BASIC COMMUTATORS AS RELATIONS: A COMPUTATIONAL PERSPECTIVE

PRIMOŽ MORAVEC AND ROBERT FITZGERALD MORSE

Abstract. In this paper we detail a computational proof that the normal closure of the set of commutators in any basic sequence of commutators of weight five in a free group F is equal to the fifth term in the lower central series of F. The weight less than five cases were answered affirmatively by Sims using computational methods. Sims was unable to prove the weight five case. In a recent paper by Jackson, Gaglione and Spellman, the weight five case has been settled affirmatively using commutator calculus. Using reduction results from Sims, we computationally show the result is true for the rank three case and using inductive results from Jackson et al. the result holds for arbitrary rank.

1. Introduction

In his the seminal paper "Verifying Nilpotence" [4], Charles Sims provides a method for verifying the nilpotence of a finitely presented group. The two main ingredients of Sims's method are finding maximal nilpotent quotients of finitely presented groups and term rewriting processes. His method has been recently applied in providing a positive answer that all 4-Engel groups are locally nilpotent [1]. One can also find a discussion and illustration of Sims's method in [2].

In [4], to illustrate his method for verifying nilpotence, Sims considers the problem of determining whether, given a basic sequence of commutators in a free group, the normal closure of those basic commutators of exactly weight k is equal to the kth term of the lower central series. He reduces the problem to determining whether the finitely presented group with basic commutators of weight exactly k as relations is nilpotent or not. Sims then applies the computational processes he earlier developed in the paper for verifying nilpotency to show this is true for $k \leq 4$. Sims was also able to show that for $k = 5$ and rank 2 the result holds.

2000 *Mathematics Subject Classification.* 20F18, 20J99.

Key words and phrases. Basic commutators, Verifying nilpotence.

A complete traditional "hand" proof for $k = 5$ case is given by Jackson, Gaglione, and Spellman [3]. The bulk of their paper deals with the rank 3 case. The purpose of this note is to revisit the analysis and computations of Sims in a modern setting utilizing computational tools, readily available to mathematicians, such as GAP and its packages, particularly Holt's Knuth-Bendix package KBMAG and Nickel's nq package for computing nilpotent quotients of finitely presented groups. Using the analysis of Sims we computationally prove the result for $k = 5$ ranks 2 and 3 and then apply the induction arguments in [3] to obtain the full result.

2. Preliminaries

Let $r > 1$ and let F be the free group on the ordered finite alphabet $X = \{x_1, x_2, \ldots, x_r\}$. Let $\mathfrak{C}$ be a basic sequence of commutators over the alphabet X that begins with the ordered alphabet. Set $\mathfrak{R}_{\mathfrak{C},k}$ to be the elements of the basic sequence $\mathfrak{C}$ of weight exactly k.

Proving the equality $(\mathfrak{R}_{\mathfrak{C},k})^F = \gamma_k(F)$, where $\gamma_k(F)$ is the kth term of the lower central series of F, is equivalent to showing the finitely presented group

$$G_k(r) = \langle X \mid \mathfrak{R}_{\mathfrak{C},k} \rangle$$

is nilpotent. The group $G_k(r)$ as defined is dependent on k, $\mathfrak{C}$ and the rank r. To apply computational methods in proving $G_k(r)$ is nilpotent, we remove its dependence on $\mathfrak{C}$. Hence if $G_k(r)$ can be shown to be nilpotent for one given basic sequence of commutators, it is nilpotent for any basic sequence of commutators. Sims proves this independence for $k \leq 9$:

Theorem 1 ([4])**.** *For $k \leq 9$, the isomorphism type of $G_k(r)$ depends on only k and r.*

To show that for a fixed k the group $G_k(r)$ is nilpotent, we need only to consider a fixed number of ranks. The following is again found in [4].

