha))$:
  - $\text{Seq}(\alpha, i) \approx \text{Seq}(\alpha, j)$ if and only if $\text{Seq}(\beta, i) \approx \text{Seq}(\beta, j)$.
  - If $\text{Seq}(\alpha, i) \approx \text{Seq}(\alpha, j)$ then $\text{pos}(\text{Seq}(\beta, i))$ and $\text{pos}(\text{Seq}(\beta, j))$ must be in exactly the same relation ($<$, $=$, or $>$) as $\text{pos}(\text{Seq}(\alpha, i))$ and $\text{pos}(\text{Seq}(\alpha, j))$.

Example 3. We have

$$[\,[\,2_4^{\mathsf{L}}, 1_3^{\mathsf{R}};\ 3_1\,], 1_4^{\mathsf{R}};\ 4_1\,] \sim [\,[\,2_2^{\mathsf{L}}, 1_1^{\mathsf{R}};\ 3_1\,], 1_3^{\mathsf{R}};\ 4_5\,].$$

As a more complicated example, we have

$$[\,[\,[\,2_2^{\mathsf{L}}, 1_1^{\mathsf{R}};\ 4_2\,], 1_2^{\mathsf{R}};\ 6_1\,], [\,[\,2_2^{\mathsf{L}}, 1_1^{\mathsf{R}};\ 4_3\,], 1_3^{\mathsf{R}};\ 5_1\,];\ 12_2\,]$$

$$\sim [\,[\,[\,2_2^{\mathsf{L}}, 1_1^{\mathsf{R}};\ 4_1\,], 1_2^{\mathsf{R}};\ 6_1\,], [\,[\,2_2^{\mathsf{L}}, 1_1^{\mathsf{R}};\ 4_3\,], 1_3^{\mathsf{R}};\ 5_1\,];\ 12_2\,]$$

$$\not\sim [\,[\,[\,2_2^{\mathsf{L}}, 1_1^{\mathsf{R}};\ 4_2\,], 1_2^{\mathsf{R}};\ 6_1\,], [\,[\,2_2^{\mathsf{L}}, 1_1^{\mathsf{R}};\ 4_2\,], 1_3^{\mathsf{R}};\ 5_1\,];\ 12_2\,].$$

Lemma 1. *The relation $\sim$ is an equivalence relation on the set of letters.*

Proof This follows from the fact that both "has the same structure as" and $\approx$ are equivalence relations on the set of letters. □

For each $\sim$-class of letters, we define a well-ordering $\leqslant$ on that class. Suppose $\alpha \sim \beta$. If α and β are both atoms then define $\alpha \leqslant \beta$ if $\text{pos}(\alpha) \leqslant \text{pos}(\beta)$. Otherwise, $\alpha \leqslant \beta$ if in lexicographic order we have

$$(\text{left}(\alpha), \text{right}(\alpha), \text{pos}(\alpha)) \leqslant (\text{left}(\beta), \text{right}(\beta), \text{pos}(\beta)).$$

(Note that if α and β are non-atoms with $\alpha \sim \beta$, then $\text{left}(\alpha) \sim \text{left}(\beta)$ and $\text{right}(\alpha) \sim \text{right}(\beta)$.)

Lemma 2. *A letter α is least in its $\sim$-class if and only if for each $\approx$-class A of* $\text{Sub}(\alpha)$*, we have* $\{\text{pos}(\rho) \mid \rho \in A\} = \{1, \ldots, |A|\}$.

We leave the straightforward proof as an exercise in understanding the many definitions in this section.

Example 4. The least letter in the $\sim$-class of

$$[\,[\,[\,2_2^{\mathsf{L}}, 1_1^{\mathsf{R}};\ 4_1\,], 1_2^{\mathsf{R}};\ 6_1\,], [\,[\,2_2^{\mathsf{L}}, 1_1^{\mathsf{R}};\ 4_3\,], 1_3^{\mathsf{R}};\ 5_1\,];\ 12_2\,]$$

is

$$[\,[\,[\,2_1^{\mathsf{L}}, 1_1^{\mathsf{R}};\ 4_1\,], 1_2^{\mathsf{R}};\ 6_1\,], [\,[\,2_1^{\mathsf{L}}, 1_1^{\mathsf{R}};\ 4_2\,], 1_3^{\mathsf{R}};\ 5_1\,];\ 12_1\,].$$

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

earlier := **function**(α, β)
#
Suppose that α and β are letters occurring in a collection (to the left),
such that α, β are not both atoms, and if both α, β are
non-atoms then left(α) $\neq$ left(β) or right(α) $\neq$ right(β).
Then this boolean function returns `true` if and only if a
generator is labelled by α strictly earlier in the collection
than a generator is labelled by β.
#
if α is an atom **then return** `true`; **fi**;
if β is an atom **then return** `false`; **fi**;
if right(α) = right(β) **then return** left_of(left(β), left(α)); **fi**;
if num(right(α)) = num(right(β)) **then return** left_of(right(α), right(β)); **fi**;
return num(right(α)) < num(right(β));
end;

Figure 2: The function "earlier".

