subgroupladders

This package provides an algorithm that computes a subgroup ladder from a permutation group up to the parent symmetric group.

0.1

8 November 2018

Ulli Kehrle

Friedrich Rober

Ulli Kehrle
Email: ulli.kehrle@rwth-aachen.de

Friedrich Rober
Email: friedrich.rober@rwth-aachen.de

Contents

Chapter 1

Introduction

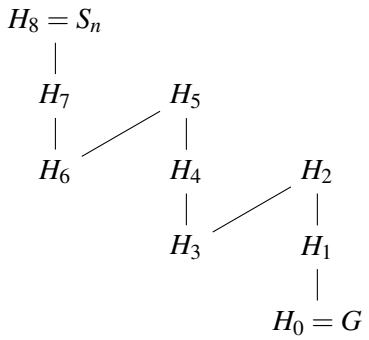
This package provides an algorithm that computes a subgroup ladder from a permutation group up to the parent symmetric group. There is also a function included which computes portions of the table of marks using subgroup chains. The authors hope that this will someday be extended to use arbitrary subgroup ladders.

Chapter 2

subgroupladders

Solutions of some problems in group theory can relatively easy be transferred to a sub- or supergroup if the index is small. Let G be a permutation group on the set $\{1,...,n\}$. So one might try to find a series of subgroups $G = H_0, ..., H_k = S_n$ of the symmetric group S_n such that H_{i-1} is a subgroup of H_i for every i and transfer the solution of a problem for the symmetric group step by step to G.

Sometimes it is not possible to find such a series with small indices between consecutive subgroups. This is where subgroup ladders may make sense: A subgroup ladder is series of subgroups $G = H_0, ..., H_k = S_n$ of the symmetric group such that for every $1 \leq i \leq k$, H_i is a subgroup of H_{i-1} or H_{i-1} is a subgroup of H_i. So we sometimes go up to a larger group in order to keep the indices small. A subgroup ladder may look like this:



If G is a Young subgroup of S_n, the algorithm in this repository can find a subgroup ladder of G such that the indices are at most the degree of the permutation group. This algorithm was described by Bernd Schmalz in [Sch90, Theorem 3.1.1]

2.1 Constructing Subgroupladders

The construction of a subgroupladder is implemented in several stages. The internal functions for each of these stages are exposed by this package. The main function of the subgroupladders package SubgroupLadder chains them together in a suitable way:

2.1.1 SubgroupLadder

▷ `SubgroupLadder(G[, refine][, n])` (function)

Returns: A subgroup ladder from `G` to the symmetric group on the moved points or on $[1..n]$,

when *n* is passed. The output is a list of records with a `Group` and a `LastDirection` field. The `LastDirection` entry is set to 1, if the last step in the ladder was an up-step, to -1, if the last step was a down-step and to 0 for the first entry.

Given a permutation group *G*, this will compute a subgroup ladder from *G* up to the symmetric group on the set of moved points of *G* If the optional third argument *n*is given, the ladder will be constructed up S_n. The optional second argument determines whether this function uses `AscendingChain` calls to find additional intermediate subgroup when the index may be large.

This functions embeds *G* first into the direct product of the induced permutation groups on the orbits, possible refined using `AscendingChain`. Then ladders for each of the direct factors are constructed and put together yielding a letter up to the Young subgroup corresponding to the orbits. This then is embedded into the parent symmetric group specified above using Schmalz's ladder, see `SubgroupLadderForYoungSubgroup`. If the transitive constituents are primitive, they will be embedded into the symmetric group on the orbit directly or using `AscendingChain`, depending on wether the `refine` option was used. For more details on the ladder constructed for imprimitive transitive constituents, see the documentation of `SubgroupLadderForImprimitive`.

