

# Twisted Conjugacy

## Computation with twisted conjugacy classes

### 3.3.1

6 May 2026

**Sam Tertoo**

**Sam Tertoo** Email: [sam.tertooy@kuleuven.be](mailto:sam.tertooy@kuleuven.be)

Homepage: <https://stertooy.github.io/>

Address: Wiskunde

KU Leuven, Kulak Kortrijk Campus

Etienne Sabbelaan 53

8500 Kortrijk

Belgium

## Abstract

The TwistedConjugacy package provides methods for solving the twisted conjugacy problem (including the "search" and "multiple" variants) and for computing Reidemeister classes, numbers, spectra, and zeta functions. It also includes utility functions for working with (double) cosets, group homomorphisms, and group derivations.

These methods are primarily designed for use with finite groups and with PcpGroups (finite or infinite) provided by the Polycyclic package.

## Copyright

© 2020–2026 Sam Tertooy

The TwistedConjugacy package is free software, it may be redistributed and/or modified under the terms and conditions of the [GNU Public License Version 2](#) or (at your option) any later version.

## Acknowledgements

This documentation was created using the GAPDoc and AutoDoc packages.

# Contents

<b>1</b>	<b>The TwistedConjugacy package</b>	<b>5</b>
1.1	Installation . . . . .	5
1.2	Loading . . . . .	5
1.3	Citing . . . . .	6
1.4	Support . . . . .	6
<b>2</b>	<b>Mathematical background</b>	<b>7</b>
<b>3</b>	<b>Twisted conjugation</b>	<b>8</b>
3.1	The twisted conjugation action . . . . .	8
3.2	The twisted conjugacy (search) problem . . . . .	8
3.3	The multiple twisted conjugacy (search) problem . . . . .	9
<b>4</b>	<b>Twisted conjugacy classes</b>	<b>10</b>
4.1	Creating a twisted conjugacy class . . . . .	10
4.2	Operations on twisted conjugacy classes . . . . .	10
4.3	Calculating all twisted conjugacy classes . . . . .	12
<b>5</b>	<b>Reidemeister numbers and spectra</b>	<b>13</b>
5.1	Reidemeister numbers . . . . .	13
5.2	Reidemeister spectra . . . . .	13
<b>6</b>	<b>Reidemeister zeta and generating functions</b>	<b>15</b>
6.1	Decomposing Reidemeister number sequences . . . . .	15
6.2	Reidemeister zeta functions . . . . .	15
6.3	Reidemeister generating functions . . . . .	16
<b>7</b>	<b>Cosets of PcpGroups</b>	<b>18</b>
7.1	Right cosets . . . . .	18
7.2	Double cosets . . . . .	19
<b>8</b>	<b>Group homomorphisms</b>	<b>21</b>
8.1	Representatives of homomorphisms between groups . . . . .	21
8.2	Coincidence and fixed point groups . . . . .	22
8.3	Induced and restricted group homomorphisms . . . . .	22

<b>9</b>	<b>Group derivations</b>	<b>24</b>
9.1	Creating group derivations . . . . .	24
9.2	Operations for group derivations . . . . .	25
9.3	Images of group derivations . . . . .	26
<b>10</b>	<b>Affine actions</b>	<b>28</b>
10.1	Creating an affine action . . . . .	28
10.2	Operations for affine actions . . . . .	28
10.3	Operations on orbits of affine actions . . . . .	29
	<b>References</b>	<b>32</b>
	<b>Index</b>	<b>33</b>

# Chapter 1

## The TwistedConjugacy package

This is the manual for the GAP 4 package TwistedConjugacy version 3.3.1, developed by Sam Tertooy.

### 1.1 Installation

If you are using GAP version 4.15.0 or newer, then TwistedConjugacy should be installed by default.

If this is not the case, but the PackageManager package is installed and loaded, you can install TwistedConjugacy from within a GAP session using InstallPackage (**PackageManager: InstallPackage**).

Example

```
gap> InstallPackage( "TwistedConjugacy" );
...
true
```

Alternatively, you can download TwistedConjugacy as a .tar.gz archive [here](#). After extracting, you should place it in a suitable pkg folder. For example, on a Debian-based Linux distribution (e.g. Ubuntu, Mint), you can place it in \$HOME/.gap/pkg (recommended) which makes it available for just yourself, or in the GAP installation directory (gap-X.Y.Z/pkg) which makes it available for all users.

You can use the following command to efficiently install the package for yourself:

Command

```
wget -q0 - https://[...].tar.gz | tar xzf - --one-top-level=$HOME/.gap/pkg
```

### 1.2 Loading

Once installed, loading TwistedConjugacy can be done by using LoadPackage (**Reference: LoadPackage**).

Example

```
gap> LoadPackage( "TwistedConjugacy" );
...
true
```

## 1.3 Citing

If you use the `TwistedConjugacy` package in your research, we would love to hear about your work via an email to the address [sam.tertooy@kuleuven.be](mailto:sam.tertooy@kuleuven.be). If you have used the `TwistedConjugacy` package in the preparation of a paper and wish to refer to it, please cite it as described below.

In Bib $\text{\TeX}$ :

```

                                BibTeX
@misc{TC3.3.1,
  author    = {Tertooy, Sam},
  title     = {{TwistedConjugacy,
                Computation with twisted conjugacy classes,
                Version 3.3.1}},
  note      = {GAP package},
  year      = {2026},
  howpublished = {\url{https://stertooy.github.io/TwistedConjugacy}}
}
```

In BibLa $\text{\TeX}$ :

```

                                BibLaTeX
@software{TC3.3.1,
  author    = {Tertooy, Sam},
  title     = {TwistedConjugacy},
  subtitle  = {Computation with twisted conjugacy classes},
  version   = {3.3.1},
  note      = {GAP package},
  year      = {2026},
  url       = {https://stertooy.github.io/TwistedConjugacy}
}
```

## 1.4 Support

If you encounter any problems, please submit them to the [issue tracker](#). If you have any questions on the usage or functionality of `TwistedConjugacy`, you may contact me via email at [sam.tertooy@kuleuven.be](mailto:sam.tertooy@kuleuven.be).

Bugs in `GAP`, in this package, or in any other package used directly or indirectly, may cause functions provided by `TwistedConjugacy` to produce errors or incorrect results. To help detect such issues, you can enable internal checks by setting the variable `TWC.ASSERT` to `true`. Note that this will come at the cost of reduced performance.

For additional safety, you can enable `GAP`'s built-in assertion features by calling `SetAssertionLevel( lev )` (we recommend setting `lev` to 2) and, when working with `PcpGroups`, setting the variables `CHECK_CENT@Polycyclic`, `CHECK_IGS@Polycyclic` and `CHECK_INTSTAB@Polycyclic` to `true`.

## Chapter 2

# Mathematical background

Let  $G$  and  $H$  be groups and let  $\varphi$  and  $\psi$  be group homomorphisms from  $H$  to  $G$ . The pair  $(\varphi, \psi)$  induces a (right) group action of  $H$  on  $G$  given by the map

$$G \times H \rightarrow G: (g, h) \mapsto \varphi(h)^{-1} g \psi(h).$$

This group action is called  $(\varphi, \psi)$ -*twisted conjugation*. The orbits are called *Reidemeister classes* or *twisted conjugacy classes*, and the number of Reidemeister classes is called the *Reidemeister number*  $R(\varphi, \psi)$  of the pair  $(\varphi, \psi)$ . The stabiliser of the identity  $1_G$  under the  $(\varphi, \psi)$ -twisted conjugacy action of  $H$  is exactly the *coincidence group*

$$\text{Coin}(\varphi, \psi) = \{ h \in H \mid \varphi(h) = \psi(h) \}.$$

Generalising this, the stabiliser of any  $g \in G$  is the coincidence group  $\text{Coin}(\iota_g \varphi, \psi)$ , with  $\iota_g$  the inner automorphism of  $G$  that conjugates by  $g$ .