5. *The function "left_of"*

One basic step in the Deep Thought algorithm is the following. Given two letters α, β, which occur in a collection (to the left), with $\mathrm{num}(\alpha) = j > i = \mathrm{num}(\beta)$, we must decide if at the first instance in the collection that α and β both label generators, the generator labelled by α is to the left of the generator labelled by β in the word being collected. If this is so, then all the letters $[\,\alpha, \beta\,;\, k_r\,]$ will occur for $1 \leqslant r \leqslant c_{i,j,k}$.

This decision is made by the function "left_of", detailed in Figure 3, operating in a double recursion with the function "earlier", detailed in Figure 2. These functions are given in an algorithmic language similar to GAP [7]. In particular, "#" denotes a comment until the end-of-line, "**fi**" denotes the end of an **if**-statement, "**od**" denotes the end of a **while**-statement, and "**return**" followed by an expression means to return the value of that expression as the function value and then terminate the execution of the function.

It not difficult to prove that earlier(α, β) and left_of(α, β) both return the correct results when given valid input, by considering the various cases the functions handle, and using induction on $\mathrm{Length}(\mathrm{Seq}(\alpha)) + \mathrm{Length}(\mathrm{Seq}(\beta))$.

On examination of the functions left_of and earlier, we see that the following (useful) lemma holds.

Lemma 3. *Suppose α is a non-atom, with* left(α) $\neq$ right(α). *Then if $\beta \sim \alpha$, we have* left_of(left(β), right(β)) *if and only if* left_of(left(α), right(α)).

(Note that left_of(left(α), right(α)) holds when the non-atom α occurs in a collection.)

6. *The Deep Thought algorithm*

We are now in a position to describe the basic Deep Thought algorithm.

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

```
left_of := function(α, β)
#
# Suppose that α and β are letters occurring in a collection
# (to the left), such that α ≠ β.
# Then this boolean function returns true if and only if
# at the first instance that letters α and β both label generators,
# the generator labelled by α is to the left of that labelled by β.
#
if both α, β are atoms then
    if side(α) = L and side(β) = R then return true; fi;
    if side(α) = R and side(β) = L then return false; fi;
    if num(α) = num(β) then return pos(α) < pos(β); fi;
    return num(α) < num(β);
fi;
if both α, β are non-atoms and right(α) = right(β) and left(α) = left(β) then
    if num(α) = num(β) then return pos(α) < pos(β); fi;
    return num(α) < num(β);
fi;
if earlier(α, β) then return not left_of(β, α); fi;
#
# At this point, we know that β appears earlier than α, and so α must be a non-atom.
#
if num(β) < num(right(α)) then return false; fi;
if β = right(α) then return false; fi;
if num(β) = num(right(α)) then return left_of(right(α), β); fi;
#
# At this point, we have num(β) > num(right(α)).
#
if β = left(α) then return false; fi;
return left_of(left(α), β);
end;
```

Figure 3: The function “left_of”.

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

The Deep Thought algorithm takes as input the integer n, and relations of the form (1). The algorithm outputs polynomials $f_1, \ldots, f_n$ in indeterminates corresponding to $x_1, \ldots, x_n, y_1, \ldots, y_n$ and the non-zero $c_{i,j,k}$, such that if z_r is the value of f_r evaluated on $x_1, \ldots, x_n, y_1, \ldots, y_n$ and the non-zero $c_{i,j,k}$, then

$$a_1^{z_1} \cdots a_n^{z_n} = a_1^{x_1} \cdots a_n^{x_n}\, a_1^{y_1} \cdots a_n^{y_n},$$

in a monoid M whose generators $a_1, \ldots, a_n$ satisfy (1).

Remark The actual values of the non-zero $c_{i,j,k}$ are only required in the evaluation of the polynomials produced by Deep Thought. Indeed, Deep Thought produces polynomials which are valid for all monoids N generated by $b_1, \ldots, b_n$ satisfying relations

$$b_j b_i = b_i b_j\, b_{j+1}^{d_{i,j,j+1}} \cdots b_n^{d_{i,j,n}} \quad (1 \leqslant i < j \leqslant n),$$

as long as $d_{i,j,k} = 0$ whenever $c_{i,j,k} = 0$. The reason we only consider the non-zero $c_{i,j,k}$ in the algorithm (rather than to obtain more general formulae) is to keep the polynomials f_r of reasonable size for the given monoid relations.

