The HallPoly Package

Version 1.0

March 2018

Alexander Cant
Bettina Eick

Alexander Cant Email: a.cant@tu-braunschweig.de

Bettina Eick Email: beick@tu-braunschweig.de

Address: AG Algebra und Diskrete Mathematik
Institut Computational Mathematics
Universitätsplatz 2
38106 Braunschweig
(Germany)

Copyright

Contents

Chapter 1

Introduction

Let G be a finitely generated torsion free nilpotent group (*T-group* for short). Choose $g_1, g_2, \ldots, g_n \in G$ such that the subgroups $G_i := \langle g_i, \ldots, g_n \rangle$ form a central series for G with infinite cyclic factors. n is unique and is called the *Hirsch-length* of G. $(g_1, \ldots, g_n)$ is called a *T-basis* for G. For every $h \in G$ there exist unique $e_1, \ldots, e_n \in \mathbb{Z}$ with $h = g_1^{e_1} \cdots g_n^{e_n}$; this is called the *normal form* of h (w.r.t. $g_1, \ldots, g_n$). Hence there are functions $F_i : \mathbb{Z}^n \times \mathbb{Z}^n \to \mathbb{Z}$ and $K_i : \mathbb{Z}^n \times \mathbb{Z} \to \mathbb{Z}$ for $1 \leq i \leq n$ with

$$(g_1^{a_1} \cdots g_n^{a_n}) \cdot (g_1^{a_1} \cdots g_n^{a_n}) = g_1^{F_1(a,b)} \cdots g_n^{F_n(a,b)}$$

and

$$(g_1^{a_1} \cdots g_n^{a_n})^x = g_1^{K_1(a,x)} \cdots g_n^{K_n(a,x)}$$

for all $a = (a_1, \ldots, a_n), b = (b_1, \ldots, b_n) \in \mathbb{Z}^n$ and $x \in \mathbb{Z}$. Philip Hall showed in [Hal57] that the functions F_i and K_i can be described as rational polynomials in $2n$ resp. $n+1$ indeterminates for $1 \leq i \leq n$. Hence the functions F_i and K_i are called *Hall-polynomials*. In particular the polynomials F_i are called *multiplication polynomials* and the polynomials K_i are called *power polynomials*.

The purpose of the HallPoly Package is to compute Hall-polynomials for arbitrary T-groups of arbitrary Hirsch-length. To do so, the T-groups need to be uniformly presented. Assume G is a T-group and $(g_1, \ldots, g_n)$ a T-basis for G. Then there is a unique $t = (t_{i,j,k} | 1 \leq i < j < k \leq n) \in \mathbb{Z}^{\binom{n}{3}}$ so that

$$g_j g_i = g_i g_j g_{j+1}^{t_{i,j,j+1}} \cdots g_n^{t_{i,j,n}}$$

for all $1 \leq i < j < n$. One can show that G has the presentation

$$G(t) := \langle g_1, \ldots, g_n \mid g_j g_i = g_i g_j g_{j+1}^{t_{i,j,j+1}} \cdots g_n^{t_{i,j,n}} \ (1 \leq i < j \leq n) \rangle.$$

Hence it is possible to describe any T-group G of Hirsch-length n by $\binom{n}{3}$ parameters $t_{i,j,k} \in \mathbb{Z}$.

Vice versa any group defined by a presentation of the form $G(t)$ for $t \in \mathbb{Z}^{\binom{n}{3}}$ is finitely generated and nilpotent with a Hirsch-length less or equal n. The group is torsion free (and therefore a T-group) iff it has Hirsch-length n.

The presentation $G(t)$ with $t \in \mathbb{Z}^{\binom{n}{3}}$ is called *consistent*, if the group defined by $G(t)$ has Hirsch-length n. The previous arguments imply that each T-group can be described via a consistent presentation. Hence we consider only consistent presentations.

Chapter 2

Main function

In this chapter we describe the main function of the HallPoly package.

2.1 Functions concerning the presentation

2.1.1 GroupByVector

▷ `GroupByVector(n, t)` (function)

This function returns the group of Hirsch-length n with presentation $G(t)$. `t` $\in \mathbb{Z}^{\binom{n}{3}}$ has to describe a consistent presentation. `t` needs to be an array so that `t[i][j][k]` $= t_{i,j,k}$.

2.1.2 VectorByGroupPcp

▷ `VectorByGroupPcp(G)` (function)

This function returns parameters `t` $\in \mathbb{Z}^{\binom{n}{3}}$ (where n is the Hirsch-length of G) so that G has the presentation $G($`t`$)$. G has to be a T-group given by generators that form a T-basis.

2.1.3 VectorByGroupCS

▷ `VectorByGroupCS(G)` (function)

This function returns parameters `t` $\in \mathbb{Z}^{\binom{n}{3}}$ (where n is the Hirsch-length of G) so that G has the presentation $G($`t`$)$. G has to be a T-group.

2.1.4 HallRecByVector

▷ `HallRecByVector(t)` (function)

This function returns a record containing the Hall polynomials for the group $G(t)$, where $t \in \mathbb{Z}^{\binom{n}{3}}$. `t` needs to be an array so that `t[i][j][k]` $= t_{i,j,k} \in \mathbb{Z}$. The parameterised Hall polynomials for the Hirsch lengths up to n have to be known.

