TwistedConjugacy

Computation with twisted conjugacy classes

2.4.0

5 December 2024

Sam Tertooy

Sam Tertooy
Email: 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 to calculate Reidemeister classes, numbers, spectra and zeta functions, as well as other methods related to homomorphisms, endomorphisms and automorphisms of groups. These methods are, for the most part, designed to be used with finite groups and polycyclically presented groups.

Copyright

Acknowledgements

This documentation was created using the GAPDOC and AUTODOC packages.

Contents

Chapter 1

Preface

Let G,H be groups and $\varphi,\psi\colon H\to G$ group homomorphisms. Then the pair (φ,ψ) induces a (right) group action on G given by

$$G\times H\to G\colon (g,h)\mapsto g\cdot h=\psi(h)^{-1}g\varphi(h).$$

This group action is called (φ,ψ)*-twisted conjugation*, and induces an equivalence relation $\sim_{\varphi,\psi}$ on G:

$$g_1\sim_{\varphi,\psi} g_2 \iff \exists h\in H : g_1\cdot h = g2.$$

The equivalence classes (i.e. the orbits of the action) are called *Reidemeister classes* and the number of Reidemeister classes is called the *Reidemeister number* $R(\varphi,\psi)$ of the pair (φ,ψ). The stabiliser of the identity 1_G for this action is the *coincidence group* $\mathrm{Coin}(\varphi,\psi)$, i.e. the subgroup of H given by

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

The TWISTEDCONJUGACY package provides methods to calculate Reidemeister classes, Reidemeister numbers and coincidence groups of pairs of group homomorphisms. These methods are implemented for finite groups and polycyclically presented groups. If H and G are both infinite polycyclically presented groups, then some methods in this package are only guaranteed to produce a result if either $G=H$ or G is nilpotent-by-finite. Otherwise, these methods may potentially throw an error: `"Error, no method found!"`

Bugs in this package, in GAP or any other package used directly or indirectly, may cause functions from this package to produce errors or even wrong results. You can set the variable `ASSERT@TwistedConjugacy` to `true`, which will cause certain functions to verify the correctness of their output. This should make results more (but not completely!) reliable, at the cost of some performance.

When using this package with PcpGroups, you can do the same for POLYCYCLIC's variables `CHECK_CENT@Polycyclic`, `CHECK_IGS@Polycyclic` and `CHECK_INTSTAB@Polycyclic`.

Chapter 2

Twisted Conjugacy

2.1 Twisted Conjugation Action

Let G,H be groups and $\varphi,\psi\colon H \to G$ group homomorphisms. Then the pair (φ,ψ) induces a (right) group action on G given by

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

This group action is called (φ,ψ)*-twisted conjugation*, and induces an equivalence relation on the group G. We say that $g_1, g_2 \in G$ are (φ,ψ)-twisted conjugate, denoted by $g_1 \sim_{\varphi,\psi} g_2$, if and only if there exists some element $h \in H$ such that $g_1 \cdot h = g_2$, or equivalently $g_1 = \psi(h) g_2 \varphi(h)^{-1}$.

If $\varphi\colon G \to G$ is an endomorphism of a group G, then by φ*-twisted conjugacy* we mean (φ, id_G)-twisted conjugacy. Most functions in this package will allow you to input a single endomorphism instead of a pair of homomorphisms. The "missing" endomorphism will automatically be assumed to be the identity mapping. Similarly, if a single group element is given instead of two, the second will be assumed to be the identity.

2.1.1 TwistedConjugation

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

Implements the twisted conjugation (right) group action induced by the pair of homomorphisms (*hom1*, *hom2*) as a function.

2.1.2 RepresentativeTwistedConjugation

▷ `RepresentativeTwistedConjugation(hom1[, hom2], g1[, g2])` (function)

Tests whether the elements *g1* and *g2* are twisted conjugate under the twisted conjugacy action of the pair of homomorphisms (*hom1*, *hom2*).

This function relies on the output of `RepresentativeTwistedConjugation`. Computes an element that maps *g1* to *g2* under the twisted conjugacy action of the pair of homomorphisms (*hom1*, *hom2*) or returns `fail` if no such element exists.

If G is abelian, this function relies on (a generalisation of) [DT21, Alg. 4]. If H is finite, it relies on a stabiliser-orbit algorithm. Otherwise, it relies on a mixture of the algorithms described in [Rom16, Thm. 3], [BKL+20, Sec. 5.4], [Rom21, Sec. 7] and [DT21, Alg. 6].

