walrus

A new approach to proving hyperbolicity

0.9991

21 February 2022

Markus Pfeiffer

Markus Pfeiffer
Email: markus.pfeiffer@st-andrews.ac.uk
Homepage: http://www.morphism.de/~markusp/
Address: School of Computer Science
University of St Andrews
Jack Cole Building, North Haugh
St Andrews, Fife, KY16 9SX
United Kingdom

Contents

Chapter 1

Overview

1.1 Examples

1.1.1 TriangleGroup

▷ `TriangleGroup(p, q, r)` (function)

Returns: a pregroup presentation

Returns a pregroup presentation for the (p,q,r)-triangle group, the pregroup is the pregroup of the free product of a cyclic group of order p generated by x and a cyclic group of order q generated by y together with the relation $(xy)^r$.

Example

```
gap> T := TriangleGroup(2,3,7);
<pregroup presentation with 3 generators and 1 relators>
gap> Pregroup(T);
<pregroup with 4 elements in table rep>
gap> Relators(T);
[ <pregroup relator ([ "2", "3" ])^7> ]
gap> T := TriangleGroup(17,22,100);
<pregroup presentation with 37 generators and 1 relators>
gap> Pregroup(T);
<pregroup with 38 elements in table rep>
gap> Relators(T);
[ <pregroup relator ([ "2", "18" ])^100> ]
gap> IsHyperbolic(T, 1/6);
true
```

1.1.2 TriangleCommutatorQuotient

▷ `TriangleCommutatorQuotient(m, n)` (function)

▷ `TriSH(arg)` (function)

Returns: a pregroup presentation

Returns a pregroup presentation of the quotient of the $(2,3,m)$-triangle group by the normal subgroup generated by nth power of the commutator of x and y. The name `TriSH` is a synonym for `TriangleCommutatorQuotient` provided for backwards compatibility and might be removed in the future.

1.1.3 RandomTriangleQuotient

▷ `RandomTriangleQuotient(p, q, r, len)` (function)

Returns: a pregroup presentation

Returns the quotient of the (p,q,r)-triangle group by a random relator of length `len`.

1.1.4 JackButtonGroup

▷ `JackButtonGroup()` (function)

Returns: a pregroup presentation

The Jack-Button group, as suggested to me by Alan Logan. It is known to be hyperbolic, but the tester fails for it. The pregroup is the pregroup of the free group of rank 3 with generators a,b, and t and two relators $t^{-1}atb^{-1}a^{-1}$ and $t^{-1}ata^{-1}b^{-1}$.

Example

```
gap> J := JackButtonGroup();
<pregroup presentation with 6 generators and 2 relators>
gap> Pregroup(J);
<pregroup of free group of rank 3>
gap> Relators(J);
[ <pregroup relator TatBA>, <pregroup relator TatAB> ]
```

1.1.5 RandomPregroupPresentation

▷ `RandomPregroupPresentation(pg, nrel, lrel)` (function)

Returns: a pregroup presentation

Returns a pregroup presentation over the given pregroup *pg* with `nrel` randomly chosen relators of length `lrel`.

1.1.6 RandomPregroupWord

▷ `RandomPregroupWord(pg, len)` (function)

Returns: a list of integers

A random list of pregroup element numbers of the pregroup `pregroup` of length `len`. When interpreted as a pregroup word this is cyclically reduced.

This package provides the operations `IsHyperbolic`, ways of testing a finitely presented group for hyperbolicity in the sense of Gromov.

The algorithm is based on ideas by Richard Parker, and the theory is described in the paper "Polynomial time proofs that groups are hyperbolic".

1.2 Testing Hyperbolicity

The main function of this package is the so-called RSym-tester. Given a (pregroup) presentation of a group, this function will try to prove whether the group defined by the presentation is hyperbolic, and will give an answer in polynomial time. Since hyperbolicity is undecidable, the answer can be positive, negative, or inconclusive.

