permut

A package to deal with permutability in finite groups

2.0.5

11 January 2024

Adolfo Ballester-Bolinches

Enric Cosme-Llópez

Ramón Esteban-Romero

Adolfo Ballester-Bolinches
Email: Adolfo.Ballester@uv.es
Address: Departament de Matemàtiques
Universitat de València
Dr. Moliner, 50
46100 Burjassot, València, Spain

Enric Cosme-Llópez
Email: Enric.Cosme@uv.es
Homepage: https://www.uv.es/coslloen
Address: Departament de Matemàtiques
Universitat de València
Dr. Moliner, 50
46100 Burjassot, València, Spain

Ramón Esteban-Romero
Email: Ramon.Esteban@uv.es
Homepage: https://www.uv.es/estebanr

Contents

Chapter 1

Introduction to the PERMUT Package

All functions defined in this package deal only with finite groups. Moreover, some of the functions assume that the orders of all subgroups are easily computable and that the decomposition of the order of a group as a product of prime numbers can be done in a reasonable time.

The package PERMUT contains some functions to deal with permutability in finite groups. It includes functions to test some subgroup embedding properties related to permutability, like permutability or Sylow permutability. It also includes some functions to check whether a group belongs to the classes of T-groups, PT-groups, and PST-groups, which are the classes of groups in which normality, permutability, and Sylow permutability, respectively, are transitive. These properties and classes of groups have been widely studied during the last years. Most of them are described in [BBERA10].

The algorithms for T-groups, PT-groups, and PST-groups of this package use some interesting local descriptions of groups in these classes, that is, given in terms of some information related to the primes p dividing their order, usually by looking at p-subgroups or p-chief factors. These characterisations show that the only difference between all three classes of groups in the soluble universe corresponds to the Sylow structure. Nevertheless, for the sake of completeness, we also provide functions that use directly the definition of these classes. In the case of T-groups and PST-groups, as well as for soluble PT-groups, we reduce the test to subnormal subgroups of defect 2 (see [BBERR07], [BBERR09], and [BBBC$^+$09]). Of course, to do this we must introduce some functions to check whether two subgroups permute and whether a subgroup is permutable or S-permutable.

Some of the definitions of group-related concepts appear more than once in this manual, in the description of different functions. Although these repetitions may seem unnecessary when reading the whole manual, we hope that they benefit users who read the online help in GAP.

In order to obtain easily counterexamples which show that a group or a subgroup does not satisfy a certain property, we have introduced what we have called "One" functions, which store such counterexamples. In some cases, the property can be checked by proving that these counterexamples do not exist.

This package requires the Format package by B. Eick and C. R. B. Wright (see [EW03]), because it uses the functions `PResidual` (**FORMAT: PResidual**) and `SystemNormalizer` (**FORMAT: SystemNormalizer**), which are defined there. Some of the examples in this manual use the library of groups of small order.

The mathematical foundations of the algorithms presented in this package have been described in [BBCLER13].

The authors acknowledge the support of the grants MTM2010-19938-C03-01 and MTM2014-54707-C3-1-P funded by the *Ministerio de Economía y Competitividad*, Spanish Government

(all authors), PGC2018-095140-B-I00, funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe" (all authors), the grant PROMETEO/2017/057 funded by GVA/10.13039/501100003359 (A. Ballester-Bolinches and R. Esteban-Romero), the grant 11271085 from National Natural Science Foundation of China (A. Ballester-Bolinches) and the predoctoral grant AP2010-2764 from the *Ministerio de Educación*, Spanish Government (E. Cosme-Llópez). The authors are also indebted to the members of the GAP council, especially Leonard Soicher, Alice Niemeyer, Max Horn, and Alexander Konovalov, as well as to the anonymous referees, for their comments which have helped us to improve the package and its documentation.

Chapter 2

Installation and Help of the PERMUT Package

Since all functions of this package are written in the GAP language, it is enough to unpack the archive file in a directory in the pkg hierarchy of your GAP 4 distribution. It has been tested on version 4.7.4 of GAP.

The PERMUT package requires that the package Format [EW03] be installed on the system, because it uses the functions `PResidual` (**FORMAT: PResidual**) and `SystemNormalizer` (**FORMAT: SystemNormalizer**) which are defined there. The Format package can be downloaded from the GAP web page (`https://www.gap-system.org`).