Twisted conjugacy originates in Reidemeister-Nielsen fixed point and coincidence theory, where it serves as a tool for studying fixed and coincidence points of continuous maps between topological spaces. Below, we briefly illustrate how and where this algebraic notion arises when studying coincidence points. Let  $X$  and  $Y$  be topological spaces with universal covers  $p: \tilde{X} \rightarrow X$  and  $q: \tilde{Y} \rightarrow Y$  and let  $\mathcal{D}(X), \mathcal{D}(Y)$  be their covering transformations groups. Let  $f, g: X \rightarrow Y$  be continuous maps with lifts  $\tilde{f}, \tilde{g}: \tilde{X} \rightarrow \tilde{Y}$ . By  $f_*: \mathcal{D}(X) \rightarrow \mathcal{D}(Y)$ , denote the group homomorphism defined by  $\tilde{f} \circ \gamma = f_*(\gamma) \circ \tilde{f}$  for all  $\gamma \in \mathcal{D}(X)$ , and let  $g_*$  be defined similarly. The set of coincidence points  $\text{Coin}(f, g)$  equals the union

$$\text{Coin}(f, g) = \bigcup_{\alpha \in \mathcal{D}(Y)} p(\text{Coin}(\tilde{f}, \alpha \tilde{g})).$$

For any two elements  $\alpha, \beta \in \mathcal{D}(Y)$ , the sets  $p(\text{Coin}(\tilde{f}, \alpha \tilde{g}))$  and  $p(\text{Coin}(\tilde{f}, \beta \tilde{g}))$  are either disjoint or equal. Moreover, they are equal if and only if there exists some  $\gamma \in \mathcal{D}(X)$  such that  $\alpha = f_*(\gamma)^{-1} \circ \beta \circ g_*(\gamma)$ , which is exactly the same as saying that  $\alpha$  and  $\beta$  are  $(f_*, g_*)$ -twisted conjugate. Thus,

$$\text{Coin}(f, g) = \bigsqcup_{[\alpha]} p(\text{Coin}(\tilde{f}, \alpha \tilde{g})),$$

where  $[\alpha]$  runs over the  $(f_*, g_*)$ -twisted conjugacy classes. For sufficiently well-behaved spaces  $X$  and  $Y$  (e.g. nilmanifolds of equal dimension) we have that if  $R(f_*, g_*) < \infty$ , then

$$R(f_*, g_*) \leq |\text{Coin}(f, g)|,$$

whereas if  $R(f_*, g_*) = \infty$  there exist continuous maps  $f'$  and  $g'$  homotopic to  $f$  and  $g$  respectively such that  $\text{Coin}(f', g') = \emptyset$ .

## Chapter 3

# Twisted conjugation

### 3.1 The twisted conjugation action

Let  $G$  and  $H$  be groups and let  $\varphi$  and  $\psi$  be group homomorphisms from  $H$  to  $G$ . The pair  $(\varphi, \psi)$  induces a (right) group action of  $H$  on  $G$  given by the map

$$G \times H \rightarrow G: (g, h) \mapsto \varphi(h)^{-1} g \psi(h).$$

This group action is called  $(\varphi, \psi)$ -*twisted conjugation*.

If  $G = H$ ,  $\varphi$  is an endomorphism of  $G$  and  $\psi = \text{id}_G$ , then the action is usually called  $\varphi$ -*twisted conjugation*. In general, for the `TwistedConjugacy` package, many functions will take two homomorphisms `hom1` and `hom2` as arguments. However, if `hom1` is an endomorphism, `hom2` can be omitted, in which case it is automatically taken to be the identity map. Similarly, some functions will take two elements  $g_1$  and  $g_2$  as arguments. If  $g_2$  is omitted, it is automatically taken to be the identity element.

#### 3.1.1 TwistedConjugation

▷ `TwistedConjugation(hom1[, hom2])` (function)  
**Returns:** a function that maps the pair  $(g, h)$  to  $\text{hom1}(h)^{-1} g \text{hom2}(h)$ .

Example

```
gap> G := AlternatingGroup( 6 );;
gap> H := SymmetricGroup( 5 );;
gap> phi := GroupHomomorphismByImages( H, G, [ (1,2)(3,5,4), (2,3)(4,5) ],
> [ (1,4)(3,6), () ] );;
gap> psi := GroupHomomorphismByImages( H, G, [ (1,2)(3,5,4), (2,3)(4,5) ],
> [ (1,2)(3,4), () ] );;
gap> tc := TwistedConjugation( phi, psi );;
gap> g1 := (4,6,5);;
gap> h1 := (1,3,2)(4,5);;
gap> tc( g1, h1 );
(1,6,4,2)(3,5)
```

### 3.2 The twisted conjugacy (search) problem

Given groups  $G$  and  $H$ , group homomorphisms  $\varphi$  and  $\psi$  from  $H$  to  $G$  and elements  $g_1, g_2 \in G$ , the *twisted conjugacy problem* is the decision problem that asks whether  $g_1$  and  $g_2$  are  $(\varphi, \psi)$ -twisted



conjugate. The *twisted conjugacy search problem* is the problem of determining an explicit  $h$  such that  $\varphi(h)^{-1}g_1\psi(h) = g_2$  (under the assumption that such  $h$  exists).

### 3.2.1 IsTwistedConjugate

▷ `IsTwistedConjugate(hom1[, hom2], g1[, g2])` (function)  
**Returns:** true if  $g_1$  and  $g_2$  are  $(hom1, hom2)$ -twisted conjugate, otherwise false.

### 3.2.2 RepresentativeTwistedConjugation

▷ `RepresentativeTwistedConjugation(hom1[, hom2], g1[, g2])` (function)  
**Returns:** an element that maps  $g_1$  to  $g_2$  under the  $(hom1, hom2)$ -twisted conjugacy action, or fail if no such element exists.

If the source group is finite, this function relies on orbit-stabiliser algorithms provided by GAP. Otherwise, it relies on a mixture of the algorithms described in [Rom16, Thm. 3], [BKL<sup>+</sup>20, Sec. 5.4], [Rom21, Sec. 7] and [DT21].

Example

```
gap> g2 := (1,6,4,2)(3,5);;
gap> IsTwistedConjugate( psi, phi, g1, g2 );
false
gap> h2 := RepresentativeTwistedConjugation( phi, psi, g1, g2 );
(1,2)
gap> tc( g1, h2 ) = g2;
true
```

## 3.3 The multiple twisted conjugacy (search) problem

Let  $H$  and  $G_1, \dots, G_n$  be groups. For each  $i \in \{1, \dots, n\}$ , let  $g_i, g'_i \in G_i$  and let  $\varphi_i, \psi_i: H \rightarrow G_i$  be group homomorphisms. The *multiple twisted conjugacy problem* is the decision problem that asks whether there exists some  $h \in H$  such that  $\varphi_i(h)^{-1}g_i\psi_i(h) = g'_i$  for all  $i \in \{1, \dots, n\}$ . The *multiple twisted conjugacy search problem* is the problem of determining an explicit  $h$  such that  $\varphi_i(h)^{-1}g_i\psi_i(h) = g'_i$  for all  $i \in \{1, \dots, n\}$  (under the assumption that such  $h$  exists).