Example
```
gap> G := AlternatingGroup( 6 );;
gap> H := SymmetricGroup( 5 );;
gap> phi := GroupHomomorphismByImages( H, G, [ (1,2)(3,5,4), (2,3)(4,5) ],
>  [ (1,2)(3,4), () ] );;
gap> psi := GroupHomomorphismByImages( H, G, [ (1,2)(3,5,4), (2,3)(4,5) ],
>  [ (1,4)(3,6), () ] );;
gap> tc := TwistedConjugation( phi, psi );;
gap> g1 := (4,6,5);;
gap> g2 := (1,6,4,2)(3,5);;
gap> IsTwistedConjugate( psi, phi, g1, g2 );
false
gap> h := RepresentativeTwistedConjugation( phi, psi, g1, g2 );
(1,2)
gap> tc( g1, h ) = g2;
true
```

2.2 Reidemeister Classes

The equivalence classes of the equivalence relation $\sim_{\varphi,\psi}$ are called the *Reidemeister classes of* (φ,ψ) or the (φ,ψ)-*twisted conjugacy classes*. We denote the Reidemeister class of $g \in G$ by $[g]_{\varphi,\psi}$. The number of Reidemeister classes is called the Reidemeister number $R(\varphi,\psi)$ and is always a positive integer or infinity.

2.2.1 ReidemeisterClass

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

If *hom1* and *hom2* are group homomorphisms from a group H to a group G, this method creates the Reidemeister class of the pair (*hom1*, *hom2*) with representative *g*. The following attributes and operations are available:

- `Representative`, which returns *g*,
- `GroupHomomorphismsOfReidemeisterClass`, which returns the list [*hom1*, *hom2*],
- `ActingDomain`, which returns the group H,
- `FunctionAction`, which returns the twisted conjugacy action on G,
- `Random`, which returns a random element belonging to the Reidemeister class,
- `\in`, which can be used to test if an element belongs to the Reidemeister class,
- `List`, which lists all elements in the Reidemeister class if there are finitely many, otherwise returns `fail`,
- `Size`, which gives the number of elements in the Reidemeister class,
- `StabiliserOfExternalSet`, which gives the stabiliser of the Reidemeister class under the twisted conjugacy action.

2.2.2 ReidemeisterClasses

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

Returns a list containing the Reidemeister classes of (*hom1*, *hom2*) if the Reidemeister number $R($*hom1*,*hom2*$)$ is finite, or returns `fail` otherwise. It is guaranteed that the Reidemeister class of the identity is in the first position.

If G is abelian, this function relies on (a generalisation of) [DT21, Alg. 5]. If G and H are finite and G is not abelian, it relies on an orbit-stabiliser algorithm. Otherwise, it relies on (variants of) [DT21, Alg. 7].

This function is only guaranteed to produce a result if either $G = H$ or G is nilpotent-by-finite.

2.2.3 RepresentativesReidemeisterClasses

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

Returns a list containing representatives of the Reidemeister classes of (*hom1*, *hom2*) if the Reidemeister number $R($*hom1*,*hom2*$)$ is finite, or returns `fail` otherwise. It is guaranteed that the identity is in the first position.

The same remarks as for `ReidemeisterClasses` are valid here.

2.2.4 ReidemeisterNumber

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

Returns the Reidemeister number of (*hom1*, *hom2*), i.e. the number of Reidemeister classes.

If G is abelian, this function relies on (a generalisation of) [Jia83, Thm. 2.5]. If $G = H$, G is finite non-abelian and $\psi = \mathrm{id}_G$, it relies on [FH94, Thm. 5]. Otherwise, it uses the output of `ReidemeisterClasses`.

This function is only guaranteed to produce a result if either $G = H$ or G is nilpotent-by-finite.

Example
```
gap> tcc := ReidemeisterClass( phi, psi, g1 );
(4,6,5)^G
gap> Representative( tcc );
(4,6,5)
gap> GroupHomomorphismsOfReidemeisterClass( tcc );
[ [ (1,2)(3,5,4), (2,3)(4,5) ] -> [ (1,2)(3,4), () ],
[ (1,2)(3,5,4), (2,3)(4,5) ] -> [ (1,4)(3,6), () ] ]
gap> ActingDomain( tcc ) = H;
true
gap> FunctionAction( tcc )( g1, h );
(1,6,4,2)(3,5)
gap> Random( tcc ) in tcc;
true
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) ])
gap> ReidemeisterClasses( 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> RepresentativesReidemeisterClasses( phi, psi ){[1..7]};
[ (), (4,5,6), (4,6,5), (3,4)(5,6), (3,4,5), (3,4,6), (3,5,4) ]
gap> NrTwistedConjugacyClasses( phi, psi );
184
```

2.3 Reidemeister Spectra

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

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

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

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

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 $\mathrm{CSpec}_R(H,G)$, i.e.

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

If `H = G` this is also denoted by $\mathrm{CSpec}_R(G)$. 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 $\mathrm{TSpec}_R(G)$, i.e.

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

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

2.3.1 ReidemeisterSpectrum

▷ `ReidemeisterSpectrum(G)` (function)

Returns the Reidemeister spectrum of `G`.
If G is abelian, this function relies on the results from [Sen23].