The PERMUT package can be loaded with

Example

```
gap> LoadPackage("permut");
-------------------------------------------------------------------------------
Loading  FORMAT 1.4.3 (Formations of Finite Soluble Groups)
by Bettina Eick (http://www.iaa.tu-bs.de/beick) and
   Charles R.B. Wright (https://pages.uoregon.edu/wright/).
maintained by:
   Bettina Eick (http://www.iaa.tu-bs.de/beick) and
   The GAP Team (support@gap-system.org).
Homepage: https://gap-packages.github.io/format/
Report issues at https://github.com/gap-packages/format/issues
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Loading  permut 2.0.4 (PERMUT: A package to deal with permutability in finite groups)
by Adolfo Ballester-Bolinches (Adolfo.Ballester@uv.es),
   Enric Cosme-Llópez (https://www.uv.es/coslloen), and
   Ramón Esteban-Romero (https://www.uv.es/estebanr).
maintained by:
   Ramón Esteban-Romero (https://www.uv.es/estebanr).
Homepage: https://gap-packages.github.io/permut/
Report issues at https://github.com/gap-packages/permut/issues
-------------------------------------------------------------------------------
true
```

Suggestions, comments, and bug reports can be sent to the email address `Ramon.Esteban@uv.es`.

Chapter 3

Permutability of Subgroups in Finite Groups

This chapter describes functions to check permutability of subgroups in a given group. First we present a function to check whether a subgroup permutes with another one, then we present functions to test whether a subgroup permutes with the members of a given family of subgroups, and finally we introduce some other subgroup embedding properties related to permutability.

3.1 Permutability functions

3.1.1 ArePermutableSubgroups

▷ `ArePermutableSubgroups([G, ]U, V)` (function)

This function returns `true` if `U` and `V` permute in `G`. The groups `U` and `V` must be subgroups of `G`. The subgroups U and V *permute* when $UV = VU$. This is equivalent to affirming that UV is a subgroup of G.

This is done by checking that the order of $\langle U,V\rangle$ is the order of their Frobenius product UV, that is, $|U||V|/|U\cap V|$. Hence the performance of this function depends strongly on the existence of good algorithms to compute the intersection of two subgroups and, of course, the order of a subgroup. Shorthands are provided for the cases in which one of `U` and `V` is a subgroup of the other one or `U` or `V` are permutable in a common supergroup.

In the version with two arguments, `U` and `V` must have a common parent or `ClosureGroup( U, V )` (see `ClosureGroup` (**Reference: ClosureGroup**)) is called to construct a common supergroup for `U` and `V`.

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> a:=Subgroup(g,[(1,2)(3,4)]);
Group([ (1,2)(3,4) ])
gap> b:=Subgroup(g,[(1,2,3)]);
Group([ (1,2,3) ])
gap> c:=Subgroup(g,[(1,2)]);
Group([ (1,2) ])
gap> ArePermutableSubgroups(g,a,b);
false
```