As a simple example consider the following. Triangle groups are known to be hyperbolic when the sum $\frac{1}{p}+\frac{1}{q}+\frac{1}{r}$ is less than 1. The parameter for `IsHyperbolic` (3.5.3) gives the algorithm a hint how hard it should try.

Example
```
gap> triangle := TriangleGroup(2,3,7);
<pregroup presentation with 3 generators and 1 relators>
gap> IsHyperbolic(triangle, 1/6);
true
gap> triangle := TriangleGroup(3,3,3);
<pregroup presentation with 3 generators and 1 relators>
gap> IsHyperbolic(triangle, 1/6);
[ fail, [ [ 1, 0, 0, 0 ], [ 2, 1, 1, 1/36 ], [ 1, 2, 2, 1/18 ],
          [ 2, 3, 3, 1/12 ], [ 1, 4, 4, 1/9 ], [ 2, 5, 5, 5/36 ],
          [ 1, 6, 6, 1/6 ] ], [ 2, 5, 5, 5/36 ] ]
```

One can also create pregroup presentations by giving a pregroup and relators, that is, words over the pregroup.

Example
```
gap> G1 := CyclicGroup(3);;
gap> pg := PregroupOfFreeProduct(G1,G1);
<pregroup with 5 elements in table rep>
gap> rel := [2,5,3,4,3,4,3,4,3,5,2,4,3,5,2,4,3,5,3,4,2,4,3,5];
[ 2, 5, 3, 4, 3, 4, 3, 4, 3, 5, 2, 4, 3, 5, 2, 4, 3, 5, 3, 4, 2, 4, 3, 5 ]
gap> pgp := NewPregroupPresentation(pg,[pg_word(pg,rel)]);
<pregroup presentation with 4 generators and 1 relators>
gap> res := RSymTest(pgp, 0);;
gap> res[1];
fail
```

1.3 The MAGMA-compatible interface

An implementation of the hyperbolicity testing algorithm and word-problem solver exist in MAGMA as well. For ease of comparison between the results these two systems give, walrus contains an interface that aims to be compatible with MAGMA's. Please refer to MAGMA's documentation for further details.

Example
```
gap> F := FreeGroup("x", "y");;
gap> AssignGeneratorVariables(F);;
gap> rred := [ y^3 ];;
gap> rgreen := [ x^4, (x*y)^4 ];;
gap> IsHyperbolic(F, rred, rgreen, 1/10);
[ fail, [ [ 1, 0, 0, 0 ], [ 2, 1, 1, 13/120 ], [ 1, 2, 2, 13/60 ],
           [ 2, 3, 3, 13/40 ], [ 1, 4, 4, 13/30 ] ], [ 2, 3, 3, 13/40 ] ]
```

Chapter 2

Pregroups

Pregroups are the fundamental building block of pregroup presentations used in the hyperbolicity tester.

2.1 Creating Pregroups

This section describes functions to create pregroups from multiplication tables, free groups, and free products of finite groups.

2.1.1 PregroupByTable

▷ `PregroupByTable(`*`enams, table`*`)` (function)
▷ `PregroupByTableNC(`*`arg`*`)` (function)

Returns: A pregroup

If *`enams`* is a list of element names, which can be arbitrary GAP objects, with the convention that `enams[1]` is the name of the identity element, and *`table`* is a square table of non-negative integers that is the multiplication table of a pregroup, then `PregroupByTable` and `PregroupByTableNC` return a pregroup in multiplication table representation.

By convention the elements of the pregroup are numbered `[1..n]` with `0` denoting an undefined product in the table.

The axioms for a pregroup are checked by `PregroupByTable` and not checked by `PregroupByTableNC`.

Example
```
gap> pregroup := PregroupByTable( "1xyY",
>                [ [1,2,3,4]
>                , [2,1,0,0]
>                , [3,4,0,1]
>                , [4,0,1,3] ] );
<pregroup with 4 elements in table rep>
```

2.1.2 PregroupByRedRelators (for IsFreeGroup, IsList, IsList)

▷ `PregroupByRedRelators(`*`F, rrel, inv`*`)` (operation)

Returns: A pregroup in table representation

Construct a pregroup from the list *rrel* of red relators and the list *inv* of involutions over the free group *F*. The argument *rred* has to be a list of elements of length 3 in the free group *F*, and *inv* has to be a list of generators of *F*.