Deep Thought starts by determining, for each $r = 1, \ldots, n$, a set reps_r. When complete, reps_r is a set of representatives for all distinct $\sim$-classes of letters α, with $\mathrm{num}(\alpha) = r$, such that α occurs in a collection of xy, for some values of the x_i, y_i, and non-zero $c_{i,j,k}$. Then, we will define a polynomial g_α, in indeterminates corresponding to the x_i, y_i, and non-zero $c_{i,j,k}$, such that g_α evaluates to the number of letters occurring in the $\sim$-class of α in the actual collection corresponding to given values of the x_i, y_i, and non-zero $c_{i,j,k}$.

The procedure set_reps for determining the sets reps_r is detailed in Figure 4.

After set_reps has been executed, we see that each reps_r is a set of inequivalent letters α, with $\mathrm{num}(\alpha) = r$, that α is least in its $\sim$-class, and that any letter β, with $\mathrm{num}(\beta) = r$, occurring in the collection of xy is equivalent to some element of reps_r.

6.1. *The polynomials g_α*

Let non-negative integers x_i, y_i, $c_{i,j,k}$ be given, and let ρ be any letter. Define t_ρ by

$$t_\rho = \begin{cases} x_{\mathrm{num}(\rho)} & \text{if } \rho \text{ an atom, } \mathrm{side}(\rho) = \mathsf{L} \\ y_{\mathrm{num}(\rho)} & \text{if } \rho \text{ an atom, } \mathrm{side}(\rho) = \mathsf{R} \\ c_{\mathrm{num}(\mathrm{right}(\rho)),\mathrm{num}(\mathrm{left}(\rho)),\mathrm{num}(\rho)} & \text{otherwise.} \end{cases}$$

So t_ρ depends only on the integers x_i, y_i, $c_{i,j,k}$ and the $\approx$-class A of ρ. We define t_A for this $\approx$-class A to be t_ρ.

Theorem 1. *Let $x = a_1^{x_1} \cdots a_n^{x_n}$, $y = a_1^{y_1} \cdots a_n^{y_n}$. Then for each $r = 1, \ldots, n$, and each $\alpha \in \mathrm{reps}_r$ calculated by the procedure* set_reps *with input n and relations (1), the number of elements in the $\sim$-class of α which occur in the collection (to the left) of xy is*

$$n_\alpha = \prod_{A \in \mathrm{Sub}(\alpha)/\approx} \binom{t_A}{|A|}. \tag{2}$$

Proof For any letter α, define $N(\alpha)$ to be the set of all letters $\beta \sim \alpha$, such that $\mathrm{pos}(\gamma) \leqslant t_\gamma$ for each $\gamma \in \mathrm{Sub}(\beta)$. Note that for each $B \in \mathrm{Sub}(\alpha)/\approx$ this restriction gives us exactly $\binom{t_B}{|B|}$ possibilities for $\{\mathrm{pos}(\sigma) \mid \sigma \in B\}$, so $N(\alpha)$ has size n_α, where n_α is given by (2).

Now if β is a letter occurring in the collection of xy, we plainly must have $\max\{\mathrm{pos}(\sigma) \mid \sigma \in B\} \leqslant t_B$, for each $\approx$-class B of $\mathrm{Sub}(\beta)$. Therefore, $N(\alpha)$ contains the set $C(\alpha)$ of letters which occur in the collection of xy and are in the $\sim$-class of α. We complete the proof of the theorem by showing that $N(\alpha) \subseteq C(\alpha)$, when α is in one of the sets reps_r.

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

```
set_reps := procedure(n, (c_{i,j,k}))
#
for r := 1, ..., n do   # initialize reps_r with representative atoms
    reps_r := {r_1^L, r_1^R};
od;
for r := 3, ..., n do    # determine the representative non-atoms
    #
    # Loop invariant: the sets reps_1, ..., reps_{r-1} are complete.
    #
    for each i, j, with 1 ⩽ i < j < r, such that c_{i,j,j+1} = ··· = c_{i,j,r-1} = 0 ≠ c_{i,j,r}, do
        #
        # Determine representatives for all letters which can possibly
        # occur when an a_j a_i is collected.
        #
        for each α ∈ reps_i, β ∈ reps_j do
            #
            # Lemma 2 is useful in the tricky business of generating
            # efficiently the pairs γ, δ to be looped over next.
            #
            for each pair γ, δ with
                γ ∼ α, δ ∼ β, and [ δ, γ ; r_1 ] the least letter in its ∼-class do
                if left_of(δ, γ) then
                    #
                    # for each k such that c_{i,j,k} ≠ 0, [ δ, γ ; k_1 ] is the least letter in its ∼-class
                    #
                    for each k ∈ {r, ..., n} such that c_{i,j,k} ≠ 0 do Add [ δ, γ ; k_1 ] to reps_k; od;
                fi;
            od;
        od;
    od;
od;
end;
```

Figure 4: The procedure “set_reps”.