Example

```
gap> G := Group([(1,2,3),(4,5,6)]);
Group([ (1,2,3), (4,5,6) ])
gap> SubgroupLadder(G);
[ rec( Group := Group([ (1,2,3), (4,5,6) ]), LastDirection := 0 ),
  rec( Group := Group([ (4,5,6), (4,5), (1,2,3) ]), LastDirection := 1 ),
  rec( Group := Group([ (4,5,6), (4,5), (1,2,3), (1,2) ]), LastDirection := 1 ),
  rec( Group := Group([ (1,2,3), (1,2), (4,5) ]), LastDirection := -1 ),
  rec( Group := Group([ (1,2,3,6), (1,2), (4,5) ]), LastDirection := 1 ),
  rec( Group := Group([ (1,2,3,6), (1,2) ]), LastDirection := -1 ),
  rec( Group := Group([ (1,2,3,5,6), (1,2) ]), LastDirection := 1),
  rec( Group := Group([ (1,2,3,4,5,6), (1,2) ]), LastDirection := 1 ) ]
gap> List(last, x->Order(x.Group));
[ 9, 18, 36, 12, 48, 24, 120, 720 ]
```

Note that the first embedding may produce a large index, even when the group is not maximal in the direct product of its transitive constituents:

Example

```
gap> G := Group([(1,2)(3,4)(5,6)(7,8)(9,10)(11,12)(13,14)(15,16)(17,18)]);
Group([ (1,2)(3,4)(5,6)(7,8)(9,10)(11,12)(13,14)(15,16)(17,18) ])
gap> L := SubgroupLadder(G);;
gap> Order(L[1].Group); Order(L[2].Group);
2
512
gap> L := SubgroupLadder(G, true);;
gap> Order(L[1].Group); Order(L[2].Group);
2
256
```

2.1.2 SubgroupLadderForYoungGroup

▷ `SubgroupLadderForYoungGroup(G[, n])` (function)

Returns: A subgroup ladder from *G* to the symmetric group on the moved points or on $[1..n]$, when *n* is passed. The output is a list of records with a `Group` and a `LastDirection` field. The

LastDirection entry is set to 1, if the last step in the ladder was an up-step, to -1, if the last step was a down-step and to 0 for the first entry.

Given a Young group G, this will compute a subgroup ladder from G up to the symmetric group of degree n. If n is given, it cannot be smaller than the largest moved point of G and the symmetric group is the canonical one acting on {1,...,n}. If the second argument is ommited, n will be the number of moved points of G and the symmetric group of degree n will act on the moved points of G. We can guarantee that all the indices are at most the degree n of the permutation group. Details on the ladder can be found in [Sch90, Satz 3.1.1].

2.1.3 SubgroupLadderForTransitive

▷ SubgroupLadderForTransitive(*G[, refine]*) (function)

Returns: A subgroup ladder from G to the symmetric group on the moved points The output is a list of records with a Group and a LastDirection field. The LastDirection entry is set to 1, if the last step in the ladder was an up-step, to -1, if the last step was a down-step and to 0 for the first entry.

Let G be a transitive permutaten group. This checks whether the group is primitive or imprimitive and constructs the ladder by directly embedding or SubgroupLadderForImprimitive respectively In the first case, the embedding will be refined with AscendingChain if the second argument is true.

2.1.4 SubgroupLadderForImprimitive

▷ SubgroupLadderForImprimitive(*G[, refine]*) (function)

Returns: A subgroup ladder from G to the symmetric group on the moved points The output is a list of records with a Group and a LastDirection field. The LastDirection entry is set to 1, if the last step in the ladder was an up-step, to -1, if the last step was a down-step and to 0 for the first entry.

Let G be an imprimitive permutation group. First this function embeds G into the smallest canonical wreath product W containg G. Then construct ladder from top group of W into the trivial group. Using this ladder, we can construct a ladder from the wreath product to its base group by iteratively replacing the top group with the groups in the other ladder. After that construct a ladder from base group to the parent symmetric group. The output is the concatenation of these ladders.

2.1.5 WreathProductSupergroupOfImprimitive

▷ WreathProductSupergroupOfImprimitive(*G*) (function)

Returns: the smallest nontrivial canonical wreath product containing G

Let G be an imprimitive permutation group. For every block system of G this constructs the smallest canonical wreath product containing G corresponding to this block system, then this returns the smallest one in total. For a block system $B = \{B_1, ..., B_k\}$, G induces a permutation group on B, denoted by G/B, and the canonical wreath product is

$$\mathrm{Stab}_G(B_1) \wr G/B \cong \Big(\mathrm{Stab}_G(B_1) \times \cdots \times \mathrm{Stab}_G(B_k)\Big) \rtimes G/B.$$

2.1.6 YoungGroupFromPartition

▷ YoungGroupFromPartition(*part*) (function)

Returns: the young subgroup $Sym(p_1)x...xSym(p_k)$.