Proposition 2. *To show that $G_k(r)$ is nilpotent, it is enough to consider the case $|X| \leq k$.*

We are now able to state our main result

Theorem 3. *Let F be the free group on the finite set X and let $\mathfrak{R}_{\mathfrak{C},5}$ be the set of commutators of weight 5 from any basic sequence of commutators $\mathfrak{C}$ over X. Then $(\mathfrak{R}_{\mathfrak{C},5})^F = \gamma_5(F)$.*

Following Theorem 1 and Proposition 2, for $k = 5$ we need to verify the nilpotency of the following groups $G_5(r)$ for $2 \leq r \leq 5$ and a fixed basic sequence of commutators of weight r. In the next section we set up our computational environment to verify the nilpotence of $G_5(r)$.

3. Computational infrastructure

In this section we provide the computational infrastructure needed to in proving Theorem 3. This consists primarily of a several GAP functions and fixing our mathematical objects and GAP objects.

Our fixed basic sequence of commutators begin with the ordered alphabet X and the ordering of those commutators of equal weight is done lexicographically. We define a fully bracketed commutator over the symbol set X as follows: if $x \in X$ then $[x]$ is fully bracketed, and if c and d are fully bracketed commutators then $[c, d]$ is fully bracketed. This definition translates directly into GAP as a list of lists. We can then recursively compute the weight of such a commutator as given below.

```
##  Compute the weight for a fully bracketed commutator <comm>.
##
Weight :=
  function(comm)
    if Length(comm)=1 then return 1; fi;
    return Weight(comm[1])+Weight(comm[2]);
  end;
```

The actual elements of X can be any totally ordered data type in GAP. We choose integers so that we can use the natural lexicographical ordering of lists of lists of integers available in GAP. The following function creates fully bracketed commutators of a given weight from the lexicographical basic sequence.

```
##  Build the lexicographical basic sequence of commutators of
##  the given <weight> and rank <r>. The symbol set is the list
##  of integers [1..r]
##
BasicComSeq :=
  function(r, weight)
      if weight=1 then
          return List([1..r], x->[x]);
      fi;
      return
        Concatenation
          ( List
            ( Partitions(weight,2),
```

```
              p->Filtered
                 ( Cartesian
                   ( BasicComSeq(r, Maximum(p)),
                     BasicComSeq(r, Minimum(p))
                   ),
                   C->(p[1]=p[2]  and C[1]>C[2]) or
                      (p[1]<>p[2] and C[1][2]<=C[2])
                 )
            )
          );
  end;
```

The following function uses `BasicComSeq` give a full sequence up to the given weight.

```
## Build the lexicographical basic sequence of commutators
## of all weights up to <w> and rank <r>.
##
FullBasicComSeq :=
  function(r, w)
    return Concatenation(List([1..w], w->BasicComSeq(r,w)));
  end;
```

The basic commutators over the symbols `[1,2,3]` of weight at most 2 are

```
gap> PrintArray(FullBasicComSeq(3,2));
[ [      1 ],
  [      2 ],
  [      3 ],
  [  [ 2 ],  [ 1 ] ],
  [  [ 3 ],  [ 1 ] ],
  [  [ 3 ],  [ 2 ] ] ]
```

We use the integer entries of such fully bracketed commutators as indices for the generators of the free group to construct their associated words in the free group. The following function constructs this associated word.

```
##  The integer entries of the fully bracketed commutator
##  <comm> are associated with the same generator of the free
##  group <F>. The commutator is then evaluated to a word in <F>.
##
Eval :=
  function(F,comm)
    if Length(comm)=1 then return F.(comm[1]); fi;
    return Comm(Eval(F,comm[1]), Eval(F,comm[2]));
  end;
```

Here is an example to compute the commutator $[[f_3, f_1], f_1]$.

```
gap> F   := FreeGroup(3);;
gap> Wrd := Eval(F,[[[3],[1]],[1]]);
f1^-1*f3^-1*f1*f3*f1^-1*f3^-1*f1^-1*f3*f1^2
gap> Wrd = Comm(Comm(F.3,F.1),F.1);
true
```