2.1.3 PregroupOfFreeProduct (for IsGroup, IsGroup)

▷ `PregroupOfFreeProduct(G, H)` (operation)

Construct the pregroup of the free product of *G* and *H*. If *G* and *H* are finite groups, then `PregroupOfFreeProduct` returns the pregroup consisting of the non-identity elements of *G* and *H* and an identity element. A product between two non-trivial elements is defined if and only if they are in the same group.

Example

```
gap> pregroup := PregroupOfFreeProduct(SmallGroup(12,2), SmallGroup(24,4));
<pregroup with 35 elements in table rep>
```

2.1.4 PregroupOfFreeGroup

▷ `PregroupOfFreeGroup(F)` (function)

Return the pregroup of the free group *F*

2.2 Filters and Representations

This section gives an overview over the filters, categories and representations defined by walrus

2.2.1 IsPregroup (for IsObject and IsCollection)

▷ `IsPregroup(arg)` (filter)

Returns: `true` or `false`

2.2.2 IsPregroupTableRep (for IsPregroup andIsComponentObjectRep andIsAttributeStoringRep)

▷ `IsPregroupTableRep(arg)` (filter)

Returns: `true` or `false`

A pregroup represented by its multiplication table, which is a square table of integers between 0 and the size of the pregroup, where 0 represents an undefined multiplication.

2.2.3 IsPregroupOfFreeGroupRep (for IsPregroup andIsComponentObjectRep andIsAttributeStoringRep)

▷ `IsPregroupOfFreeGroupRep(arg)` (filter)

Returns: `true` or `false`

Pregroup of a free group of rank k. The only defined products are $1 \cdot x = x \cdot 1 = x$ and $xx^{-1} = x^{-1}x = 1$, for all generators x.

2.2.4 IsPregroupOfFreeProductRep (for IsPregroup andIsComponentObjectRep andIsAttributeStoringRep)

▷ `IsPregroupOfFreeProductRep(`*arg*`)` (filter)

Returns: `true` or `false`

Pregroup of the free product of a list of groups where products between non-trivial elements g, h are defined if g,h are contained in the same group.

2.3 Attributes, Properties, and Operations

This section gives an overview over the attributes, properties, and operatins defined for pregroups.

2.3.1 [] (for IsPregroup, IsInt)

▷ `[](`*pregroup, i*`)` (operation)

Get the `i`th element of *pregroup*. By convention the 1st element is the identity element.

2.3.2 IntermultPairs (for IsPregroup)

▷ `IntermultPairs(`*pregroup*`)` (attribute)

Returns the set of intermult pairs of the pregroup

2.3.3 One (for IsPregroup)

▷ `One(`*pregroup*`)` (attribute)

The identity element of *pregroup*.

2.3.4 MultiplicationTable (for IsPregroup)

▷ `MultiplicationTable(`*pregroup*`)` (attribute)

The multiplication table of *pregroup*

2.3.5 SetPregroupElementNames (for IsPregroup, IsList)

▷ `SetPregroupElementNames(`*pregroup, names*`)` (operation)

Can be used to set more user-friendly display names for the elements of *pregroup*. The list *names* has to be of length `Size(`*pregroup*`)`.