2.2 General tools

2.2.1 SolveRec

▷ `SolveRec(g, v)` (function)

This function returns the unique rational polynomial $f(v)$ that satifies $f(0) = 0$ and $f(v+1) = f(v) + g(v)$ for every positive integer v. `v` is an indeterminate and g contains the coefficients of `g(v)` (i.e. $g(v) = \sum_{i=0}^{m} g[i+1]\, v^i$).

Example

```
gap> v := Indeterminate(Rationals, "v");;
gap> g := [1,1];;
gap> SolveRec(g, v);
1/2*v^2+1/2*v
```

2.2.2 PolyDecomp

▷ `PolyDecomp(f)` (function)

This function returns the degree and the number of monomials of `f`, where `f` is a polynomial in the variables `A[1],...,A[n],B[1],...,B[n],x` with parameters `T[i][j][k]`.

The degree is defined as the maximal degree of a monomial in `f` and the degree of a monomial is the sum of the exponents of its indeterminates.

2.3 Computing Hall-polynomials

2.3.1 HallConstruction

▷ `HallConstruction(n, gb)` (function)

This calls the algorithm that computes new parameterized Hall-polynomials for Hirsch-length `n`. Parameterized Hall-polynomials up to Hirsch-length `n-1` have to be stored in the record `HallRec`. The result is stored in `HallRec`. `gb` is a boolean that specifies, whether Groebner bases should be used for reducing the computed polynomials.

Example

```
gap> HallConstruction(5,true);
 - Step 5: compute mult-polynomial
    compute r[i][j] for 1 by 2
    compute r[i][j] for 1 by 3
    compute r[i][j] for 1 by 4
    compute r[i][j] for 1 by 5
    compute r[i][j] for 2 by 3
    compute r[i][j] for 2 by 4
    compute r[i][j] for 2 by 5
    compute r[i][j] for 3 by 4
    compute r[i][j] for 3 by 5
    compute r[i][j] for 4 by 5
    starting symbolic collection
    got poly with 69 summands
 - Step 5: reduce mult-polynomial via inherit
```

```
   mult poly has 44 summands
- Step 5: compute new consistency relations
get Groebner in HallConsistency
... done
- Step 5: reduce mult polynomial via consistency
   mult poly has 44 summands
- Step 5: compute power polynomial
   extract terms
   solve recursion
- Step 5: reduce power polynomial
   power poly has 84 summands
- Step 5: done
```

2.3.2 HallMult

▷ `HallMult(t, a, b)` (function)

If `t` is a vector of parameters (either with a consistent presentation or indeterminates) of length `n` and `a` and `b` are vectors of length `n` (either integer values or indeterminates), then this function returns the multiplication polynomials for that case.

The parameterised Hall-polynomials for Hirsch length `n` have to be known.

2.3.3 HallPower

▷ `HallPower(t, a, x)` (function)

If `t` is a vector of parameters (either with a consistent presentation or indeterminates) of length `n` and `a` is a vector of length `n` (either integer values or indeterminates) and `x` is an integer or an indeterminate, then this function returns the power polynomials for that case.

The parameterised Hall-polynomials for Hirsch length `n` have to be known.

2.4 Methods for storing things

2.4.1 SaveHallPolynomials

▷ `SaveHallPolynomials(n)` (function)

This function stores the multiplication and power polynomials plus the consistency relations for Hirsch-length `n` (stored in *`HallRec[n]`*) in the file *`hallpoly/lib/hrln.gi`*. If the file already exists or the Hall-polynomials for Hirsch-length *n* haven't been computed yet, an error message is returned.

Example

```
gap> SaveHallPolynomials(6);
true
gap> SaveHallPolynomials(8);
There are no Hall-polynomials to store.
false
```

2.4.2 SaveParameters

▷ SaveParameters(*t*, *file*) (function)

This function stores the parameters *t* in a file named *file* in the directory *hallpoly/lib*. *t* has to be an array of size $n \times n \times n$.

Example

```
gap> t := GenT(6);;
gap> SaveParameters(t,"test");
true
```

2.4.3 CheckFileExists

▷ CheckFileExists(*n*) (function)

This function returns, whether the file for the Hall-polynomials of Hirsch-length *n* in *hallpoly/lib* already exists.

Example

```
gap> CheckFileExists(6);
true
```

2.5 Runtime test

2.5.1 RuntimeTestHallColl

▷ RuntimeTestHallColl(a, b) (function)

This function compares the runtime of evaluating Hall polynomials and the standard collection algorithm. To do so, the function computes *a* random subgroups in the group of 6×6 unitriangular matrices over $\mathbb{Z}$ with Hirsch length ≤ 7 and *b* random products in each subgroup.

Returns the overall runtimes in milliseconds.

Example

```
gap> RuntimeTestHallColl(2,4);
 Looking for subgroup 1...
 Found a subgroup of Hirsch length 7.
  Hall polynomials: 0ms, Collection: 12400ms
  Hall polynomials: 4ms, Collection: 2992ms
  Hall polynomials: 4ms, Collection: 93384ms
  Hall polynomials: 4ms, Collection: 608ms

 Looking for subgroup 2...
 Found a subgroup of Hirsch length 6.
  Hall polynomials: 0ms, Collection: 36ms
  Hall polynomials: 0ms, Collection: 8ms
  Hall polynomials: 0ms, Collection: 100ms
  Hall polynomials: 0ms, Collection: 36ms

 Overall runtime: Hall polynomials:  0:00:00.012, Collection:  0:01:49.564
[ 12, 109564 ]
```

References

[Hal57] Philip Hall. *The Edmonton Notes on Nilpotent Groups*. Queen Mary College, London, 1957. 4

Index