The function `Eval` is used to create a presentation for $G_5(2)$

```
gap> F := FreeGroup(2);;
gap> # Evaluate as words in the free group F the words in
gap> # the basic sequence of length 5.
gap> R := List(BasicComSeq(2,5),c->Eval(F,c));;
gap> G_5_2 := F/R;
<fp group on the generators [ f1, f2 ]>
```

The group `G_5_2` has a maximal nilpotent quotient of class 4 and we give its simplified presentation for later comparison.

```
gap> NilpotencyClassOfGroup(NilpotentQuotient(G_5_2));
4
gap> PresentationFpGroup(SimplifiedFpGroup(G_5_2));
<presentation with 2 gens and 6 rels of total length 154>
```

To show that $G_5(2)$ is nilpotent we create a group isomorphic to $G_5(2)$ with generators associated with the commutators in a basic sequence of up to weight 4. Here is the general situation. Let $\mathfrak{C}$ be a basic sequence of commutators over the set $X = \{x_1, \ldots, x_r\}$. Let $x_1, \ldots, x_r, c_{r+1}, \ldots, c_n$ be basic sequence of commutators up to weight less then k with $x_1, \ldots, x_r$ being an ordering of the generating set X and $n = \sum_{w=1}^{k-1} W(r, w)$, where $W(r, w)$ is Witt's formula for computing the number of basic commutators of weight w of rank r. Set

$$\mathfrak{G} = \langle x_1, \ldots, x_n \mid x_{r+1} = c_{r+1}, \ldots, x_n = c_n, \mathfrak{R}_{\mathfrak{C},k} \rangle.$$

Applying Tietze transformations we see that our presentation for $\mathfrak{G}$ simplifies to $G_k(r)$. Sims's procedure for verifying the nilpotency of $G_k(r)$ requires us to show that $G_k(r)$ has a maximal nilpotent quotient and that $\mathfrak{G}$ has a consistent polycyclic presentation. The group $G_5(2)$ has a maximal nilpotent quotient of class 4. What remains to be done is to set up $\mathfrak{G}$ for this case and apply the Knuth-Bendix rewriting procedure to show it has consistent polycyclic presentation.

We can build $\mathfrak{G}$ using the infrastructure in place. Moreover, we can express each basic commutator on $\mathfrak{G}$ of weight 2 or greater as a commutator in two generators. The following GAP function does this recursive substitution.

```
##  Express each basic commutator as a fully bracketed
##  commutator with indices in the lexicographical basic
##  sequence.
```

```
##
Recursive :=
  function(r,w)
    if w=1 then return BasicComSeq(r,w); fi;
    return
      List
      ( BasicComSeq(r,w), c->
        [ [ Position
            ( FullBasicComSeq(r, Weight(c[1])),
              c[1]
            )
          ],
          [ Position
            ( FullBasicComSeq(r, Weight(c[2])),
              c[2]
            )
          ]
        ]
      );
end;
```

We now create $\mathfrak{G}$ as a GAP object.

```
gap> # Build the generators of the group.
gap> gencoms := Concatenation(
>               List([1..4], w->Recursive(2,w)));;
gap> # Create the free group and the relations for each
gap> # generators
gap> F:= FreeGroup(Length(gencoms));;
gap> rels := List([1..Length(gencoms)],
>                 i->F.(i)/Eval(F,gencoms[i]));;
gap> # Create the relations of G_5(2) and append them
gap> # to the relations already created and form the FP group.
gap> Append(rels,List(Recursive(2,5),c->Eval(F,c)));;
gap> G := F/rels;
<fp group on the generators [ f1, f2, f3, f4, f5, f6, f7, f8 ]>
gap> # Gives a similar simplified presentation as G_5_2
gap> PresentationFpGroup(SimplifiedFpGroup(G));
<presentation with 2 gens and 6 rels of total length 154>
```

From the GAP group `G` we can create a rewriting system and check to see if it is confluent. However, as the group is presented neither the rewriting algorithms in `GAP` or using the `KBMAG` package gives an answer. In the next section we discuss some strategies to show that `G` has a consistent polycyclic presentation.