2.3.6 PregroupElementNames (for IsPregroup)

▷ `PregroupElementNames(`*pregroup*`)` (operation)

Return the list of names of elements of *pregroup*

2.4 Elements of Pregroups

2.4.1 IsElementOfPregroup (for IsMultiplicativeElementWithInverse)

▷ `IsElementOfPregroup(arg)` (filter)

Returns: `true` or `false`

2.4.2 IsElementOfPregroupRep (for IsElementOfPregroup and IsComponentObjectRep)

▷ `IsElementOfPregroupRep(arg)` (filter)

Returns: `true` or `false`

2.4.3 IsElementOfPregroupOfFreeGroupRep (for IsElementOfPregroup and IsComponentObjectRep)

▷ `IsElementOfPregroupOfFreeGroupRep(arg)` (filter)

Returns: `true` or `false`

2.4.4 PregroupOf (for IsElementOfPregroup)

▷ `PregroupOf(p)` (attribute)

The pregroup that the element *p* is contained in.

2.4.5 IsDefinedMultiplication (for IsElementOfPregroup, IsElementOfPregroup)

▷ `IsDefinedMultiplication(p, q)` (operation)

Tests whether the multiplication of *p* and *q* is defined in the pregroup containing *p* and *q*.

2.4.6 IsIntermultPair (for IsElementOfPregroup, IsElementOfPregroup)

▷ `IsIntermultPair(p, q)` (operation)

Tests whether (p,q) is an intermult pair. defined.

2.4.7 PregroupInverse (for IsElementOfPregroup)

▷ `PregroupInverse(p)` (attribute)

Return the inverse of *p*.

2.5 Small Pregroups

This package contains a small database of pregroups of sizes 1 to 7. The database was computed by Chris Jefferson using the Minion constraint solver.

These small pregroups currently used for testing. Accessing the small pregroups database works as follows.

2.5.1 NrSmallPregroups

▷ `NrSmallPregroups(n)` (function)

Returns: an integer.

Returns the number of pregroups of size `n` available in the database.

2.5.2 SmallPregroup

▷ `SmallPregroup(n, i)` (function)

Returns: a pregroup.

Returns the `i`th pregroup of size `n` from the database of small pregroups.

Chapter 3

Pregroup Presentations

3.1 Concepts

Given a pregroup P there is a universal group $\mathscr{U}(P)$ that contains P. The concept of a pregroup presentation is a generalisation of presentations over the free group, that is a pregroup presentation is a way of defining a group as a quotient of a universal group over a pregroup by giving relator words over the pregroup.

For the purposes of the RSym tester we introduce some more concepts.

3.1.1 Locations

A *location* on a pregroup relator $w = a_1 a_2 \dots a_n$ is an index i between 1 and n and denotes the location between a_i (the `InLetter` (3.2.2)) and a_{i+1} (the `OutLetter` (3.2.3)), where the relator is considered cyclically, that is, when $i = n$ then the outletter is a_1.

3.1.2 Places

A *place* $R(L,x,C)$ on a pregroup relator R is a location (3.1.1) together with a letter from the pregroup and a colour, which is either *red* or *green*.

3.2 Attributes

3.2.1 IsPregroupLocation (for IsObject)

▷ `IsPregroupLocation(`*arg*`)` (filter)

Returns: `true` or `false`

3.2.2 InLetter (for IsPregroupLocation)

▷ `InLetter(`*arg*`)` (attribute)

3.2.3 OutLetter (for IsPregroupLocation)

▷ `OutLetter(arg)` (attribute)

3.2.4 Places (for IsPregroupLocation)

▷ `Places(arg)` (attribute)

3.2.5 NextLocation (for IsPregroupLocation)

▷ `NextLocation(arg)` (attribute)

3.2.6 PrevLocation (for IsPregroupLocation)

▷ `PrevLocation(arg)` (attribute)

3.2.7 __ID (for IsPregroupLocation)

▷ `__ID(arg)` (attribute)

3.3 Creating Pregroup Presentations

3.3.1 NewPregroupPresentation

▷ `NewPregroupPresentation(pregroup, relators)` (function)

Returns: a pregroup presentation

Creates a pregroup presentation over the *pregroup* with relators *relators*.

3.3.2 PregroupPresentationFromFp

▷ `PregroupPresentationFromFp(F, rred, rgreen)` (function)

Returns: a pregroup presentation

Creates a pregroup presentation over the pregroup defined by *F* and *rred* with relators *rgreen*.