`IsTwistedConjugate` (3.2.1) and `RepresentativeTwistedConjugation` (3.2.2) can take lists instead of their usual arguments to solve these problems.

Example

```
gap> tau := GroupHomomorphismByImages( H, G, [ (1,2)(3,5,4), (2,3)(4,5) ],
> [ (1,2)(3,6), () ] );;
gap> khi := GroupHomomorphismByImages( H, G, [ (1,2)(3,5,4), (2,3)(4,5) ],
> [ (1,3)(4,6), () ] );;
gap> IsTwistedConjugate( [ phi, psi ], [ khi, tau ],
> [ (1,5)(4,6), (1,4)(3,5) ], [ (1,4,5,3,6), (2,4,5,6,3) ] );
true
gap> RepresentativeTwistedConjugation( [ phi, psi ], [ khi, tau ],
> [ (1,5)(4,6), (1,4)(3,5) ], [ (1,4,5,3,6), (2,4,5,6,3) ] );
(1,2)
```

## Chapter 4

# Twisted conjugacy classes

The orbits of the  $(\varphi, \psi)$ -twisted conjugacy action are called the  $(\varphi, \psi)$ -*twisted conjugacy classes* or the *Reidemeister classes* of  $(\varphi, \psi)$ . We denote the twisted conjugacy class of  $g \in G$  by  $[g]_{\varphi, \psi}$ .

### 4.1 Creating a twisted conjugacy class

#### 4.1.1 TwistedConjugacyClass

▷ `TwistedConjugacyClass(hom1[, hom2], g)` (function)

▷ `ReidemeisterClass(hom1[, hom2], g)` (function)

**Returns:** the  $(hom1, hom2)$ -twisted conjugacy class of  $g$ .

Example

```
gap> tcc := TwistedConjugacyClass( phi, psi, g1 );
(4,6,5)^G
```

### 4.2 Operations on twisted conjugacy classes

#### 4.2.1 Representative

▷ `Representative(tcc)` (attribute)

**Returns:** the group element that was used to construct  $tcc$ .

#### 4.2.2 ActingDomain

▷ `ActingDomain(tcc)` (attribute)

**Returns:** the group whose twisted conjugacy action  $tcc$  is an orbit of.

#### 4.2.3 FunctionAction

▷ `FunctionAction(tcc)` (attribute)

**Returns:** the twisted conjugacy action that  $tcc$  is an orbit of.

#### 4.2.4 `\in`

- ▷ `\in(g, tcc)` (operation)  
**Returns:** true if  $g$  is an element of  $tcc$ , otherwise false.

#### 4.2.5 `IsFinite`

- ▷ `IsFinite(tcc)` (property)  
**Returns:** true if  $tcc$  is finite, otherwise false.

#### 4.2.6 `Size`

- ▷ `Size(tcc)` (attribute)  
**Returns:** the number of elements in  $tcc$ .  
 This is calculated using the orbit-stabiliser theorem.

#### 4.2.7 `StabiliserOfExternalSet`

- ▷ `StabiliserOfExternalSet(tcc)` (attribute)  
 ▷ `StabilizerOfExternalSet(tcc)` (attribute)  
**Returns:** the stabiliser of `Representative(tcc)` under the action `FunctionAction(tcc)`.

#### 4.2.8 `List`

- ▷ `List(tcc)` (function)  
 ▷ `AsList(tcc)` (attribute)  
**Returns:** a list containing the elements of  $tcc$ .  
 If  $tcc$  is infinite, this will run forever or cause an error. It is recommended to first test the finiteness of  $tcc$  using `IsFinite` (4.2.5).

#### 4.2.9 `Random`

- ▷ `Random(tcc)` (operation)  
**Returns:** a random element in  $tcc$ .

#### 4.2.10 `\=`

- ▷ `\=(tcc1, tcc2)` (operation)  
**Returns:** true if  $tcc1$  is equal to  $tcc2$ , otherwise false.

Example

```
gap> Representative( tcc );
(4,6,5)
gap> ActingDomain( tcc ) = H;
true
gap> FunctionAction( tcc )( g1, h2 );
(1,6,4,2)(3,5)
gap> List( tcc );
[ (4,6,5), (1,6,4,2)(3,5) ]
gap> Size( tcc );
2
```

```
gap> StabiliserOfExternalSet( tcc );
Group([ (1,2,3,4,5), (1,3,4,5,2) ])
```

## 4.3 Calculating all twisted conjugacy classes

### 4.3.1 TwistedConjugacyClasses

▷ `TwistedConjugacyClasses(hom1[, hom2][, N])` (function)  
 ▷ `ReidemeisterClasses(hom1[, hom2][, N])` (function)

**Returns:** a list containing the  $(hom1, hom2)$ -twisted conjugacy classes if there are finitely many, or fail otherwise.

If the argument  $N$  is provided, it must be a normal subgroup of  $\text{Range}(hom1)$ ; the function will then only return the Reidemeister classes that intersect  $N$  non-trivially. It is guaranteed that the Reidemeister class of the identity is in the first position, and that the representatives of the classes belong to  $N$  if this argument is provided.

If  $G$  and  $H$  are finite, this function relies on an orbit-stabiliser algorithm. Otherwise, it relies on the algorithms in [DT21] and [Ter26].

### 4.3.2 RepresentativesTwistedConjugacyClasses

▷ `RepresentativesTwistedConjugacyClasses(hom1[, hom2][, N])` (function)  
 ▷ `RepresentativesReidemeisterClasses(hom1[, hom2][, N])` (function)

**Returns:** a list containing representatives of the  $(hom1, hom2)$ -twisted conjugacy classes if there are finitely many, or fail otherwise.

If the argument  $N$  is provided, it must be a normal subgroup of  $\text{Range}(hom1)$ ; the function will then only return the representatives of the twisted conjugacy classes that intersect  $N$  non-trivially. It is guaranteed that the identity is in the first position, and that all elements belong to  $N$  if this argument is provided.

Example

```
gap> TwistedConjugacyClasses( phi, psi ){ [ 1 .. 7 ] };
[ ()^G, (4,5,6)^G, (4,6,5)^G, (3,4)(5,6)^G, (3,4,5)^G, (3,4,6)^G, (3,5,4)^G ]
gap> RepresentativesTwistedConjugacyClasses( phi, psi ){ [ 1 .. 7 ] };
[ (), (4,5,6), (4,6,5), (3,4)(5,6), (3,4,5), (3,4,6), (3,5,4) ]
```

## Chapter 5

# Reidemeister numbers and spectra

### 5.1 Reidemeister numbers

The number of twisted conjugacy classes is called the Reidemeister number and is always a positive integer or infinity.

#### 5.1.1 ReidemeisterNumber

▷ `ReidemeisterNumber(hom1[, hom2])` (function)

▷ `NrTwistedConjugacyClasses(hom1[, hom2])` (function)

**Returns:** the Reidemeister number of  $(hom1, hom2)$ .

If  $G$  is abelian, this function relies on (a generalisation of) [Jia83, Thm. 2.5]. If  $G = H$ ,  $G$  is finite non-abelian and  $\psi = \text{id}_G$ , it relies on [Ree59, Thm. 1]. Otherwise, it simply calculates the twisted conjugacy classes and then counts them.