4. Knuth Bendix rewriting procedure

Let $G = F/R$ be a finitely presented group. The Knuth-Bendix procedure is designed to construct a normal form for the elements of G in terms of the generators of F. The normal form of an element is the least word in the generators of F and their inverses that represents the element in G, with respect to a specific ordering on the set of all words in F. This procedure has its ring theoretic analogue, namely the Gröbner bases of ideals of polynomial rings.

KBMAG offers four types of orderings used in rewriting systems. These are shortlex, recursive, wtlex and wreathprod. Wreath product orderings are the ones on which reduction to normal form in polycyclic groups is based ([5], page 395). Thus they are particularly well suited for finding rewriting systems of polycyclic groups. A special case of wreath product orderings is the so called recursive ordering. This can be described as follows. Let u and v be strings in the generators of G. If one of u and v, say v, is empty, then $u \succeq v$. Otherwise, let $u = u'a$ and $v = v'b$, where a and b are generators. Then $u \succ v$ if and only if one of the following holds:

(1) $a = b$ and $u' \succ v'$,
(2) $a \succ b$ and $u \succ v'$,
(3) $b \succ a$ and $u' \succ v$.

It becomes clear that the recursive ordering is the one that follows the spirit of the concept of basic commutators. Thus we use this ordering as our primary choice.

We order the generators of the group in reverse order. This is motivated by a similar trick used commonly in computations of Gröbner bases.

Using the GAP infrastructure in Section 3, we create the same group $\mathfrak{G}$ as before but reverse the ordering of the generators of the free group in creating it. We need only one more function to help us relabel the basic commutators in this reverse ordering.

```
##  Treat each numeric entry in each commutator as a difference
##  value from which a new number (label) can be created.
##
Relabel :=
  function(d,comm)
    if Weight(comm)=1 then return [d-comm[1]]; fi;
    return [Relabel(d,comm[1]),Relabel(d,comm[2])];
  end;
```

We now create $\mathfrak{G}$ in GAP whose recursive presentation is ordered in reverse for $r = 2$. We know that there are 8 commutators that will be generators of $\mathfrak{G}$ from our previous analysis.

```
gap> F := FreeGroup(8);;
gap> # Build the generating set of commutators
gap> gencoms := Concatenation(List(([1..4]), w->Recursive(2,w)));;
gap> # For r=2 we will have 1-8 generators -- reverse the labels
gap> gencoms := Reversed(List(gencoms,c->Relabel(9,c)));;
gap> rels := List([1..8], i->F.(i)/Eval(F,gencoms[i]));;
gap> # Build the relations of weight 5
gap> coms := List(Recursive(2,5),c->Relabel(9,c));;
gap> Append(rels, List(coms, c->Eval(F,c)));;
gap> G := F/rels;
<fp group on the generators [ f1, f2, f3, f4, f5, f6, f7, f8 ]>
```

We can do some simple checks to see that we have the right group. It has a maximal nilpotent quotient of class 4 and after applying Tietze transformations we arrive back to a similar presentation with $G_5(2)$.

```
gap> # The maximal nilpotent quotient is 4
gap> NilpotencyClassOfGroup(NilpotentQuotient(G));
4
gap> PresentationFpGroup(SimplifiedFpGroup(G));
<presentation with 2 gens and 6 rels of total length 154>
```

We now apply the Knuth-Bendix procedure to the GAP object `G` using the recursive ordering and it finds a confluent system of relations.

```
gap> R := KBMAGRewritingSystem(G);;
gap> SetOrderingOfKBMAGRewritingSystem(R,"recursive");;
gap> KnuthBendix(R);
true
```