```
gap> ArePermutableSubgroups(g,a,c);
true
gap> ArePermutableSubgroups(g,b,c);
true
gap> ArePermutableSubgroups(b,c);
true
gap> ArePermutableSubgroups(b,a);
false
```

3.2 Embedding properties related to permutability

In the following we describe some functions which allow us to test whether a subgroup permutes with the members of some families of subgroups. We pay special attention to the families of all subgroups and all Sylow subgroups of the group. In some cases, we have introduced some "One" functions, which give an element or a subgroup in the relevant family of subgroups of the group which shows that the given property fails, or `fail` otherwise.

3.2.1 PermutMaxTries

▷ `PermutMaxTries` (global variable)

This variable contains the maximum number of random attempts of permutability checks before trying general deterministic methods in the functions `IsPermutable` (3.2.2) and `IsIwasawaSylow` (5.3.5). Its default value is 10.

3.2.2 IsPermutable

▷ `IsPermutable(`*G*`, `*H*`)` (operation)
▷ `IsPermutableInParent(`*H*`)` (property)

This property returns `true` if the subgroup *H* is permutable in *G*, otherwise it returns `false`. We say that a subgroup H of a group G is *permutable* in G if H permutes with all subgroups of G.

If the attribute `OneSubgroupNotPermutingWithInParent` (3.2.3) has been set, it is used if possible. Otherwise, the algorithm checks looks for a cyclic subgroup not permuting with *H*. The number of such cyclic subgroups is controlled by the variable `PermutMaxTries` (3.2.1), by default, 10. If *H* permutes with all these subgroups, then the algorithm checks whether *H* is hypercentrally embedded in *G* and that the Sylow p-subgroups of H/H_G permute with all cyclic p-subgroups of G/H_G for each prime p dividing the order of G/H_G. This is a sufficient condition for permutability.

3.2.3 OneSubgroupNotPermutingWith

▷ `OneSubgroupNotPermutingWith(`*G*`, `*H*`)` (function)
▷ `OneSubgroupNotPermutingWithInParent(`*H*`)` (attribute)

This attribute finds a cyclic subgroup of *G* which does not permute with *H*, that is, a subgroup which shows that *H* is not permutable in *G*. Recall that a subgroup H of a group G is *permutable* in G if H permutes with all subgroups of G.

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> a:=Subgroup(g,[(1,2)(3,4)]);
Group([ (1,2)(3,4) ])
gap> b:=Subgroup(g,[(1,2,3)]);
Group([ (1,2,3) ])
gap> c:=Subgroup(g,[(1,2)]);
Group([ (1,2) ])
gap> IsPermutable(g,a);
false
gap> IsPermutable(g,b);
false
gap> IsPermutable(g,c);
false
gap> OneSubgroupNotPermutingWith(g,b);
Group([ (1,3,4) ])
gap> v:=Subgroup(g,[(1,2)(3,4),(1,3)(2,4)]);
Group([ (1,2)(3,4), (1,3)(2,4) ])
gap> OneSubgroupNotPermutingWith(g,v);
fail
gap> IsPermutable(g,b);
false
gap> IsPermutable(g,v);
true
```

Example

```
gap> g:=SmallGroup(16,6);
<pc group of size 16 with 4 generators>
gap> h:=Subgroup(g,[g.2]);
Group([ f2 ])
gap> IsNormal(g,h);
false
gap> IsPermutable(g,h);
true
```

Sometimes one does not require a subgroup to permute with all subgroups of the group, but only with a selected family of subgroups of the group. The general case is the following.

3.2.4 IsFPermutable

▷ `IsFPermutable(`*G*`, `*H*`, `*f*`)` (function)

In this function, *H* is a subgroup of *G* and *f* must be a list of subgroups of *G*. It returns `true` if *H* permutes with all members of *f* and `false` otherwise.

This function uses the function `OneFSubgroupNotPermutingWith` (3.2.5). Hence it tries to use the values of `IsPermutableInParent` (3.2.2) and `OneSubgroupNotPermutingWithInParent` (3.2.3) if one of them is set, and if it returns `false` it tries to set the values of `IsPermutableInParent` (3.2.2) and `OneSubgroupNotPermutingWithInParent` (3.2.3).

3.2.5 OneFSubgroupNotPermutingWith

▷ `OneFSubgroupNotPermutingWith(G, H, f)` (operation)

In this operation, H is a subgroup of G and f must be a list of subgroups of G. It returns a subgroup in f not permuting with H if such a subgroup exists, and `fail` otherwise.

This function tries to use the values of `IsPermutableInParent` (3.2.2) and `OneSubgroupNotPermutingWithInParent` (3.2.3) if one of them is set. If it returns `fail`, then it tries to set the value of `IsPermutableInParent` (3.2.2) and `OneSubgroupNotPermutingWithInParent` (3.2.3).

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> a:=Subgroup(g,[(1,2,3,4),(1,3)]);
Group([ (1,2,3,4), (1,3) ])
gap> Size(a);
8
gap> OneFSubgroupNotPermutingWith(g,a,MaximalSubgroups(g));
Group([ (1,2), (3,4), (1,3)(2,4) ])
gap> IsFPermutable(g,a,MaximalSubgroups(g));
false
gap> HasIsPermutableInParent(a);
true
gap> IsPermutableInParent(a);
false
gap> HasOneSubgroupNotPermutingWithInParent(a);
true
gap> OneSubgroupNotPermutingWithInParent(a);
Group([ (1,2), (3,4), (1,3)(2,4) ])
gap> IsFPermutable(g,a,AllSubnormalSubgroups(g));
true
gap> OneFSubgroupNotPermutingWith(g,a,AllSubnormalSubgroups(g));
fail
gap> sylows:=g->Union(List(SylowSubgroups(g),
>           t->ConjugacyClassSubgroups(g,t)));
function( g ) ... end
gap> OneFSubgroupNotPermutingWith(g,a,sylows(g));
Group([ (3,4), (1,4)(2,3), (1,3)(2,4) ])
```

The following functions can be considered as particular cases of the previous function for some subgroup embedding functors. However, they can be stored as "in parent" attributes or properties and in some cases we have tried to give more efficient code.

3.2.6 IsSPermutable

▷ `IsSPermutable(G, H)` (operation)

▷ `IsSPermutableInParent(H)` (property)

This operation returns `true` if a subgroup H of G is S-permutable in G, that is, H permutes with all Sylow subgroups of G, and returns `false` otherwise.

Example