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

Let $R = \cup_{r=1}^{n} \text{reps}_r$. For each letter $\alpha \in R$ and for each $\beta \in N(\alpha)$, we show that $\beta \in C(\alpha)$, by induction on $l = \text{Length}(\text{Seq}(\beta))$.

If $l = 1$ then β is an atom, and $\text{pos}(\beta) \leqslant t_\beta$, that is, $\text{pos}(\beta) \leqslant x_{\text{num}(\beta)}$ if $\text{side}(\beta) = \mathsf{L}$, and $\text{pos}(\beta) \leqslant y_{\text{num}(\beta)}$ if $\text{side}(\beta) = \mathsf{R}$. Thus the result holds for $l = 1$.

Now assume that $\beta \in N(\alpha)$ is a non-atom, and let λ be the least letter in the $\sim$-class of $\text{left}(\beta)$ and ρ be the least letter in the $\sim$-class of $\text{right}(\beta)$. Since $\beta \sim \alpha$, it follows that $\text{left}(\beta) \sim \text{left}(\alpha)$, and $\text{right}(\beta) \sim \text{right}(\alpha)$; so λ is also the least letter in the $\sim$-class of $\text{left}(\alpha)$ and ρ is the least letter in the $\sim$-class of $\text{right}(\alpha)$. Since $\alpha \in R$, from the set_reps construction of the sets reps_r, it follows that $\lambda \in R$ and $\rho \in R$. Also $\beta \in N(\alpha)$ implies $\text{left}(\beta) \in N(\lambda)$ and $\text{right}(\beta) \in N(\rho)$. Therefore, by our inductive hypothesis, $\text{left}(\beta)$ and $\text{right}(\beta)$ occur in the collection of xy. Since $\alpha \in R$, we know that $\text{left\_of}(\text{left}(\alpha), \text{right}(\alpha))$ holds, but then $\text{left\_of}(\text{left}(\beta), \text{right}(\beta))$ also holds by Lemma 3. But $\text{pos}(\beta) \leqslant t_\beta = c_{\text{num}(\text{right}(\beta)),\text{num}(\text{left}(\beta)),\text{num}(\beta)}$, so β occurs in the collection of xy, and the proof is complete. □

We now define the polynomial g_α that evaluates to the number of letters equivalent under $\sim$ to α occurring in the collection of xy. This definition is justified by the preceding theorem.

Let $X_i, Y_i, C_{i,j,k}$ be indeterminates, and let ρ be any letter. Define the indeterminate T_ρ by

$$T_\rho = \begin{cases} X_{\text{num}(\rho)} & \text{if } \rho \text{ an atom, } \text{side}(\rho) = \mathsf{L} \\ Y_{\text{num}(\rho)} & \text{if } \rho \text{ an atom, } \text{side}(\rho) = \mathsf{R} \\ C_{\text{num}(\text{right}(\rho)),\text{num}(\text{left}(\rho)),\text{num}(\rho)} & \text{otherwise.} \end{cases}$$

So T_ρ depends only on the indeterminates $X_i, Y_i, C_{i,j,k}$ and the $\approx$-class A of ρ. We define T_A for this $\approx$-class A to be T_ρ. Let T be an indeterminate, k a non-negative integer, and define the polynomial $\binom{T}{k}$ by

$$\binom{T}{k} = \prod_{i=1}^{k} \frac{T - i + 1}{i}.$$

Now, finally, define

$$g_\alpha = \prod_{A \in \text{Sub}(\alpha)/\approx} \binom{T_A}{|A|}.$$

For example, if $\delta = [\,[\,2_1^{\mathsf{L}}, 1_1^{\mathsf{R}}\,;\, 3_1\,], 1_2^{\mathsf{R}}\,;\, 4_1\,]$, then $g_\delta = \binom{X_2}{1}\binom{Y_1}{2}\binom{C_{1,2,3}}{1}\binom{C_{1,3,4}}{1}$.

Note that g_α has total degree equal to $|\text{Sub}(\alpha)|$.

6.2. *The output of Deep Thought*

The final steps in the Deep Thought algorithm, after computing the sets reps_r, are to calculate

$$f_r = \sum_{\alpha \in \text{reps}_r} g_\alpha,$$

for $1 \leqslant r \leqslant n$, and to output these f_r.