2.3.2 ExtendedReidemeisterSpectrum

▷ `ExtendedReidemeisterSpectrum(G)` (function)

Returns the extended Reidemeister spectrum of `G`.

2.3.3 CoincidenceReidemeisterSpectrum

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

Returns the coincidence Reidemeister spectrum of `H` and `G`.

2.3.4 TotalReidemeisterSpectrum

▷ `TotalReidemeisterSpectrum(G)` (function)

Returns the total Reidemeister spectrum of `G`.

Example

```
gap> Q := QuaternionGroup( 8 );;
gap> D := DihedralGroup( 8 );;
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 ]
```

2.4 Reidemeister Zeta Functions

Let $\varphi, \psi \colon G \to G$ 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 (φ, ψ) is defined as

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

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

2.4.1 ReidemeisterZetaCoefficients

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

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

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

This function returns a list containing two sublists: the first sublist contains one period of the sequence $(P_n)_{n\in\mathbb{N}}$, the second sublist contains $(Q_n)_{n\in\mathbb{N}}$ up to the part where it becomes the constant zero sequence.

2.4.2 IsRationalReidemeisterZeta

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

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

2.4.3 ReidemeisterZeta

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

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

2.4.4 PrintReidemeisterZeta

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

Returns a string describing the Reidemeister zeta function of *endo1* and *endo2*. This is often more readable than evaluating `ReidemeisterZeta` in an indeterminate, and does not require rationality.

Example
```
gap> khi := GroupHomomorphismByImages( G, G, [ (1,2,3,4,5), (4,5,6) ],
>  [ (1,2,6,3,5), (1,4,5) ] );;
gap> ReidemeisterZetaCoefficients( khi );
[ [ 7 ], [  ] ]
gap> IsRationalReidemeisterZeta( khi );
true
gap> ReidemeisterZeta( khi );
function( s ) ... end
gap> s := Indeterminate( Rationals, "s" );;
gap> ReidemeisterZeta( khi )(s);
(1)/(-s^7+7*s^6-21*s^5+35*s^4-35*s^3+21*s^2-7*s+1)
gap> PrintReidemeisterZeta( khi );
"(1-s)^(-7)"
```

Chapter 3

Multiple Twisted Conjugacy Problem

3.1 The Multiple Twisted Conjugacy 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 $\varphi_i, \psi_i \colon H \to G_i$ be group homomorphisms. The multiple twisted conjugacy problem is the problem of finding some $h \in H$ such that $g_i = \psi_i(h) g_i' \varphi_i(h)^{-1}$ for all $i \in \{1,\dots,n\}$.

3.1.1 IsTwistedConjugateMultiple

▷ `IsTwistedConjugateMultiple(`*`hom1List[, hom2List], g1List[, g2List]`*`)` (function)

Verifies whether the multiple twisted conjugacy problem for the given homomorphisms and elements has a solution.

3.1.2 RepresentativeTwistedConjugationMultiple

▷ `RepresentativeTwistedConjugationMultiple(`*`hom1List[, hom2List], g1List[, g2List]`*`)` (function)

Computes a solution to the multiple twisted conjugacy problem for the given homomorphisms and elements, or returns `fail` if no solution exists.

Example

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

```
>  [ (1,5)(4,6), (1,4)(3,5) ], [ (1,4,5,3,6), (2,4,5,6,3) ] );
(1,2)
```

Chapter 4

Homomorphisms

4.1 Representatives of homomorphisms between groups

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

4.1.1 RepresentativesAutomorphismClasses

▷ `RepresentativesAutomorphismClasses(G)` (function)

Let *G* be a group. This command returns a list of the automorphisms of *G* up to composition with inner automorphisms.

4.1.2 RepresentativesEndomorphismClasses

▷ `RepresentativesEndomorphismClasses(G)` (function)

Let *G* be a group. This command 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. For 2-generated groups, this function takes its source code from `AllHomomorphismClasses`.

4.1.3 RepresentativesHomomorphismClasses

▷ `RepresentativesHomomorphismClasses(H, G)` (function)

Let *G* and *H* be groups. This command returns a list of the homomorphisms from *H* to *G*, up to composition with inner automorphisms of *G*. This does the same as calling `AllHomomorphismClasses(H,G)`, but should be faster for abelian and non-2-generated groups. For 2-generated groups, this function takes its source code from `AllHomomorphismClasses`.