Example

```
gap> Q := QuaternionGroup( 8 );
gap> phi := GroupHomomorphismByImages( Q, Q, [ Q.1, Q.2 ], [ Q.2, Q.1 ] );
gap> ReidemeisterNumber( phi );
3
gap> D := DihedralGroup( 8 );
gap> psi := GroupHomomorphismByImages( Q, D, [ Q.1, Q.2 ], [ D.1 * D.2, D.3 ] );
gap> chi := GroupHomomorphismByImages( Q, D, [ Q.1, Q.2 ], [ D.1, D.3 ] );
gap> ReidemeisterNumber( psi, chi );
4
```

### 5.2 Reidemeister spectra

The set of all Reidemeister numbers of automorphisms is called the *Reidemeister spectrum* and is denoted by  $\text{Spec}_R(G)$ , i.e.

$$\text{Spec}_R(G) := \{ R(\varphi) \mid \varphi \in \text{Aut}(G) \}.$$

The set of all Reidemeister numbers of endomorphisms is called the *extended Reidemeister spectrum* and is denoted by  $\text{ESpec}_R(G)$ , i.e.

$$\text{ESpec}_R(G) := \{ R(\varphi) \mid \varphi \in \text{End}(G) \}.$$

The set of all Reidemeister numbers of pairs of homomorphisms from a group  $H$  to a group  $G$  is called the *coincidence Reidemeister spectrum* of  $H$  and  $G$  and is denoted by  $\text{CSpec}_R(H, G)$ , i.e.

$$\text{CSpec}_R(H, G) := \{R(\varphi, \psi) \mid \varphi, \psi \in \text{Hom}(H, G)\}.$$

If  $H = G$  this is also denoted by  $\text{CSpec}_R(G)$ . The set of all Reidemeister numbers of pairs of homomorphisms from every group  $H$  to a group  $G$  is called the *total Reidemeister spectrum* and is denoted by  $\text{TSpec}_R(G)$ , i.e.

$$\text{TSpec}_R(G) := \bigcup_H \text{CSpec}_R(H, G).$$

Please note that the functions below are only implemented for finite groups.

### 5.2.1 ReidemeisterSpectrum

▷ `ReidemeisterSpectrum( $G$ )` (function)

**Returns:** the Reidemeister spectrum of  $G$ .

If  $G$  is abelian, this function relies on the results from [Sen23]. Otherwise, it relies on [Ree59, Thm. 1].

### 5.2.2 ExtendedReidemeisterSpectrum

▷ `ExtendedReidemeisterSpectrum( $G$ )` (function)

**Returns:** the extended Reidemeister spectrum of  $G$ .

If  $G$  is simple, this is the union of its Reidemeister spectrum with the element 1. If  $G$  is abelian, this is just the set of all divisors its order. Otherwise, this function relies on [Ree59, Thm. 1].

### 5.2.3 CoincidenceReidemeisterSpectrum

▷ `CoincidenceReidemeisterSpectrum( $[H, ]G$ )` (function)

**Returns:** the coincidence Reidemeister spectrum of  $H$  and  $G$ .

### 5.2.4 TotalReidemeisterSpectrum

▷ `TotalReidemeisterSpectrum( $G$ )` (function)

**Returns:** the total Reidemeister spectrum of  $H$  and  $G$ .

Example

```
gap> ReidemeisterSpectrum( Q );
[ 2, 3, 5 ]
gap> ExtendedReidemeisterSpectrum( Q );
[ 1, 2, 3, 5 ]
gap> CoincidenceReidemeisterSpectrum( Q );
[ 1, 2, 3, 4, 5, 8 ]
gap> CoincidenceReidemeisterSpectrum( D, Q );
[ 4, 8 ]
gap> CoincidenceReidemeisterSpectrum( Q, D );
[ 2, 3, 4, 6, 8 ]
gap> TotalReidemeisterSpectrum( Q );
[ 1, 2, 3, 4, 5, 6, 8 ]
```

## Chapter 6

# Reidemeister zeta and generating functions

Please note that the functions in this chapter are only implemented for endomorphisms of finite groups.

### 6.1 Decomposing Reidemeister number sequences

For a finite group, the sequence of Reidemeister numbers of the iterates of  $\text{endo1}$  and  $\text{endo2}$ , i.e. the sequence  $R(\text{endo1}, \text{endo2})$ ,  $R(\text{endo1}^2, \text{endo2}^2)$ , ..., is eventually periodic. Thus there exist a periodic sequence  $(P_n)_{n \in \mathbb{N}}$  and an eventually zero sequence  $(Q_n)_{n \in \mathbb{N}}$  such that

$$\forall n \in \mathbb{N} : R(\varphi^n, \psi^n) = P_n + Q_n.$$

#### 6.1.1 IteratedReidemeisterNumberDecomposition

▷ `IteratedReidemeisterNumberDecomposition(endo1[, endo2])` (function)

**Returns:** two lists of integers: the first list contains one period of the sequence  $(P_n)_{n \in \mathbb{N}}$ , the second list contains  $(Q_n)_{n \in \mathbb{N}}$  up to the part where it becomes the constant zero sequence.

Example

```
gap> G := PcGroupCode( 3913, 12 );;
gap> phi := GroupHomomorphismByImages( G, G, [ G.1, G.3 ],
> [ One( G ), One( G ) ] );;
gap> psi := GroupHomomorphismByImages( G, G, [ G.1, G.3 ],
> [ G.2, One( G ) ] );;
gap> IteratedReidemeisterNumberDecomposition( phi );
[ [ 1 ], [ ] ]
gap> IteratedReidemeisterNumberDecomposition( phi, psi );
[ [ 12 ], [ -6 ] ]
```

### 6.2 Reidemeister zeta functions

Let  $\varphi, \psi : G \rightarrow G$  be endomorphisms such that  $R(\varphi^n, \psi^n) < \infty$  for all  $n \in \mathbb{N}$ . Then the *Reidemeister zeta function*  $Z_{\varphi, \psi}(s)$  of the pair  $(\varphi, \psi)$  is defined as

$$Z_{\varphi, \psi}(s) := \exp \sum_{n=1}^{\infty} \frac{R(\varphi^n, \psi^n)}{n} s^n.$$

### 6.2.1 IsRationalReidemeisterZetaFunction

▷ `IsRationalReidemeisterZetaFunction(endo1[, endo2])` (function)

**Returns:** true if the Reidemeister zeta function of *endo1* and *endo2* is rational, otherwise false.

### 6.2.2 ReidemeisterZetaFunction

▷ `ReidemeisterZetaFunction(endo1[, endo2])` (function)

**Returns:** the Reidemeister zeta function of *endo1* and *endo2* if it is rational, otherwise fail.

### 6.2.3 PrintReidemeisterZetaFunction

▷ `PrintReidemeisterZetaFunction(endo1[, endo2])` (function)

**Returns:** a string describing the Reidemeister zeta function of *endo1* and *endo2*.

This is often more readable than evaluating `ReidemeisterZetaFunction` (6.2.2) in an indeterminate, and does not require rationality.