Remark To evaluate an f_r on given integers $x_i, y_i, c_{i,j,k}$, we simply substitute for each indeterminate T_A in the expression for f_r, the corresponding value t_A. Note that f_r is a rational polynomial which takes integer values when evaluated on integers.

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

7. *The group case and negative exponents*

Let G be an arbitrary finitely generated nilpotent group. Then for some n, G contains a generating sequence $a_1, \ldots, a_n$, such that these generators satisfy relations of the form (1), for certain (not necessarily non-negative) integers $c_{i,j,k}$ $(1 \leqslant i < j < k \leqslant n)$. Each element of G can be written in the form $a_1^{x_1} \cdots a_n^{x_n}$, where $x_1, \ldots, x_n$ are integers.

We may run the Deep Thought algorithm with input n and the relations (1), even if some of the $c_{i,j,k}$ are negative, since the algorithm only cares whether a given $c_{i,j,k}$ is zero or not. Suppose this run produces sets $\text{reps}_1, \ldots, \text{reps}_n$, and polynomials $f_1, \ldots, f_n$ as output. Then we have the following:

Theorem 2. *Let $x_1, \ldots, x_n, y_1, \ldots, y_n$ be arbitrary integers, and for $1 \leqslant r \leqslant n$, let z_r be the integer obtained by evaluating f_r, on the given x_i, y_i, and non-zero $c_{i,j,k}$. If $x = a_1^{x_1} \cdots a_n^{x_n}$, and $y = a_1^{y_1} \cdots a_n^{y_n}$, and $z = a_1^{z_1} \cdots a_n^{z_n}$, then $xy = z$ in G.*

Proof Since G is a polycyclic group, it is residually finite by a theorem of K. Hirsch (see [6, 5.4.17]), and so $xy = z$ in G if and only if $\bar{x}\bar{y} = \bar{z}$ in every finite quotient $\bar{G}$ of G. Thus, it suffices to prove the theorem under the assumption that G is finite.

We now assume that G is finite, but $xy \neq z$ in G. Then $xy \neq z$ in G if we add to each exponent x_r, y_r, and z_r of x, y, and z some multiple of $|G|$. Note that this would be a consequence of adding to each x_i, y_i, and non-zero $c_{i,j,k}$ a multiple of

$$\max\{|A| \mid A \in \text{Sub}(\alpha)/\approx, \alpha \in \cup_{r=1}^{n} \text{reps}_r\}! \times |G|,$$

and recalculating each z_r. But then this would imply that $xy \neq z$ in G even if each x_i, y_i, and $c_{i,j,k}$ is non-negative, which would contradict the fact that Deep Thought works when all such exponents are non-negative. □

7.1. *Inverting elements of G*

We now show how to invert elements of G using Deep Thought polynomials. More generally, we show how to solve for y in the equation $xy = z$ in G, where now $x = a_1^{x_1} \cdots a_n^{x_n}$ and $z = a_1^{z_1} \cdots a_n^{z_n}$ are given, and we wish to determine $y = a_1^{y_1} \cdots a_n^{y_n}$ (of course, calculating $y = x^{-1}$ is just the special case $z_1 = \cdots = z_n = 0$).

We first make a copy x' of x. Then, for $i := 1, \ldots, n$, given the loop invariant that x' is of the form $a_1^{z_1} \cdots a_{i-1}^{z_{i-1}} a_i^{x'_i} \cdots a_n^{x'_n}$, set $y_i := z_i - x'_i$, and use the Deep Thought polynomials to calculate $x' := x' a_i^{y_i}$.

7.2. *Calculating normal forms of elements of G*

We have shown that Deep Thought polynomials can be used to multiply and invert elements of an arbitrary finitely generated nilpotent group, and that Deep Thought handles negative exponents correctly. The only problem is that the result $w = a_1^{w_1} \ldots a_n^{w_n}$ of such a multiplication or inversion need not be uniquely determined by the group element represented by w. We get around this problem by using a consistent power-commutator presentation, and we show how to use the power relations of such a presentation and Deep Thought to convert a result into canonical (normal) form.

Let G be an arbitrary finitely generated nilpotent group. Then G has (for some n) a so-called consistent power-commutator presentation of the form

$$\langle a_1, \ldots, a_n \mid a_i^{m_i} = a_{i+1}^{c_{i,i,i+1}} \cdots a_n^{c_{i,i,n}},\ a_j a_i = a_i a_j\, a_{j+1}^{c_{i,j,j+1}} \cdots a_n^{c_{i,j,n}}\ (i < j) \rangle, \qquad (3)$$

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

such that each m_i $(1 \leqslant i \leqslant n)$ is a non-negative integer, and each element of G has a unique expression of the form $a_1^{x_1} \cdots a_n^{x_n}$, where $x_1, \ldots, x_n$ are integers, and if $m_i > 0$, then $0 \leqslant x_i < m_i$. (If $m_i = 0$ then we must have $c_{i,i,i+1} = \cdots = c_{i,i,n} = 0$ and the relation with $m_i = 0$ can effectively be ignored.) This unique expression for an element of G is referred to as the *normal form* of that element.