3.3.3 PregroupPresentationToFpGroup

▷ `PregroupPresentationToFpGroup(presentation)` (function)

Returns: a finitely presented group

Converts the pregroup presentation *presentation* into a finitely presented group.

3.4 Filters, Attributes, and Properties

3.4.1 IsPregroupPresentation (for IsObject)

▷ `IsPregroupPresentation(arg)` (filter)

Returns: `true` or `false`

3.4.2 (for IsPregroupPresentation and IsComponentObjectRep and IsAttributeStoringRep)

▷ `(arg)` (filter)

Returns: `true` or `false`

3.5 Hyperbolicity testing for pregroup presentations

3.5.1 RSymTestOp (for IsPregroupPresentation, IsRat)

▷ `RSymTestOp(presentation, epsilon)` (operation)

Test the group presented by *presentation* for hyperbolicity using the RSym tester with parameter *epsilon*.

3.5.2 RSymTest

▷ `RSymTest(args...)` (function)

This is a wrapper for `RSymTestOp` (3.5.1). If the first argument given is a free group, the second and third lists of words over the free group, and the fourth a rational, then this function creates a pregroup presentation from the input data and invokes `RSymTestOp` (3.5.1) on it. If the first argument is a pregroup presentation and the second argument is rational number, then it invokes `RSymTestOp` (3.5.1) on that input.

3.5.3 IsHyperbolic (for IsPregroupPresentation)

▷ `IsHyperbolic(presentation)` (operation)
▷ `IsHyperbolic(presentation, epsilon)` (operation)
▷ `IsHyperbolic(F, rred, rgreen, epsilon)` (operation)

Tests a given presentation for hyperbolicity using the RSym test procedure.

3.6 Input and Output of Pregroup Presentations

3.6.1 PregroupPresentationToKBMAG

▷ `PregroupPresentationToKBMAG(presentation)` (function)

Returns: A KBMAG rewriting system

Turns the pregroup presentation *presentation* into valid input for Knuth-Bendix rewriting using KBMAG. Only available if the kbmag package is available.

3.6.2 PregroupPresentationToStream

▷ `PregroupPresentationToStream(stream, presentation)` (function)

Writes the pregroup presentation *presentation* to *stream*.

Example
```
gap> T := TriangleGroup(2,3,7);;
gap> str := "";; stream := OutputTextString(str, true);;
gap> PregroupPresentationToStream(stream, T);
gap> Print(str);
rec(
  rels := [ [ 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3 ] ],
  table := [ [ 1, 2, 3, 4 ], [ 2, 1, 0, 0 ], [ 3, 0, 4, 1 ], [ 4, 0, 1, 3 ] ] );
```

3.6.3 PregroupPresentationFromStream

▷ `PregroupPresentationFromStream(stream)` (function)

Returns: A pregroup presentation

Reads a pregroup presentation from an input stream in the same format that `PregroupPresentationToStream` (3.6.2) uses.

Example
```
gap> stream := InputTextString(str);
InputTextString(0,146)
gap> PregroupPresentationFromStream(stream);
<pregroup presentation with 3 generators and 1 relators>
```

3.6.4 PregroupPresentationToSimpleStream

▷ `PregroupPresentationToSimpleStream(stream, presentation)` (function)

Writes the pregroup presentation *presentation* to *stream*. Uses a simpler format than `PregroupPresentationToStream` (3.6.2)

3.6.5 PregroupPresentationToFile

▷ `PregroupPresentationToFile(filename, presentation)` (function)

Writes the pregroup presentation *presentation* to file with name *filename*.

3.6.6 PregroupPresentationFromFile

▷ `PregroupPresentationFromFile(filename)` (function)

Reads a pregroup presentation from file with *filename*.

3.6.7 PregroupPresentationToSimpleFile

▷ `PregroupPresentationToSimpleFile(stream, presentation)` (function)

Writes the pregroup presentation *presentation* to file with name *filename* in a simple format.

Index