Example

```
gap> IsRationalReidemeisterZetaFunction( phi );
true
gap> IsRationalReidemeisterZetaFunction( phi, psi );
false
gap> s := Indeterminate( Rationals, "s" );;
gap> ReidemeisterZetaFunction( phi )( s );
(1)/(-s+1)
gap> PrintReidemeisterZetaFunction( phi, psi );
"exp(-6*s)*(1-s)^(-12)"
```

## 6.3 Reidemeister generating functions

Let  $\varphi, \psi: G \rightarrow G$  be endomorphisms such that  $R(\varphi^n, \psi^n) < \infty$  for all  $n \in \mathbb{N}$ . Then the *Reidemeister generating function*  $Z_{\varphi, \psi}^*(s)$  of the pair  $(\varphi, \psi)$  is defined as

$$Z_{\varphi, \psi}^*(s) := \sum_{n=1}^{\infty} R(\varphi^n, \psi^n) s^n.$$

### 6.3.1 IsRationalReidemeisterGeneratingFunction

▷ `IsRationalReidemeisterGeneratingFunction(endo1[, endo2])` (function)

**Returns:** true if the Reidemeister generating function of *endo1* and *endo2* is rational, otherwise false.

### 6.3.2 ReidemeisterGeneratingFunction

▷ `ReidemeisterGeneratingFunction(endo1[, endo2])` (function)

**Returns:** the Reidemeister generating function of *endo1* and *endo2* if it is rational, otherwise fail.



### 6.3.3 PrintReidemeisterGeneratingFunction

▷ `PrintReidemeisterGeneratingFunction(endo1[, endo2])` (function)

**Returns:** a string describing the Reidemeister generating function of *endo1* and *endo2*.

This is often more readable than evaluating `ReidemeisterGeneratingFunction` (6.3.2) in an indeterminate, and does not require rationality.

Example

```
gap> IsRationalReidemeisterGeneratingFunction( phi, psi );
true
gap> ReidemeisterGeneratingFunction( phi, psi )( 2 );
-36
gap> PrintReidemeisterGeneratingFunction( phi, psi );
"(6*s^2+6*s)/(-s+1)"
```

## Chapter 7

# Cosets of PcpGroups

GAP is well-equipped to deal with *finite* cosets. However, if a coset is infinite, methods may not be available, may be faulty, or may run forever. The `TwistedConjugacy` package provides additional methods for existing functions that can deal with infinite cosets of `PcpGroups`.

The only completely new functions are `DoubleCosetIndex` (7.2.8) and its NC version.

### 7.1 Right cosets

Calculating the intersection of two right cosets  $Hx$  and  $Ky$  can be reduced to calculating the intersection  $H \cap K$  and verifying whether  $xy^{-1} \in HK$  (see \in (7.2.1)).

#### 7.1.1 Intersection

- ▷ `Intersection(C1, C2, ...)` (function)
- ▷ `Intersection(L)` (function)

**Returns:** the intersection of the right cosets  $C1, C2, \dots$

Alternatively, this function also accepts a single list of right cosets  $L$  as argument.

This intersection is always a right coset, or the empty list.

Example

```
gap> G := ExamplesOfSomePcpGroups( 5 );
gap> H := Subgroup( G, [ G.1 * G.2 ^ -1, G.3 ^ 2 ] );
gap> K := Subgroup( G, [ G.1 ^ 2, G.2 * G.3 ] );
gap> x := G.1 * G.3 ^ -1;
gap> y := G.1 * G.2 ^ 2 * G.3;
gap> z := G.3 * G.4 ^ -1;
gap> Hx := RightCoset( H, x );
gap> Ky := RightCoset( K, y );
gap> Intersection( Hx, Ky );
RightCoset(<group with 1 generator>,<object>)
gap> Kz := RightCoset( K, z );
gap> Intersection( Hx, Kz );
[ ]
```

## 7.2 Double cosets

Algorithms designed for computing with twisted conjugacy classes can be leveraged to do computations involving double cosets, see [Ter26, Sec. 9] for a description on this. When the `TwistedConjugacy` package is loaded, it does this automatically, and the functions below should then work for `PcpGroups`, even if they are infinite.

### 7.2.1 `\in`

▷ `\in(g, D)` (operation)  
**Returns:** true if  $g$  is an element of  $D$ , otherwise false.

### 7.2.2 `IsFinite`

▷ `IsFinite(D)` (property)  
**Returns:** true if  $D$  is finite, otherwise false.

### 7.2.3 `Size`

▷ `Size(D)` (attribute)  
**Returns:** the number of elements in  $D$ .

### 7.2.4 `List`

▷ `List(D)` (function)  
 ▷ `AsList(D)` (attribute)  
**Returns:** a list containing the elements of  $D$ .  
 If  $D$  is infinite, this will run forever or cause an error. It is recommended to first test the finiteness of  $D$  using `IsFinite` (7.2.2).

### 7.2.5 `\=`

▷ `\=(C, D)` (operation)  
**Returns:** true if  $C$  and  $D$  are the same double coset, otherwise false.

### 7.2.6 `DoubleCosets`

▷ `DoubleCosets(G, H, K)` (function)  
 ▷ `DoubleCosetsNC(G, H, K)` (operation)  
**Returns:** a duplicate-free list of all  $(H, K)$ -double cosets in  $G$  if there are finitely many, otherwise fail.

The groups  $H$  and  $K$  must be subgroups of the group  $G$ . The NC version does not check whether this is the case.

### 7.2.7 DoubleCosetRepsAndSizes

▷ `DoubleCosetRepsAndSizes( $G, H, K$ )` (operation)

**Returns:** a list containing pairs of the form  $[r, n]$ , where  $r$  is a representative and  $n$  is the size of a double coset.

While for finite groups this function is supposed to be faster than `DoubleCosetsNC` (7.2.6), for `PcpGroups` it is usually *slower*.

### 7.2.8 DoubleCosetIndex

▷ `DoubleCosetIndex( $G, H, K$ )` (function)

▷ `DoubleCosetIndexNC( $G, H, K$ )` (operation)

**Returns:** the double coset index of the pair  $(H, K)$ .

The groups  $H$  and  $K$  must be subgroups of the group  $G$ . The NC version does not check whether this is the case.

Example

```
gap> HxK := DoubleCoset( H, x, K );;
gap> HzK := DoubleCoset( H, z, K );;
gap> y in HxK;
true
gap> HxK = HzK;
false
gap> DoubleCosets( G, H, K );
[ DoubleCoset(<group with 2 generators>,<object>,
               <group of size infinity with 2 generators>),
  DoubleCoset(<group with 2 generators>,<object>,
               <group of size infinity with 2 generators>) ]
gap> DoubleCosetIndex( G, H, K );
2
```

## Chapter 8

# Group homomorphisms

### 8.1 Representatives of homomorphisms between groups

Please note that the functions below are only implemented for finite groups.

#### 8.1.1 RepresentativesAutomorphismClasses

- ▷ `RepresentativesAutomorphismClasses( $G$ )` (function)  
**Returns:** a list of the automorphisms of  $G$  up to composition with inner automorphisms.

#### 8.1.2 RepresentativesEndomorphismClasses

- ▷ `RepresentativesEndomorphismClasses( $G$ )` (function)  
**Returns:** a list of the endomorphisms of  $G$  up to composition with inner automorphisms.  
This does the same as calling `AllHomomorphismClasses( $G, G$ )`, but should be faster for abelian and non-2-generated groups.

#### 8.1.3 RepresentativesHomomorphismClasses

- ▷ `RepresentativesHomomorphismClasses( $H, G$ )` (function)  
**Returns:** a list of the homomorphisms from  $H$  to  $G$ , up to composition with inner automorphisms of  $G$ .

Similar to the previous function, this function does the same as calling `AllHomomorphismClasses( $H, G$ )`, but should be faster for abelian and non-2-generated groups.

