FGA

Free Group Algorithms

1.5.0

4 April 2023

Christian Sievers

Note: This version of FGA is a fork of the original FGA, maintained by the GAP Team.

Christian Sievers
Email: c.sievers@tu-bs.de

Contents

Chapter 1

Introduction

1.1 Overview

This manual describes the FGA (*Free Group Algorithms*) package, a GAP package for computations with finitely generated subgroups of free groups.

This package allows you to (constructively) test membership and conjugacy, and to compute free generators, the rank, the index, normalizers, centralizers, and intersections where the groups involved are finitely generated subgroups of free groups. In addition, it provides generators and a finite presentation for the automorphism group of a finitely generated free group and allows to write any such automorphism as word in these generators.

See Chapter 'Functionality of the FGA package' for details.

Chapter 'Installing and loading the FGA package' explains how to install and load the FGA package.

1.2 Implementation and background

The methods which are used work mainly with inverse finite automata, a variation of an idea known from theoretical computer science. An inverse finite automaton is a finite state automaton over a symmetric alphabet, i.e. one in which every letter has an inverse, such that each transition between two states for a letter corresponds to a transition in the opposite direction for the inverse letter.

Most of these techniques are described in Chapter 4 of [Sim94], where the same concept is called coset automaton. The method to obtain this automaton is called basic coset enumeration, and in fact it is coset enumeration where only important cosets are defined. Here a coset Gg is called important when there are words w and v such that wv is reduced and denotes an element of G and w denotes an element of Gg.

In [BMMW00], the connection between finitely generated subgroups of free groups and inverse finite automata is used to transfer results about the space complexity of problems concerning inverse finite automata to analogous results about finitely generated subgroups of free groups.

Chapter 6 of [Sim94] describes the Reidemeister–Schreier procedure and a variant called extended coset enumeration which yields a presentation in the given generators. The FGA package uses a variation thereof for its constructive membership test: it leaves out the part of the algorithm that fills in relations and interprets the resulting extended coset table differently. This algorithm might be called extended basic coset enumeration.

Some word oriented algorithms in the FGA package use basic facts about free groups. These can, for example, be found in [LS77].

The presentation of the automorphism groups follows [Neu33]. The algorithm for writing an automorphism in the generators works first at the level of Nielsen generators and uses relations from [Nie24].

The theoretical background for most of this implementation is explained in [Sie03].

1.3 Integration of the package

The FGA package mainly installs new methods for operations that are already known to GAP. They overlap with methods in the GAP library in the case of groups of finite index. In this case, GAPs methods are usually faster, and the FGA package tries to recognize such cases and to refer to GAP.

The methods of the FGA package will only be selected when the groups involved know they are finitely generated. This may not always be the case for groups that were not created by methods of the FGA package. In such a case you will get a `no method found` error, or GAP may try a coset enumeration that stops with the message `the coset enumeration has defined more than 256000 cosets`. You may then call `GeneratorsOfGroup`, and try again.

Please inform the package author if you observe any remaining problems.

1.4 License

Like the GAP system itself, the FGA package is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You can find the GNU General Public License in the file `COPYING` of the FGA package, and also in the file `GPL` in the `etc` directory of the main GAP distribution, or see `http://www.gnu.org/licenses/gpl.html`.

Chapter 2

Functionality of the FGA package

This chapter describes methods available from the FGA package.

In the following, let *f* be a free group created by `FreeGroup(n)`, and let *u*, *u1* and *u2* be finitely generated subgroups of *f* created by `Group` or `Subgroup`, or computed from some other subgroup of *f*. Let *elm* be an element of *f*.

For example:

Example

```
gap> f := FreeGroup( 2 );
<free group on the generators [ f1, f2 ]>
gap> u := Group( f.1^2, f.2^2, f.1*f.2 );
Group([ f1^2, f2^2, f1*f2 ])
gap> u1 := Subgroup( u, [f.1^2, f.1^4*f.2^6] );
Group([ f1^2, f1^4*f2^6 ])
gap> elm := f.1;
f1
gap> u2 := Normalizer( u, elm );
Group([ f1^2 ])
```

2.1 New operations for free groups

These new operations are available for finitely generated subgroups of free groups:

2.1.1 FreeGeneratorsOfGroup

▷ `FreeGeneratorsOfGroup(u)` (attribute)

returns a list of free generators of the finitely generated subgroup *u* of a free group.