Now if each $m_i = 0$ in the consistent power-commutator presentation (3) for G, then Deep Thought polynomials can be used directly to multiply elements in G, such that the result is always in normal form. Otherwise, we may obtain a result $z = a_1^{z_1} \cdots a_n^{z_n}$, such that for some i with $m_i > 0$, we have $z_i < 0$ or $z_i \geqslant m_i$. Suppose this is the case, and let j be the least i with this property. Then we need to "normalize" $z^* = a_j^{z_j} \cdots a_n^{z_n}$, that is, replace it by a word in normal form representing the same element of G. Let $z_j = qm_j + r$, where q and r are integers and $0 \leqslant r < m_j$. We calculate

$$w := (a_{j+1}^{c_{j,j,j+1}} \cdots a_n^{c_{j,j,n}})^q \, a_{j+1}^{z_{j+1}} \cdots a_n^{z_n},$$

using Deep Thought polynomials, so that w has the form $a_{j+1}^{w_{j+1}} \cdots a_n^{w_n}$. We then (recursively) normalize w, and so the normal form for z^* is $a_j^r \, w$, and the normal form for z is

$$a_1^{z_1} \cdots a_{j-1}^{z_{j-1}} \, a_j^r \, w.$$

7.3. *Determining the order of an element of G*

Suppose (3) is a consistent power-commutator presentation for G, and x is an element of G in normal form. Then we can calculate the order $|x|$ of x as follows.

If x is the empty word then $|x| = 1$. Otherwise, $x = a_j^{x_j} \cdots a_n^{x_n}$, with $j \leqslant n$ and $x_j \neq 0$. If $m_j = 0$ then $|x| = \infty$. Otherwise let $m = m_j / \gcd(m_j, x_j)$. Now $|x| = \infty$ if and only if $|x^m| = \infty$, and if the order of x is finite then m divides $|x|$. We then calculate the normal form of $y = x^m$, which is $a_l^{y_l} \cdots a_n^{y_n}$, with $l > j$, and recursively determine $|y|$. If $|y| = \infty$ then $|x| = \infty$, otherwise $|x| = m|y|$.

7.4. *On the degree of Deep Thought polynomials*

Suppose (3) is a consistent power-commutator presentation for the group G, such that the central series defined by $a_1, \ldots, a_n$ refines the lower central series

$$G = \gamma_1(G) > \gamma_2(G) > \cdots > \gamma_{c+1}(G) = \{1\}$$

of G. Let α be an element of a set reps_r determined by Deep Thought with input (3).

It is not difficult to see that $\mathrm{Seq}(\alpha)$ contains at most c elements which are atoms (since $[\gamma_i(G), \gamma_j(G)] \leqslant \gamma_{i+j}(G)$). It follows that g_α, considered as a polynomial in the X_i and Y_i only, has degree at most c, and the same holds for the Deep Thought polynomial f_r.

Suppose $\mathrm{Seq}(\alpha)$ has exactly d elements which are atoms. Then

$$\mathrm{Length}(\mathrm{Seq}(\alpha)) = 2d - 1$$

(proof by induction on d). It follows that g_α has total degree (in the X_i, Y_i, and $C_{i,j,k}$) at most $2d - 1 \leqslant 2c - 1$, and the same holds for the Deep Thought polynomial f_r.

8. *Efficiency issues in applying Deep Thought*

Let G be a group with presentation (3), and suppose we are interested in efficient multiplication and inversion of elements in G, rather than in the polynomials f_r produced by Deep Thought.

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

for $r := 1, \ldots, n$ **do**
  if $r = s$ **then**
    $\text{reps}_{rs} := \{r_1^{\mathsf{L}}, r_1^{\mathsf{R}}\}$;
  else
    $\text{reps}_{rs} := \{r_1^{\mathsf{L}}\}$;
  fi;
od;

Figure 5: Initial **for**-loop for the calculation of the sets reps_{rs}.

8.1. *The polynomials* f_{rs}

The first important observation is that we should not actually calculate and use the Deep Thought polynomials f_r for multiplying and inverting elements of G, but instead, closely related polynomials f_{rs}, described below, which can be calculated by a small variant of the Deep Thought algorithm.