```
gap> g:=SmallGroup(8,3);
<pc group of size 8 with 3 generators>
gap> IsSPermutable(g,Subgroup(g,[g.1]));
true
gap> IsPermutable(g,Subgroup(g,[g.1]));
false
```

3.2.7 OneSylowSubgroupNotPermutingWith

▷ OneSylowSubgroupNotPermutingWith(*G*, *H*) (operation)
▷ OneSylowSubgroupNotPermutingWithInParent(*H*) (attribute)

The argument *H* must be a subgroup of *G*. If *H* is S-permutable in *G*, then it returns fail. Otherwise, it returns a Sylow subgroup of *G* which does not permute with *H*. We say that a subgroup *H* of a group *G* is S-permutable in *G* if *H* permutes with all Sylow subgroups of *G*.

Example

```
gap> g:=SymmetricGroup(4);;
gap> a:=Subgroup(g,[(1,2)(3,4)]);;
gap> OneSylowSubgroupNotPermutingWith(g,a);
Group([ (2,4,3) ])
```

3.2.8 IsSNPermutable

▷ IsSNPermutable(*G*, *H*) (operation)
▷ IsSNPermutableInParent(*H*) (attribute)

This operation returns true if *H* permutes with all system normalisers of *G*, and false otherwise. Here *G* must be a soluble group and *H* must be a subgroup of *G*. If the function is applied to an insoluble group, it gives an error.

3.2.9 OneSystemNormaliserNotPermutingWith

▷ OneSystemNormaliserNotPermutingWith(*G*, *H*) (operation)
▷ OneSystemNormaliserNotPermutingWithInParent(*H*) (attribute)

Here *G* must be a soluble group and *H* must be a subgroup of *G*. If *H* permutes with all system normalisers of *G*, then this operation returns fail. Otherwise, it returns a system normaliser *D* of *G* such that *H* does not permute with *D*. If the group *G* is not soluble, then it gives an error.

Example

```
gap> g:=Group((1,2,3),(4,5,6),(1,2));
Group([ (1,2,3), (4,5,6), (1,2) ])
gap> a:=Subgroup(g,[(1,2,3)(4,5,6)]);
Group([ (1,2,3)(4,5,6) ])
gap> IsSNPermutable(g,a);
true
gap> IsSPermutable(g,a);
false
```

The next functions are not particular cases of `IsFPermutable` (3.2.4) or `OneFSubgroupNotPermutingWith` (3.2.5), but we include them in the package because every subgroup permuting with all its conjugates is subnormal (see [Fog97]).

3.2.10 IsConjugatePermutable

▷ `IsConjugatePermutable(`*G*`,` *H*`)` (operation)
▷ `IsConjugatePermutableInParent(`*H*`)` (property)

This operation takes the value `true` if *H* permutes with all its conjugates, and the value `false` otherwise.

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> a:=Subgroup(g,[(1,2)(3,4)]);
Group([ (1,2)(3,4) ])
gap> IsPermutable(g,a);
false
gap> IsConjugatePermutable(g,a);
true
```

3.2.11 OneConjugateSubgroupNotPermutingWith

▷ `OneConjugateSubgroupNotPermutingWith(`*G*`,` *H*`)` (operation)
▷ `OneConjugateSubgroupNotPermutingWithInParent(`*H*`)` (attribute)

This operation finds a conjugate subgroup of *H* which does not permute with *H* if such a subgroup exists. If *H* permutes with all its conjugates, then this operation returns `fail`.

Example

```
gap> g:=SmallGroup(16,7);
<pc group of size 16 with 4 generators>
gap> h:=Subgroup(g,[g.1*g.4]);
Group([ f1*f4 ])
gap> IsConjugatePermutable(g,h);
false
gap> OneConjugateSubgroupNotPermutingWith(g,h);
Group([ f1*f3 ])
```

Next we introduce some subgroup embedding functions related to permutability which have proved to be useful in some characterisations of soluble T-groups, PT-groups, and PST-groups. The "One" functions return a value which proves that the corresponding subgroup embedding property is false.