Example

```
gap> G := PcGroupCode( 1018013, 28 );;  
gap> AutS := RepresentativesAutomorphismClasses( G );;  
gap> Size( AutS );  
6  
gap> EndS := RepresentativesEndomorphismClasses( G );;  
gap> Size( EndS );  
10  
gap> H := PcGroupCode( 36293, 28 );;  
gap> HomS := RepresentativesHomomorphismClasses( H, G );;  
gap> Size( HomS );  
4
```

## 8.2 Coincidence and fixed point groups

### 8.2.1 FixedPointGroup

▷ `FixedPointGroup(endo)` (function)

**Returns:** the subgroup of `Source(endo)` consisting of the elements fixed under the endomorphism `endo`.

### 8.2.2 CoincidenceGroup

▷ `CoincidenceGroup(hom1, hom2[, ...])` (function)

**Returns:** the subgroup of `Source(hom1)` consisting of the elements  $h$  for which  $h^{\text{hom1}} = h^{\text{hom2}} = \dots$

For infinite non-abelian groups, this function relies on a mixture of the algorithms described in [Rom16, Thm. 2], [BKL<sup>+</sup>20, Sec. 5.4] and [Rom21, Sec. 7].

Example

```
gap> phi := GroupHomomorphismByImages( G, G, [ G.1, G.3 ],
> [ G.1 * G.2, G.3 ^ 3 ] );;
gap> Set( FixedPointGroup( phi ) );
[ <identity> of ..., f2 ]
gap> psi := GroupHomomorphismByImages( H, G, [ H.1, H.2, H.3 ],
> [ One( G ), G.2, One( G ) ] );;
gap> khi := GroupHomomorphismByImages( H, G, [ H.1, H.2, H.3 ],
> [ G.2, G.2, One( G ) ] );;
gap> CoincidenceGroup( psi, khi );
Group([ f2, f3 ])
```

## 8.3 Induced and restricted group homomorphisms

### 8.3.1 InducedHomomorphism

▷ `InducedHomomorphism(epi1, epi2, hom)` (function)

**Returns:** the homomorphism induced by `hom` between the images of `epi1` and `epi2`.

Let `hom` be a group homomorphism from a group  $H$  to a group  $G$ , let `epi1` be an epimorphism from  $H$  to a group  $Q$  and let `epi2` be an epimorphism from  $G$  to a group  $P$  such that the kernel of `epi1` is mapped into the kernel of `epi2` by `hom`. This command returns the homomorphism from  $Q$  to  $P$  that maps  $h^{\text{epi1}}$  to  $(h^{\text{hom}})^{\text{epi2}}$ , for any element  $h$  of  $H$ . This function generalises `InducedAutomorphism` (**Reference:** `InducedAutomorphism`) to homomorphisms.

### 8.3.2 RestrictedHomomorphism

▷ `RestrictedHomomorphism(hom, N, M)` (function)

**Returns:** the homomorphism `hom`, but restricted as a map from  $N$  to  $M$ .

Let `hom` be a group homomorphism from a group  $H$  to a group  $G$ , and let  $N$  be subgroup of  $H$  such that its image under `hom` is a subgroup of  $M$ . This command returns the homomorphism from  $N$  to  $M$  that maps  $n$  to  $n^{\text{hom}}$  for any element  $n$  of  $N$ . No checks are made to verify that `hom` maps  $N$  into  $M$ . This function is similar to `RestrictedMapping` (**Reference:** `RestrictedMapping`), but its range is explicitly set to  $M$ .

## Example

```
gap> N := DerivedSubgroup( G );;  
gap> p := NaturalHomomorphismByNormalSubgroup( G, N );  
[ f1, f2, f3 ] -> [ f1, f2, <identity> of ... ]  
gap> ind := InducedHomomorphism( p, p, phi );  
[ f1 ] -> [ f1*f2 ]  
gap> res := RestrictedHomomorphism( phi, N, N );  
[ f3 ] -> [ f3^3 ]
```

## Chapter 9

# Group derivations

Let  $G$  and  $H$  be groups and let  $H$  act on  $G$  via automorphisms, i.e. there is a group homomorphism

$$\alpha: H \rightarrow \text{Aut}(G): h \mapsto \alpha_h$$

such that  $g^h = \alpha_h(g)$  for all  $g \in G$  and  $h \in H$ . A *group derivation*  $\delta: H \rightarrow G$  is a map such that

$$\delta(h_1 h_2) = \delta(h_1)^{h_2} \delta(h_2).$$

Note that we do not require  $G$  to be abelian.

Algorithms designed for computing with twisted conjugacy classes can be leveraged to do computations involving group derivations, see [Ter26, Sec. 10] for a description on this.

Please note that the functions in this chapter require  $G$  and  $H$  to either both be finite, or both be PcpGroups.

## 9.1 Creating group derivations

### 9.1.1 GroupDerivationByImages

- ▷ `GroupDerivationByImages( $H, G[[, gens], imgs], act$ )` (function)
- ▷ `GroupDerivationByImagesNC( $H, G[[, gens], imgs], act$ )` (function)

**Returns:** the specified group derivation, or fail if the given arguments do not define a derivation.

This works in the same vein as `GroupHomomorphismByImages` (**Reference: GroupHomomorphismByImages**). The group  $H$  acts on the group  $G$  via  $act$ , which must be a homomorphism from  $H$  into a group of automorphisms of  $G$ . This command then returns the group derivation defined by mapping the list  $gens$  of generators of  $H$  to the list  $imgs$  of images in  $G$ .

If omitted, the arguments  $gens$  and  $imgs$  default to the `GeneratorsOfGroup` value of  $H$  and  $G$  respectively.

This function checks whether  $gens$  generate  $H$  and whether the mapping of the generators extends to a group derivation. This test can be expensive, so if one is certain that the given arguments produce a group derivation, these checks can be avoided by using the NC version.

### 9.1.2 GroupDerivationByFunction

- ▷ `GroupDerivationByFunction( $H, G, fun, act$ )` (function)

**Returns:** the specified group derivation.



`GroupDerivationByFunction` works in the same vein as `GroupHomomorphismByFunction` (**Reference: `GroupHomomorphismByFunction`**). The group  $H$  acts on the group  $G$  via  $act$ , which must be a homomorphism from  $H$  into a group of automorphisms of  $G$ . This command then returns the group derivation defined by mapping the element  $h$  of  $H$  to the element  $fun(h)$  of  $G$ , where  $fun$  is a GAP function.

No tests are performed to check whether the arguments really produce a group derivation.

### 9.1.3 GroupDerivationByAffineAction

▷ `GroupDerivationByAffineAction( $H$ ,  $G$ ,  $act$ )` (function)

**Returns:** the derivation that makes up the translational part of the affine action.

Example

```
gap> H := PcGroupCode( 149167619499417164, 72 );;
gap> G := PcGroupCode( 5551210572, 72 );;
gap> inn := InnerAutomorphism( G, G.2 );;
gap> hom := GroupHomomorphismByImages( G, G, [ G.1 * G.2, G.5 ],
> [ G.1 * G.2 ^ 2 * G.3 ^ 2 * G.4, G.5 ] );;
gap> act := GroupHomomorphismByImages( H, AutomorphismGroup( G ),
> [ H.2, H.1 * H.4 ], [ inn, hom ] );;
gap> gens := [ H.2, H.1 * H.4 ];;
gap> imgs := [ G.5, G.2 ];;
gap> der := GroupDerivationByImages( H, G, gens, imgs, act );
Group derivation [ f2, f1*f4 ] -> [ f5, f2 ]
```

## 9.2 Operations for group derivations

Many of the functions, operations, attributes... available to group homomorphisms are available for group derivations as well. We list some of the more useful ones.

### 9.2.1 IsInjective

▷ `IsInjective( $der$ )` (property)

**Returns:** true if the group derivation  $der$  is injective, otherwise false.

### 9.2.2 IsSurjective

▷ `IsSurjective( $der$ )` (property)

**Returns:** true if the group derivation  $der$  is surjective, otherwise false.

### 9.2.3 IsBijective

▷ `IsBijective( $der$ )` (property)

**Returns:** true if the group derivation  $der$  is bijective, otherwise false.

### 9.2.4 Kernel

▷ `Kernel( $der$ )` (operation)

**Returns:** the set of elements that are mapped to the identity by  $der$ .

This will always be a subgroup of `Source( $der$ )`.