Let $\text{reps}_1, \ldots, \text{reps}_n$ be the sets of representative letters produced by Deep Thought, using input (3). For $1 \leqslant r, s \leqslant n$, define

$$\text{reps}_{rs} =$$

$$\{\alpha \in \text{reps}_r \mid \text{if } \beta \in \text{Sub}(\alpha),\ \beta \text{ an atom with side}(\beta) = \mathsf{R},\ \text{then num}(\beta) = s\},$$

and

$$f_{rs} = \sum_{\alpha \in \text{reps}_{rs}} g_\alpha.$$

Then for $1 \leqslant r, s \leqslant n$, we see that if z_r is the value of f_{rs} evaluated on $x_1, \ldots, x_n$, y_s, and the (non-zero) $c_{i,j,k}$, then

$$a_1^{x_1} \cdots a_n^{x_n}\, a_s^{y_s} = a_1^{z_1} \cdots a_n^{z_n} \tag{4}$$

in the group G.

To determine $z = xy$, for arbitrary $y = a_1^{y_1} \cdots a_n^{y_n}$, we just set $z := x$, and then, for $s := 1, \ldots, n$, we set $z := z a_s^{y_s}$, using the f_{rs}.

Furthermore, efficient multiplication of the form (4) is exactly what we need when inverting elements of G and, as it will turn out, what we also need for hybrid collection in G, described in Section 9.

Even better, calculating all the sets reps_{rs} is computationally no harder (and is often easier) than calculating all the sets reps_r. To calculate the sets reps_{rs}, for a fixed s and for $1 \leqslant r \leqslant n$, we use an algorithm which is almost the same as that detailed in Figure 4. The first change is that the initial **for**-loop should be replaced by the code in Figure 5. Then, thoughout the rest of the algorithm, every occurrence of $\text{reps}_\square$ should be replaced by $\text{reps}_{\square s}$.

8.2. *More on efficiency*

The next observation is that if (in our fixed presentation (3)) some $c_{i,j,k} > 0$, then when calculating the sets reps_{rs}, we need not include a representative α with an $\approx$-class A of subletters with $T_A = C_{i,j,k}$ but $|A| > c_{i,j,k}$, because if $\beta \sim \alpha$ is a subletter of γ, then g_γ

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

while w is not in normal form **do**

Let w^* be the leftmost subword of w of the form $a_j a_i$ with $i < j$, or of the form $a_i^{m_i}$, where $m_i > 0$ in (3).

if $w^* = a_j a_i$, with $i < j$, **then** replace w^* in w by $a_i a_j\, a_{j+1}^{c_{i,j,j+1}} \cdots a_n^{c_{i,j,n}}$;

else replace w^* in w by $a_{i+1}^{c_{i,i,i+1}} \cdots a_n^{c_{i,i,n}}$ **fi**;

od;

Figure 6: Collection from the left, of w in G.

would always evaluate to 0 with this $c_{i,j,k}$.

Another obvious point is that we should substitute the actual value $c_{i,j,k}$ for each indeterminate $C_{i,j,k}$ in a polynomial produced by Deep Thought, and try to simplify such polynomials to make them easier to evaluate. For example, we should make use of the fact that if α and β are non-atoms with $\mathrm{left}(\alpha) = \mathrm{left}(\beta)$ and $\mathrm{right}(\alpha) = \mathrm{right}(\beta)$ then the polynomials g_α and g_β are nearly the same.

One small trick which has proved fruitful is to precompute the values of $\binom{m}{k}$ for small $|m|$ and k (say $0 \leqslant |m|, k \leqslant 20$), and use these precomputed values when evaluating Deep Thought polynomials.

If a generator a_i of G has finite order, we should work modulo this order when calculating with the exponent of a_i, to reduce the work involved in integer arithmetic. The calculation of the order of an element of G is described in Section 7.3, above.

9. *Hybrid collection*

We have discussed how to use Deep Thought polynomials to multiply and invert elements in the finitely generated nilpotent group G defined by the consistent power-commutator presentation (3). However, it may sometimes be more efficient (in terms of space or time) to adopt another strategy, such as collection from the left [**4**], detailed in Figure 6. (For ease of exposition, we shall assume that all exponents are non-negative. This is certainly the case for our implementation of collection from the left in (finite) p-groups.) Deep Thought tends to be best for groups of low class and high exponent, and we now describe how to combine Deep Thought with collection from the left to be able to multiply and invert more efficiently than one or the other approach could on its own.

The trick is to determine an integer $d \leqslant n$, so that Deep Thought deals efficiently with $G_d = \langle a_d, \ldots, a_n \rangle$. This may require some experimentation. We then calculate the Deep Thought polynomials f_{rs} for $d \leqslant s \leqslant r \leqslant n$, to be able to multiply $a_1^{x_1} \cdots a_n^{x_n}\, a_s^{y_s}$ efficiently using Deep Thought.