3.2.12 IsWeaklySPermutable

▷ `IsWeaklySPermutable(`*G*`,` *H*`)` (operation)
▷ `IsWeaklySPermutableInParent(`*H*`)` (property)

The value returned by this operation is `true` when *H* is a *weakly S-permutable* subgroup of *G*, that is, *H* is S-permutable in $\langle H, H^g \rangle$ implies that *H* is S-permutable in $\langle H, g \rangle$, and `false` otherwise.

3.2.13 OneElementShowingNotWeaklySPermutable

▷ `OneElementShowingNotWeaklySPermutable(G, H)` (operation)
▷ `OneElementShowingNotWeaklySPermutableInParent(H)` (attribute)

If `H` is a weakly S-permutable subgroup of `G`, then this operation returns `fail`. Otherwise, the value returned by this operation is an element $g \in G$ such that `H` is S-permutable in $\langle H, H^g \rangle$, but H is not S-permutable in $\langle H, g \rangle$. A subgroup H of a group G is said to be weakly S-permutable if H is S-permutable in $\langle H, H^g \rangle$ implies that H is S-permutable in $\langle H, g \rangle$.

3.2.14 IsWeaklyPermutable

▷ `IsWeaklyPermutable(G, H)` (operation)
▷ `IsWeaklyPermutableInParent(H)` (property)

This operation returns `true` if `H` is weakly permutable in `G`, and `false` otherwise. A subgroup H of G is *weakly permutable* if the fact that H is S-permutable in $\langle H, H^g \rangle$, implies that H is S-permutable in $\langle H, g \rangle$.

3.2.15 OneElementShowingNotWeaklyPermutable

▷ `OneElementShowingNotWeaklyPermutable(G, H)` (operation)
▷ `OneElementShowingNotWeaklyPermutableInParent(H)` (attribute)

If `H` is a weakly permutable subgroup of `G`, then this operation returns `fail`. Otherwise, the value returned by this operation is an element $g \in G$ such that `H` is permutable in $\langle H, H^g \rangle$, but H is not permutable in $\langle H, g \rangle$. A subgroup H of a group G is said to be *weakly permutable* if the fact that H is permutable in $\langle H, H^g \rangle$ implies that H is permutable in $\langle H, g \rangle$.

3.2.16 IsWeaklyNormal

▷ `IsWeaklyNormal(G, H)` (operation)
▷ `IsWeaklyNormalInParent(H)` (property)

This operation returns `true` if `H` is weakly normal in `G`, and `false` otherwise. A subgroup H of G is *weakly normal* whenever if $H^g \leq \mathrm{N}_G(H)$, then $g \in \mathrm{N}_G(H)$.

3.2.17 OneElementShowingNotWeaklyNormal

▷ `OneElementShowingNotWeaklyNormal(G, H)` (operation)
▷ `OneElementShowingNotWeaklyNormalInParent(H)` (attribute)

If `H` is a weakly normal subgroup of `G`, then this function returns `fail`. Otherwise, the value returned by this operation is an element g such that $H^g \leq \mathrm{N}_G(H)$ is a subgroup of $\mathrm{N}_G(H)$ but $g \notin \mathrm{N}_G(H)$.

Example

```
gap> g:=DihedralGroup(8);
<pc group of size 8 with 3 generators>
```

```
gap> a:=Subgroup(g,[g.1]);
Group([ f1 ])
gap> IsWeaklySPermutable(g,a);
true
gap> IsWeaklyPermutable(g,a);
false
gap> x:=OneElementShowingNotWeaklyPermutable(g,a);
f2
gap> IsSubgroup(Normalizer(g,a),ConjugateSubgroup(a,x));
true
gap> x in Normalizer(g,a);
false
```

3.2.18 IsWithSubnormalizerCondition

▷ IsWithSubnormalizerCondition(*G*, *H*) (operation)
▷ IsWithSubnormalizerConditionInParent(*H*) (property)
▷ IsWithSubnormaliserCondition(*G*, *H*) (operation)
▷ IsWithSubnormaliserConditionInParent(*H*) (property)

This operation returns `true` if the subgroup H satisfies the subnormaliser condition in G, and `false` otherwise.