The elements in this list form an N-reduced set. In addition to being a free (and thus minimal) generating set for *u*, this means that whenever *v1*, *v2* and *v3* are elements or inverses of elements of this list, then

- $v1\,v2 \neq 1$ implies $|v1\,v2| \geq \max(|v1|,|v2|)$, and
- $v1\,v2 \neq 1$ and $v2\,v3 \neq 1$ implies $|v1\,v2\,v3| > |v1| - |v2| + |v3|$

hold, where $|.|$ denotes the word length.

2.1.2 RankOfFreeGroup

▷ `RankOfFreeGroup(u)` (attribute)
▷ `Rank(u)` (operation)

returns the rank of the finitely generated subgroup `u` of a free group.

2.1.3 CyclicallyReducedWord

▷ `CyclicallyReducedWord(elm)` (operation)

returns the cyclically reduced form of `elm`.

2.2 Method installations

This section lists operations that are already known to GAP. FGA installs new methods for them so that they can also be used with free groups. In cases where FGA installs methods that are usually only used internally, user functions are shown instead.

2.2.1 Normalizer

▷ `Normalizer(u1, u2)` (operation)
▷ `Normalizer(u, elm)` (operation)

The first variant returns the normalizer of the finitely generated subgroup `u2` in `u1`.

The second variant returns the normalizer of $\langle$`elm`$\rangle$ in the finitely generated subgroup `u` (see `Normalizer` (**Reference: Normalizer**) in the Reference Manual).

2.2.2 RepresentativeAction

▷ `RepresentativeAction(u, d, e)` (operation)
▷ `IsConjugate(u, d, e)` (operation)

`RepresentativeAction` returns an element $r \in$ `u`, where `u` is a finitely generated subgroup of a free group, such that $d^r =$ `e`, or fail, if no such `r` exists. `d` and `e` may be elements or subgroups of `u`.

`IsConjugate` returns a boolean indicating whether such an element `r` exists.

2.2.3 Centralizer

▷ `Centralizer(u, u2)` (operation)
▷ `Centralizer(u, elm)` (operation)

returns the centralizer of `u2` or `elm` in the finitely generated subgroup `u` of a free group.

2.2.4 Index

▷ Index(*u1, u2*) (operation)
▷ IndexNC(*u1, u2*) (operation)

return the index of *u2* in *u1*, where *u1* and *u2* are finitely generated subgroups of a free group. The first variant returns fail if *u2* is not a subgroup of *u1*, the second may return anything in this case.

2.2.5 Intersection

▷ Intersection(*u1, u2, \dots*) (function)

returns the intersection of *u1* and *u2*, where *u1* and *u2* are finitely generated subgroups of a free group.

2.2.6 \in

▷ \in(*elm, u*) (method)

tests whether *elm* is contained in the finitely generated subgroup *u* of a free group.

2.2.7 IsSubgroup

▷ IsSubgroup(*u1, u2*) (function)

tests whether *u2* is a subgroup of *u1*, where *u1* and *u2* are finitely generated subgroups of a free group.

2.2.8 \=

▷ \=(*u1, u2*) (method)

test whether the two finitely generated subgroups *u1* and *u2* of a free group are equal.

2.2.9 MinimalGeneratingSet

▷ MinimalGeneratingSet(*u*) (attribute)
▷ SmallGeneratingSet(*u*) (attribute)
▷ GeneratorsOfGroup(*u*) (attribute)

return generating sets for the finitely generated subgroup *u* of a free group. MinimalGeneratingSet and SmallGeneratingSet return the same free generators as FreeGeneratorsOfGroup, which are in fact a minimal generating set. GeneratorsOfGroup also returns these generators, if no other generators were stored at creation time.