Now, in the process of collection, our hybrid collector is either in Deep Thought mode or in from-the-left mode, and is multiplying a word $x = a_1^{x_1} \cdots a_n^{x_n}$ in normal form (stored as an exponent vector in practice) times some other word v (stored on a stack in practice). (We have finished exactly when $v = 1$.)

Suppose we are in Deep Thought mode and v starts with a word of the form $v^* = a_s^{y_s}$. If $s \geqslant d$, then we remove v^* from v, calculate $x := x v^*$ using Deep Thought polynomials

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

(but not normalizing the result), and continue the process. If $s < d$ then we first normalize x using Deep Thought polynomials, switch to from-the-left mode, and continue the process.

Suppose we are in from-the-left mode and v starts with a word of the form $a_s^{y_s}$. If $s \geqslant d$, then we switch to Deep Thought mode, and continue the process. If $s < d$ then we remove the leading a_s from v, and collect this a_s into x using collection from the left in the ordinary way, and then continue the process.

10. *Implementations of Deep Thought*

We have implemented the Deep Thought algorithm and the hybrid collector described here (the hybrid for p-groups only), in the C programming language. Our Deep Thought implementation seems to work well for arbitrary finitely generated nilpotent groups up to about class 8 or more, and the hybrid collector can speed up multiplication in p-groups of much higher class. The Deep Thought C implementation has also been used by Paul Igodt and his colleagues to study various aspects of finitely generated torsion-free nilpotent groups (see, for example, [**1**]).

More recently, Deep Thought has been implemented by Wolfgang Merkwitz in the GAP system, and will thus be available for public use. This implementation is described in [**5**].

Merkwitz [**5**] comes to the conclusion that Deep Thought might be used to calculate in a p-group if $p \geqslant 11$. He gives a detailed account of the experiments leading to this conclusion. His evidence is very much in line with ours. He finds that, working on a Pentium PC operating at 166 MHz, the time taken to construct the Deep Thought polynomials in some p-groups of composition length 35 and nilpotency class 9 is about three seconds, regardless of the prime p. He finds that multiplication using Deep Thought out-performs collection (from the left) in evaluating the product of two random words by a factor of about 7 if $p = 7$, and a factor of about 10,000 if $p = 47$.

Merkwitz also finds that, for example, computing the derived subgroup of the groups he considers using collection takes 0.2 seconds, regardless of the prime. This presumably means that the collections performed are almost entirely trivial. The time required using Deep Thought is also, of course, effectively independent of the prime, but is almost five times as long. This emphasises the fact that, as currently implemented, collection from the left is out-performing Deep Thought multiplication in trivial calculations.

Acknowledgements

We thank Wolfgang Merkwitz, Joachim Neubüser, Werner Nickel, Charles Sims and Michael Vaughan-Lee for interesting and useful discussions. This research was partly supported by a European Union HCM grant in Computational Group Theory.

References

1. K. Dekimpe and P. Igodt, 'Computational aspects of affine representations for torsion free nilpotent groups via the Seifert construction', *J. Pure Appl. Algebra* 84 (1993) 165–190. 23

2. P. Hall, 'A contribution to the theory of groups of prime-power order', *Proc. London Math. Soc. (2)* 36 (1934) 29–95. 10, 10

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press

3. P. HALL, 'Nilpotent groups', Notes of lectures given at the Canadian Mathematical Congress 1957 Summer Seminar, in *The collected works of Philip Hall* (Clarendon Press, Oxford, 1988) pp. 415–462. 9, 9

4. C.R. LEEDHAM-GREEN and L.H. SOICHER, 'Collection from the left and other strategies', *J. Symbolic Comp.* 9 (1990) 665–675. 10, 22

5. W.W. MERKWITZ, 'Symbolische multiplikation in nilpotenten Gruppen mit Deep Thought', Diplomarbeit, RWTH Aachen, 1997. 23, 23

6. D.J.S. ROBINSON, *A course in the theory of groups* (Second Edition) (Springer, New York and Berlin, 1996). 19

7. M. SCHÖNERT *et al.*, 'GAP: groups, algorithms and programming', version 3, release 4, Lehrstuhl D für Mathematik, RWTH Aachen, 1994. 14

8. C.C. SIMS, *Computation with finitely presented groups* (Cambridge University Press, Cambridge, 1994). 9

C. R. Leedham-Green C.R.Leedham-Green@qmw.ac.uk
Leonard H. Soicher L.H.Soicher@qmw.ac.uk

School of Mathematical Sciences
Queen Mary and Westfield College
Mile End Road, London E1 4NS, U.K.

https://doi.org/10.1112/S1461157000000127 Published online by Cambridge University Press