### 9.2.5 Image

- ▷ `Image(der)` (function)
- ▷ `Image(der, elm)` (function)
- ▷ `Image(der, sub)` (function)

**Returns:** the image of the group derivation *der*.

One can optionally give an element *elm* or a subgroup *sub* as a second argument, in which case `Image` will calculate the image of this argument under *der*.

### 9.2.6 PreImagesRepresentative

- ▷ `PreImagesRepresentative(der, elm)` (operation)

**Returns:** a preimage of the element *elm* under the group derivation *der*, or `fail` if no preimage exists.

### 9.2.7 PreImages

- ▷ `PreImages(der, elm)` (function)

**Returns:** the set of all preimages of the element *elm* under the group derivation *der*.

This will always be a (right) coset of `Kernel(der)`, or the empty list.

Example

```
gap> IsInjective( der ) or IsSurjective( der );
false
gap> K := Kernel( der );
gap> Size( K );
9
gap> ImH := Image( der );
Group derivation image in Group( [ f1, f2, f3, f4, f5 ] )
gap> h1 := H.1 * H.3;;
gap> g := Image( der, h1 );
f2*f4
gap> ImK := Image( der, K );
Group derivation image in Group( [ f1, f2, f3, f4, f5 ] )
gap> h2 := PreImagesRepresentative( der, g );
gap> Image( der, h2 ) = g;
true
gap> PreIm := PreImages( der, g );
RightCoset(<group of size 9 with 2 generators>,<object>)
gap> PreIm = RightCoset( K, h2 );
true
```

## 9.3 Images of group derivations

In general, the image of a group derivation is not a subgroup. However, it is still possible to do a membership test, to calculate the number of elements, and to enumerate the elements if there are only finitely many.

### 9.3.1 `\in`

- ▷ `\in(elm, img)` (operation)  
**Returns:** true if *elm* is an element of *img*, otherwise false.

### 9.3.2 `IsFinite`

- ▷ `IsFinite(img)` (property)  
**Returns:** true if *img* is finite, otherwise false.

### 9.3.3 `Size`

- ▷ `Size(img)` (attribute)  
**Returns:** the number of elements in *img*.

### 9.3.4 `List`

- ▷ `List(img)` (function)  
 ▷ `AsList(img)` (attribute)  
**Returns:** a list containing the elements of *img*.  
 If *img* is infinite, this will run forever or cause an error. It is recommended to first test the finiteness of *img* using `IsFinite` (9.3.2).

Example

```
gap> Size( ImH );
8
gap> Size( ImK );
1
gap> g in ImH;
true
gap> g in ImK;
false
gap> List( ImK );
[ <identity> of ... ]
```

# Chapter 10

## Affine actions

Let  $G$  and  $H$  be groups, let  $H$  act on  $G$  (via automorphisms) by

$$\alpha: H \rightarrow \text{Aut}(G): h \mapsto \alpha_h$$

and let  $\delta: H \rightarrow G$  be a group derivation with respect to this action. Then we can construct a new action, called the *affine action* associated to  $\delta$ , by

$$G \times H \rightarrow G: g^h = \alpha_h(g)\delta(h).$$

If  $K$  is a subgroup of  $H$ , then the restriction of the affine action of  $H$  on  $G$  to  $K$  coincides with the affine action of  $K$  on  $G$  associated to the restriction of  $\delta$  to  $K$ .

Algorithms designed for computing with twisted conjugacy classes can be leveraged to do computations involving affine actions, see [Ter26, Sec. 10] for a description on this.

Please note that the functions in this chapter require  $G$  and  $H$  to either both be finite, or both be PcpGroups.

### 10.1 Creating an affine action

#### 10.1.1 AffineActionByGroupDerivation

▷ `AffineActionByGroupDerivation( $K$ ,  $der$ )` (function)

**Returns:** the affine action of  $K$  associated to the derivation  $der$ .

The group  $K$  must be a subgroup of `Source( $der$ )`.

Example

```
gap> aff := AffineActionByGroupDerivation( H, der );
function( g, k ) ... end
```

### 10.2 Operations for affine actions

These functions are analogues of existing GAP functions for group actions.

#### 10.2.1 OrbitAffineAction

▷ `OrbitAffineAction( $K$ ,  $g$ ,  $der$ )` (function)

**Returns:** the orbit of  $g$  under the affine action of  $K$  associated to  $der$ .

The group  $K$  must be a subgroup of `Source( $der$ )`.

### 10.2.2 OrbitsAffineAction

▷ `OrbitsAffineAction(K, der)` (function)

**Returns:** a list containing the orbits under the affine action of *K* associated to *der* if there are finitely many, or fail if there are infinitely many.

The group *K* must be a subgroup of `Source(der)`.

### 10.2.3 NrOrbitsAffineAction

▷ `NrOrbitsAffineAction(K, der)` (function)

**Returns:** the number of orbits under the affine action of *K* associated to *der*.

### 10.2.4 StabiliserAffineAction

▷ `StabiliserAffineAction(K, g, der)` (function)

▷ `StabilizerAffineAction(K, g, der)` (function)

**Returns:** the stabiliser of *g* under the affine action of *K* associated to *der*.

The group *K* must be a subgroup of `Source(der)`.

### 10.2.5 RepresentativeAffineAction

▷ `RepresentativeAffineAction(K, g1, g2, der)` (function)

**Returns:** an element of *K* that maps *g1* to *g2* under the affine action of *K* associated to *der*, or fail if no such element exists.

The group *K* must be a subgroup of `Source(der)`.

Example

```
gap> g1 := G.1;;
gap> orb := OrbitAffineAction( H, g1, der );
f1^G
gap> NrOrbitsAffineAction( H, der );
10
gap> stab := StabiliserAffineAction( H, g1, der );;
gap> Set( stab );
[ <identity> of ..., f3, f3^2, f2^2*f5, f2*f4*f5,
  f2^2*f3*f5, f2*f3*f4*f5, f2^2*f3^2*f5, f2*f3^2*f4*f5 ]
gap> g2 := G.1 * G.4 * G.5;;
gap> h := RepresentativeAffineAction( H, g1, g2, der );;
gap> aff( g1, h ) = g2;
true
```

## 10.3 Operations on orbits of affine actions

### 10.3.1 Representative

▷ `Representative(orb)` (attribute)

**Returns:** the group element that was used to construct *orb*.

### 10.3.2 ActingDomain

- ▷ `ActingDomain(orb)` (attribute)  
**Returns:** the group whose affine action *orb* is an orbit of.

### 10.3.3 FunctionAction

- ▷ `FunctionAction(orb)` (attribute)  
**Returns:** the affine action that *orb* is an orbit of.

### 10.3.4 \in

- ▷ `\in(elm, orb)` (operation)  
**Returns:** true if *elm* is an element of *orb*, otherwise false.

### 10.3.5 IsFinite

- ▷ `IsFinite(orb)` (property)  
**Returns:** true if *orb* is finite, otherwise false.

### 10.3.6 Size

- ▷ `Size(orb)` (attribute)  
**Returns:** the number of elements in *orb*.

### 10.3.7 StabiliserOfExternalSet

- ▷ `StabiliserOfExternalSet(orb)` (attribute)  
 ▷ `StabilizerOfExternalSet(orb)` (attribute)  
**Returns:** the stabiliser of `Representative(orb)` under the action `FunctionAction(orb)`.