2.3 Constructive membership test

It is not only possible to test whether an element is in a finitely generated subgroup of free group, this can also be done constructively. The idiomatic way to do so is by using a homomorphism.

Here is an example that computes how to write `f.1^2` in the generators `a=f1^2*f2^2` and `b=f.1^2*f.2`, checks the result, and then tries to write `f.1` in the same generators:

Example

```
gap> f := FreeGroup( 2 );
<free group on the generators [ f1, f2 ]>
gap> ua := f.1^2*f.2^2;; ub := f.1^2*f.2;;
gap> u := Group( ua, ub );;
gap> g := FreeGroup( "a", "b" );;
gap> hom := GroupHomomorphismByImages( g, u,
>             GeneratorsOfGroup(g),
>             GeneratorsOfGroup(u) );
[ a, b ] -> [ f1^2*f2^2, f1^2*f2 ]
gap> # how can f.1^2 be expressed?
gap> PreImagesRepresentative( hom, f.1^2 );
b*a^-1*b
gap> last ^ hom; # check this
f1^2
gap> ub * ua^-1 * ub; # another check
f1^2
gap> PreImagesRepresentative( hom, f.1 ); # try f.1
fail
gap> f.1 in u;
false
```

There are also lower level operations to get the same results.

2.3.1 AsWordLetterRepInGenerators

▷ `AsWordLetterRepInGenerators(`*elm*`, `*u*`)` (operation)
▷ `AsWordLetterRepInFreeGenerators(`*elm*`, `*u*`)` (operation)

return a letter representation (see Section (**Reference: Representations for Associative Words**) in the GAP Reference Manual) of the given *elm* relative to the generators the group was created with or the free generators as returned by `FreeGeneratorsOfGroup`.

Continuing the above example:

Example

```
gap> AsWordLetterRepInGenerators( f.1^2, u );
[ 2, -1, 2 ]
gap> AsWordLetterRepInFreeGenerators( f.1^2, u );
[ 2 ]
```

This means: to get `f.1^2`, multiply the second of the given generators with the inverse of the first and again with the second; or just take the second free generator.

2.4 Automorphism groups of free groups

The FGA package knows presentations of the automorphism groups of free groups. It also allows to express an automorphism as word in the generators of these presentations. This sections repeats the GAP standard methods to do so and shows functions to obtain the generating automorphisms.

2.4.1 AutomorphismGroup

▷ `AutomorphismGroup(`*u*`)` (attribute)

returns the automorphism group of the finitely generated subgroup *u* of a free group.

Only a few methods will work with this group. But there is a way to obtain an isomorphic finitely presented group:

2.4.2 IsomorphismFpGroup

▷ `IsomorphismFpGroup(`*group*`)` (attribute)

returns an isomorphism of *group* to a finitely presented group. For automorphism groups of free groups, the FGA package implements the presentations of [Neu33]. The finitely presented group itself can then be obtained with the command `Range`.

Here is an example:

Example

```
gap> f := FreeGroup( 2 );;
gap> a := AutomorphismGroup( f );;
gap> iso := IsomorphismFpGroup( a );;
gap> Range( iso );
<fp group on the generators [ O, P, U ]>
```

To express an automorphism as word in the generators of the presentation, just apply the isomorphism obtained from `IsomorphismFpGroup`.

Example

```
gap> aut := GroupHomomorphismByImages( f, f,
>              GeneratorsOfGroup( f ), [ f.1^f.2, f.1*f.2 ] );
[ f1, f2 ] -> [ f2^-1*f1*f2, f1*f2 ]
gap> ImageElm( iso, aut );
O^2*U*O*P^-1*U
```

It is also possible to use `aut^iso` or `Image( iso, aut )`. Using `Image` will perform additional checks on the arguments.