Example
```
gap> G := SymmetricGroup( 6 );;
gap> Auts := RepresentativesAutomorphismClasses( G );;
gap> Size( Auts );
2
gap> ForAll( Auts, IsGroupHomomorphism and IsEndoMapping and IsBijective );
true
```

```
gap> Ends := RepresentativesEndomorphismClasses( G );;
gap> Size( Ends );
6
gap> ForAll( Ends, IsGroupHomomorphism and IsEndoMapping );
true
gap> H := SymmetricGroup( 5 );;
gap> Homs := RepresentativesHomomorphismClasses( H, G );;
gap> Size( Homs );
6
gap> ForAll( Homs, IsGroupHomomorphism );
true
```

4.2 Coincidence and Fixed Point Groups

4.2.1 FixedPointGroup

▷ `FixedPointGroup(endo)` (function)

Let *endo* be an endomorphism of a group G. This command returns the subgroup of G consisting of the elements fixed under the endomorphism *endo*.

This function does the same as `CoincidenceGroup`(*endo*,id_G).

4.2.2 CoincidenceGroup

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

Let *hom1*, *hom2*, ... be group homomorphisms from a group H to a group G. This command returns the subgroup of H consisting of the elements h for which h^*hom1* = h^*hom2* = ...

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

Example
```
gap> phi := GroupHomomorphismByImages( G, G, [ (1,2,5,6,4), (1,2)(3,6)(4,5) ],
>  [ (2,3,4,5,6), (1,2) ] );;
gap> Set( FixedPointGroup( phi ) );
[ (), (1,2,3,6,5), (1,3,5,2,6), (1,5,6,3,2), (1,6,2,5,3) ]
gap> psi := GroupHomomorphismByImages( H, G, [ (1,2,3,4,5), (1,2) ],
>  [ (), (1,2) ] );;
gap> khi := GroupHomomorphismByImages( H, G, [ (1,2,3,4,5), (1,2) ],
>  [ (), (1,2)(3,4) ] );;
gap> CoincidenceGroup( psi, khi ) = AlternatingGroup( 5 );
true
```

4.3 Induced and restricted group homomorphisms

4.3.1 InducedHomomorphism

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

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 induced by *hom* via *epi1* and *epi2*, that is, the homomorphism from Q to P which maps h^*epi1* to (h^*hom*)^*epi2*, for any element h of H. This generalises `InducedAutomorphism` to homomorphisms.

4.3.2 RestrictedHomomorphism

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

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 induced by *hom*. This is similar to `RestrictedMapping`, but the range is explicitly set to *M*.

Example

```
gap> G := PcGroupCode( 1018013, 28 );;
gap> phi := GroupHomomorphismByImages( G, G, [ G.1, G.3 ],
>  [ G.1*G.2*G.3^2, G.3^4 ] );;
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> Source( ind ) = Range( p ) and Range( ind ) = Range( p );
true
gap> res := RestrictedHomomorphism( phi, N, N );
[ f3 ] -> [ f3^4 ]
gap> Source( res ) = N and Range( res ) = N;
true
```

Chapter 5

Cosets

Please note that the functions below are implemented only for PcpGroups. They are (currently) very inefficient, so use with caution.

5.1 Intersection of cosets in PcpGroups

5.1.1 Intersection

▷ `Intersection(C1, C2, ...)` (function)
▷ `Intersection(list)` (function)
▷ `Intersection2(C1, C2)` (operation)

Calculates the intersection of the (right) cosets `C1`, `C2`, ... Alternatively, `list` may be a list of (right) cosets. This intersection is either a new coset, or an empty list.

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

5.2 Membership in double cosets in PcpGroups

5.2.1 \in (for IsPcpElement, IsDoubleCoset)

▷ `\in(g, D)` (operation)

Given an element `g` of a PcpGroup and a double coset `D` of that same group, this function tests whether `g` is an element of `D`.

Example
```
gap> HxK := DoubleCoset( H, x, K );;
gap> G.1 in HxK;
```

```
false
gap> G.2 in HxK;
true
```

References

[BKL+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*. De Gruyter, 2020. 5, 14

[DT21] K. Dekimpe and S. Tertooy. Algorithms for twisted conjugacy classes of polycyclic-by-finite groups. *Topology Appl.*, 293:107565, 2021. 5, 7

[FH94] A. Fel'shtyn and R. Hill. The Reidemeister zeta function with applications to Nielsen theory and a connection with Reidemeister torsion. *K-Theory*, 8(4):367–393, 1994. 7

[Jia83] B. Jiang. *Lectures on Nielsen fixed point theory*, volume 14 of *Contemp. Math.* Amer. Math. Soc., 1983. 7

[Rom16] V. Roman'kov. On solvability of equations with endomorphisms in nilpotent groups. *Sib. Elektron. Mat. Izv.*, 13:716–725, 2016. 5, 14

[Rom21] V. Roman'kov. Algorithmic theory of solvable groups. *Prikl. Diskretn. Mat.*, 52:16–64, 2021. 5, 14

[Sen23] P. Senden. The Reidemeister spectrum of finite abelian groups. *Proc. Edinburgh Math. Soc.*, 66(4):940–959, 2023. 8

Index