The set up for the rank 3 case follows exactly the same as the rank 2 case. There are 32 commutators that will be the generators of $\mathfrak{G}$ with 48 relations which are the basic commutators of weight 5 and rank 3.

```
gap> F := FreeGroup(32);;
gap> # Build the generating set of commutators
gap> gencoms := Concatenation(List(([1..4]), w->Recursive(3,w)));;
gap> # For r=3 we will have 1-32 generators -- reverse the labels
gap> gencoms := Reversed(List(gencoms,c->Relabel(33,c)));;
gap> rels := List([1..32], i->F.(i)/Eval(F,gencoms[i]));;
gap> # Build the relations of weight 5
gap> coms := List(Recursive(3,5),c->Relabel(33,c));;
gap> Append(rels, List(coms, c->Eval(F,c)));;
gap> G := F/rels;
<fp group with 32 generators>
```

Our check again works out.

```
gap> # The maximal nilpotent quotient is 4
gap> NilpotencyClassOfGroup(NilpotentQuotient(G));
4
gap> PresentationFpGroup(SimplifiedFpGroup(G));
<presentation with 3 gens and 48 rels of total length 1276>
```

We now have to set up some more parameters in KBMAG in particular a bound on the length of the equations and the number of equations and states allowed before starting the rewriting process.

```
gap> R := KBMAGRewritingSystem(G);;
gap> SetOrderingOfKBMAGRewritingSystem(R,"recursive");;
gap> O:=OptionsRecordOfKBMAGRewritingSystem(R);;
gap> O.maxstoredlen:=[50,50];;
gap> O.maxstates:=2^20;;
gap> O.maxeqns:=2^20;;
gap> KnuthBendix(R);
#WARNING: Because of the control parameters you set,
#         the system may
#         not be confluent. Unbind the parameters and
#         re-run KnuthBendix to check!
#I  System computed is NOT confluent.
false
gap> KnuthBendix(R);
true
```

This finishes in about 8 minutes or so. In the next section we will use this result along with the analytical work from Jackson, Gaglione, and Spellman [3] to complete the proof of Theorem 3.

5. Proof of Theorem 3 and Conclusion

We complete the proof of Theorem 3. Our computations in Section 4 show that result is true for $r = 3$. Theorems 4.53 and 5.10 in [3] give the result for $r = 4$ $r = 5$ respectively using the inductive result on $r = 3$ and $r = 4$ respectively. This completes the proof of the theorem.

Our computations above show that finding a maximal nilpotent quotient is relatively automatic to compute, at least in the finitely presented groups we are working with. However, the "naive" use of rewriting procedures provide no results even for the $r = 2$ case. However with some finesse we are able to settle the question for $r = 3$. This case take up nearly 20 pages in [3]. While even the $r = 2$ case took some work, Sims in [4] was able to make this computation. For $r = 3$ we assume it was out of reach for the computational power available to him at the time.

We attempted the weight 6 and rank 2 case and made no progress at all. This is still an open question as is all other weights greater than 5. Possible other approaches are to show that $G_k(r)$ is a homomorphic image of a nilpotent group or use the Sievers's Free Group Algorithms package, `FGA`, to assist in making the computations found in [3].

References

[1] George Havas and M. R. Vaughan-Lee. 4-Engel groups are locally nilpotent. *Internat. J. Algebra Comput.*, 15(4):649–682, 2005.

[2] 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.

[3] D. A. Jackson, A. M. Gaglione, and D. Spellman. Weight five basic commutators as relators. Submitted.

[4] Charles C. Sims. Verifying nilpotence. *J. Symbolic Comput.*, 3(3):231–247, 1987.

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

Department of Mathematics, University of Ljubljana, Jadranska 21, 1000 Ljubljana, Slovenia

E-mail address: `primoz.moravec@fmf.uni-lj.si`

URL: `www.fmf.uni-lj.si/~moravec`

Department of Electrical Engineering and Computer Science, University of Evansville, Evansville IN 47722 USA

E-mail address: `rfmorse@evansville.edu`