Given a partial partition $part = (p_1, \ldots, p_k)$, this will compute the Young subgroup corresponding to this partition. Every p_i is a list of positive integers such that the union of the p_i is disjoint. The Young subgroup equals the internal direct product of the symmetric groups on the p_i

2.1.7 YoungGroupFromPartitionNC

▷ `YoungGroupFromPartitionNC(part)` (function)

Returns: a group

Like the above, but does not tests whether the argument is a list of disjoint lists.

2.1.8 DirectProductPermGroupsWithoutRenaming

▷ `DirectProductPermGroupsWithoutRenaming(list)` (function)

Returns: the direct product P.

Constructs a direct product of a list *list* of permutation groups with pairwise disjoint moved points such that all embeddings are canonical.

2.1.9 DirectProductPermGroupsWithoutRenamingNC

▷ `DirectProductPermGroupsWithoutRenamingNC(list)` (function)

Returns: the direct product P.

Like the above, but does not tests whether the argument is a list of permutation groups with pairwise disjoint moved points.

2.1.10 WreathProductWithoutRenaming

▷ `WreathProductWithoutRenaming(basefactor, topgroup, perms)` (function)

Construct a wreath product, such that the basegroup is the direct product of the conjugate groups of basefactors by using perms The top group acts on the basegroup by permuting the factors and conjugating with the corresponding perms

2.2 Computing the Table of Marks

The table of marks is a concept introduced by Burnside in 1909 in [Bur09] with many applications in Combinatorics.

Let G be a finite group. Let $U_1, \ldots, U_n$ be representatives of the conjugacy classes of subgroups of G. Then the table of marks of G is the $n \times n$ square matrix M where $M_{i,j}$ is the number of fixed points of U_i on the right cosets of U_j in G.

The complete table of marks can be computed from the subgroup lattice using GAPs builtin `TableOfMarks` function. However, one is often interested in small portions of the table of marks, so we wish to construct an algorithm that efficiently computes sections of the table of marks. As a right coset $U_j x$ is a fixed point in the action of U_j if and only if the double coset $U_j x U_i$ is actually a right coset, one can compute entries of the table of marks using double cosets. In [Sch90], Schmalz computes double coset representatives using subgroup ladders. However, the resulting algorithm to compute the table of marks is not very efficient.

The following method computes entries the table of marks directly just using subgroup chains.

2.2.1 TableOfMarksPartial

▷ `TableOfMarksPartial(`*`list, G[, sikpConjTest]`*`)` (function)

Returns: the desired section of the table of marks as a matrix.

Given a group `G`, this computes the partial table of marks induced by the subgroups of `G` in *`list`*. It will test the supplied subgroups for conjugacy. Passing `false` as the optional third argument *`skipConjTest`* will skip this test.

2.2.2 TableOfMarksEntryWithChain

▷ `TableOfMarksEntryWithChain(`*`G, chain, U`*`)` (function)

Returns: integer

Internal function called by TableOfMarksPartial, which computes recursively one entry of the table of marks. Let G be the parent group of the table of marks, i.e. $U,V \leq G$, where chain is an ascending subgroup chain of the form $V \leq \ldots \leq B \leq A \leq \ldots \leq G$. By iteration we compute the fixed points of $R[G:V]$ with respect to the action by right multiplication of U.

Chapter 3

License

References

[Bur09] W. Burnside. On the Theory of Groups of Finite Order. *Proc. London Math. Soc. (2)*, 7:1–7, 1909. 7

[Sch90] Bernd Schmalz. Verwendung von Untergruppenleitern zur Bestimmung von Doppelnebenklassen. *Bayreuth. Math. Schr.*, (31):109–143, 1990. 4, 6, 7

Index