The FGA package provides a simpler way to create endomorphisms:

2.4.3 FreeGroupEndomorphismByImages

▷ `FreeGroupEndomorphismByImages(`*g*`, `*imgs*`)` (function)

returns the endomorphism that maps the free generators of the finitely generated subgroup *g* of a free group to the elements listed in *imgs*. You may then apply `IsBijective` to check whether it is an automorphism.

The following functions return automorphisms that correspond to the generators in the presentation:

2.4.4 FreeGroupAutomorphismsGeneratorO

▷ `FreeGroupAutomorphismsGeneratorO(`*group*`)` (function)
▷ `FreeGroupAutomorphismsGeneratorP(`*group*`)` (function)
▷ `FreeGroupAutomorphismsGeneratorU(`*group*`)` (function)
▷ `FreeGroupAutomorphismsGeneratorS(`*group*`)` (function)
▷ `FreeGroupAutomorphismsGeneratorT(`*group*`)` (function)
▷ `FreeGroupAutomorphismsGeneratorQ(`*group*`)` (function)
▷ `FreeGroupAutomorphismsGeneratorR(`*group*`)` (function)

return the automorphism which maps the free generators $[x_1, x_2, \ldots, x_n]$ of *group* to

O: $[x_1^{-1}, x_2, \ldots, x_n]$ $(n \geq 1)$

P: $[x_2, x_1, x_3, \ldots, x_n]$ $(n \geq 2)$

U: $[x_1 x_2, x_2, x_3, \ldots, x_n]$ $(n \geq 2)$

S: $[x_2^{-1}, x_3^{-1}, \ldots, x_n^{-1}, x_1^{-1}]$ $(n \geq 1)$

T: $[x_2, x_1^{-1}, x_3, \ldots, x_n]$ $(n \geq 2)$

Q: $[x_2, x_3, \ldots, x_n, x_1]$ $(n \geq 2)$

R: $[x_2^{-1}, x_1, x_3, x_4, \ldots, x_{n-2}, x_n x_{n-1}^{-1}, x_{n-1}^{-1}]$ $(n \geq 4)$

Chapter 3

Installing and loading the FGA package

3.1 Installing the FGA package

The installation of the FGA package follows standard GAP rules. So the standard method is to unpack the archive into the pkg directory of your GAP distribution. This will create an fga subdirectory.

For other non-standard options please see Chapter (**Reference: Installing a GAP Package**) in the GAP Reference Manual.

3.2 Loading the FGA package

The FGA package is configured to autoload, so its functionality is usually available once GAP is started.

If GAP does not autoload, you can request the package with the LoadPackage command like this:

Example

```
gap> LoadPackage( "fga" );
-------------------------------------------------------------------------------
Loading  FGA 1.5.0-DEV (Free Group Algorithms)
by Christian Sievers (c.sievers@tu-bs.de).
maintained by:
   The GAP Team (support@gap-system.org).
Homepage: https://gap-packages.github.io/fga/
Report issues at https://github.com/gap-packages/fga/issues
-------------------------------------------------------------------------------
true
```

You will not see the banner if FGA has already been loaded.

The LoadPackage command and ways to disable autoloading are described in Section (**Reference: Loading a GAP Package**) in the GAP Reference Manual.

References

[BMMW00] J.-C. Birget, S. Margolis, J. Meakin, and P. Weil. PSPACE-complete problems for subgroups of free groups and inverse finite automata. *Theoretical Computer Science*, 242:247–281, 2000. 3

[LS77] Roger C. Lyndon and Paul E. Schupp. *Combinatorial Group Theory*. Springer, 1977. 4

[Neu33] Bernd Neumann. Die Automorphismengruppe der freien Gruppen. *Math. Annalen*, 107:367–386, 1933. 4, 9

[Nie24] J. Nielsen. Die Isomorphismengruppe der freien Gruppen. *Math. Annalen*, 91:169–209, 1924. 4

[Sie03] Christian Sievers. Algorithmen fur freie Gruppen. Diplomarbeit, TU Braunschweig, 2003. 4

[Sim94] C. C. Sims. *Computation with Finitely Presented Groups*, volume 48 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1994. 3

Index