### 10.3.8 List

- ▷ `List(orb)` (function)  
 ▷ `AsList(orb)` (attribute)

**Returns:** a list containing the elements of *orb*.

If *orb* is infinite, this will run forever or cause an error. It is recommended to first test the finiteness of *orb* using `IsFinite` ([10.3.5](#)).

### 10.3.9 Random

- ▷ `Random(orb)` (operation)  
**Returns:** a random element in *orb*.

### 10.3.10 \=

- ▷ `\=(orb1, orb2)` (operation)  
**Returns:** true if *orb1* is equal to *orb2*, otherwise false.

## Example

```
gap> g2 in orb;  
true  
gap> G.2 in orb;  
false  
gap> Size( orb );  
8
```

# References

- [BKL<sup>+</sup>20] F. Bassino, I. Kapovich, M. Lohrey, A. Miasnikov, C. Nicaud, A. Nikolaev, I. Rivin, V. Shpilrain, A. Ushakov, and P. Weil. *Complexity and Randomness in Group Theory — GAGTA BOOK 1*. De Gruyter, 2020. URL: <https://doi.org/10/grwd34>. 9, 22
- [DT21] K. Dekimpe and S. Tertooy. Algorithms for twisted conjugacy classes of polycyclic-by-finite groups. *Topology Appl.*, 293:107565, 2021. URL: <https://doi.org/10/grvpwc>. 9, 12
- [Jia83] B. Jiang. *Lectures on Nielsen fixed point theory*, volume 14 of *Contemp. Math.* Amer. Math. Soc., 1983. URL: <https://doi.org/10/fxzf86>. 13
- [Ree59] R. Ree. On generalized conjugate classes in a finite group. *Ill. J. Math.*, 3(3):440–444, 1959. URL: <https://doi.org/10/hb2rkv>. 13, 14
- [Rom16] V. Roman’kov. On solvability of equations with endomorphisms in nilpotent groups. *Sib. Elektron. Mat. Izv.*, 13:716–725, 2016. URL: <https://doi.org/10/grv3bv>. 9, 22
- [Rom21] V. Roman’kov. Algorithmic theory of solvable groups. *Prikl. Diskretn. Mat.*, 52:16–64, 2021. URL: <https://doi.org/10/grwdfp>. 9, 22
- [Sen23] P. Senden. The Reidemeister Spectrum of Finite Abelian Groups. *Proc. Edinburgh Math. Soc.*, 66(4):940–959, 2023. URL: <https://doi.org/10/gszbsf>. 14
- [Ter26] Sam Tertooy. Algorithms for twisted conjugacy classes of polycyclic-by-finite groups II. *J. Algebra*, 2026. Advance Online Publication. URL: <https://doi.org/10/q5jb>. 12, 19, 24, 28



# Index

- $\backslash =$ 
  - for double cosets of a `PcpGroup`, 19
  - for orbits of an affine action, 30
  - for twisted conjugacy classes, 11
- $\backslash \text{in}$ 
  - for an element and a double coset of a `PcpGroup`, 19
  - for an element and a group derivation image, 27
  - for an element and a twisted conjugacy class, 11
  - for an element and an orbit of an affine action, 30
- `ActingDomain`
  - of a twisted conjugacy class, 10
  - of an orbit of an affine action, 30
- `AffineActionByGroupDerivation`, 28
- `AsList`
  - for a double coset of a `PcpGroup`, 19
  - for a group derivation image, 27
  - for a twisted conjugacy class, 11
  - for an orbit of an affine action, 30
- `CoincidenceGroup`, 22
- `CoincidenceReidemeisterSpectrum`, 14
- `DoubleCosetIndex`, 20
- `DoubleCosetIndexNC`, 20
- `DoubleCosetRepsAndSizes`
  - for `PcpGroups`, 20
- `DoubleCosets`
  - for `PcpGroups`, 19
- `DoubleCosetsNC`
  - for `PcpGroups`, 19
- `ExtendedReidemeisterSpectrum`, 14
- `FixedPointGroup`, 22
- `FunctionAction`
  - of a twisted conjugacy class, 10
  - of an orbit of an affine action, 30
- `GroupDerivationByAffineAction`, 25
- `GroupDerivationByFunction`, 24
- `GroupDerivationByImages`, 24
- `GroupDerivationByImagesNC`, 24
- `Image`
  - of a group derivation, 26
  - of a subgroup under a group derivation, 26
  - of an element under a group derivation, 26
- `InducedHomomorphism`, 22
- `Intersection`
  - of a list of right cosets of a `PcpGroup`, 18
  - of right cosets of a `PcpGroup`, 18
- `IsBijective`
  - for a group derivation, 25
- `IsFinite`
  - for a double coset of a `PcpGroup`, 19
  - for a group derivation image, 27
  - for a twisted conjugacy class, 11
  - for an orbit of an affine action, 30
- `IsInjective`
  - for a group derivation, 25
- `IsRationalReidemeisterGeneratingFunction`, 16
- `IsRationalReidemeisterZetaFunction`, 16
- `IsSurjective`
  - for a group derivation, 25
- `IsTwistedConjugate`, 9
- `IteratedReidemeisterNumberDecomposition`, 15
- `Kernel`
  - of a group derivation, 25
- `List`
  - of a double coset of a `PcpGroup`, 19
  - of a group derivation image, 27

- of a twisted conjugacy class, [11](#)
  - of an orbit of an affine action, [30](#)
- [NrOrbitsAffineAction, 29](#)
- [NrTwistedConjugacyClasses, 13](#)
- [OrbitAffineAction, 28](#)
- [OrbitsAffineAction, 29](#)
- [PreImages](#)
  - of an element under a group derivation, [26](#)
- [PreImagesRepresentative](#)
  - of an element under a group derivation, [26](#)
- [PrintReidemeisterGeneratingFunction, 17](#)
- [PrintReidemeisterZetaFunction, 16](#)
- [Random](#)
  - in a twisted conjugacy class, [11](#)
  - in an orbit of an affine action, [30](#)
- [ReidemeisterClass, 10](#)
- [ReidemeisterClasses, 12](#)
- [ReidemeisterGeneratingFunction, 16](#)
- [ReidemeisterNumber, 13](#)
- [ReidemeisterSpectrum, 14](#)
- [ReidemeisterZetaFunction, 16](#)
- [Representative](#)
  - of a twisted conjugacy class, [10](#)
  - of an orbit of an affine action, [29](#)
- [RepresentativeAffineAction, 29](#)
- [RepresentativesAutomorphismClasses, 21](#)
- [RepresentativesEndomorphismClasses, 21](#)
- [RepresentativesHomomorphismClasses, 21](#)
- [RepresentativesReidemeisterClasses, 12](#)
- [RepresentativesTwistedConjugacyClasses, 12](#)
- [RepresentativeTwistedConjugation, 9](#)
- [RestrictedHomomorphism, 22](#)
- [Size](#)
  - of a double coset of a PcpGroup, [19](#)
  - of a group derivation image, [27](#)
  - of a twisted conjugacy class, [11](#)
  - of an orbit of an affine action, [30](#)
- [StabiliserAffineAction, 29](#)
- [StabiliserOfExternalSet](#)
  - of a twisted conjugacy class, [11](#)
  - of an orbit of an affine action, [30](#)
- [TotalReidemeisterSpectrum, 14](#)
- [TwistedConjugacyClass, 10](#)
- [TwistedConjugacyClasses, 12](#)
- [TwistedConjugation, 8](#)