A subgroup H is said to *satisfy the subnormaliser condition* in G if the condition that H is subnormal in a subgroup K of G implies that H is normal in K.

3.2.19 OneSubgroupInWhichSubnormalNotNormal

▷ OneSubgroupInWhichSubnormalNotNormal(*G*, *H*) (operation)
▷ OneSubgroupInWhichSubnormalNotNormalInParent(*H*) (attribute)

This function returns a subgroup K of G such that H is subnormal in K and H is not normal in K, if this subgroup exists; otherwise, it returns `fail`.

3.2.20 IsWithSubpermutizerCondition

▷ IsWithSubpermutizerCondition(*G*, *H*) (operation)
▷ IsWithSubpermutizerConditionInParent(*H*) (property)
▷ IsWithSubpermutiserCondition(*G*, *H*) (operation)
▷ IsWithSubpermutiserConditionInParent(*H*) (property)

This operation returns `true` if the subgroup H satisfies the subpermutiser condition in G, and `false` otherwise.

A subgroup H is said to *satisfy the subpermutiser condition* in G if the condition that H is subnormal in a subgroup K of G implies that H is permutable in K.

3.2.21 OneSubgroupInWhichSubnormalNotPermutable

▷ `OneSubgroupInWhichSubnormalNotPermutable(`*G*`, `*H*`)` (operation)
▷ `OneSubgroupInWhichSubnormalNotPermutableInParent(`*H*`)` (attribute)

This function returns a subgroup K of G such that H is subnormal in K and H is not permutable in K if this subgroup exists; otherwise it returns `fail`.

3.2.22 IsWithSSubpermutizerCondition

▷ `IsWithSSubpermutizerCondition(`*G*`, `*H*`)` (operation)
▷ `IsWithSSubpermutizerConditionInParent(`*H*`)` (property)
▷ `IsWithSSubpermutiserCondition(`*G*`, `*H*`)` (operation)
▷ `IsWithSSubpermutiserConditionInParent(`*H*`)` (property)

This operation returns `true` if the subgroup H satisfies the S-subpermutiser condition in G, and `false` otherwise.

A subgroup H is said to *satisfy the S-subpermutiser condition* in G if the condition that H is subnormal in a subgroup K of G implies that H is S-permutable in K.

3.2.23 OneSubgroupInWhichSubnormalNotSPermutable

▷ `OneSubgroupInWhichSubnormalNotSPermutable(`*G*`, `*H*`)` (operation)
▷ `OneSubgroupInWhichSubnormalNotSPermutableInParent(`*H*`)` (attribute)

This function returns a subgroup K of G such that H is subnormal in K and H is not S-permutable in K if such a subgroup exists; otherwise it returns `fail`.

Example

```
gap> g:=SmallGroup(324,160);
<pc group of size 324 with 6 generators>
gap> a:=Subgroup(g,[g.3,g.5]);
Group([ f3, f5 ])
gap> IsWithSubnormalizerCondition(g,a);
true
gap> IsWeaklyNormal(g,a);
false
gap> IsWeaklySPermutable(g,a);
false
gap> x:=OneElementShowingNotWeaklyNormal(g,a);
f1
gap> ConjugateSubgroup(a,x)=a;
false
gap> IsSubset(Normalizer(g,a),ConjugateSubgroup(a,x));
true
```

Chapter 4

T-groups, PT-groups, and PST-groups

This chapter explains the functions to check whether a given group is a T-group, a PT-group, or a PST-group.

Recall that a group G is:

a T-group
when every subnormal subgroup of G is normal,

a PT-group
when every subnormal subgroup of G is permutable,

a PST-group
when every subnormal subgroup of G is S-permutable.

We also present functions to identify groups in other classes related to these ones.

The "One" functions are defined to provide examples of subgroups or elements showing that a group theoretical property for a group or for a subgroup is false.

4.1 "One" functions

4.1.1 OneSubnormalNonNormalSubgroup

▷ `OneSubnormalNonNormalSubgroup(G)` (attribute)

`OneSubnormalNonNormalSubgroup` returns a subnormal subgroup of defect 2 which is not normal in the group `G`, if such a subgroup exists. If such a subgroup does not exist because the group is a T-group, it returns `fail`.

A T-group is a group in which normality is transitive, that is, if H is a normal subgroup of K and K is a normal subgroup of G, then H is a normal subgroup of G. Finite T-groups are the groups in which every subnormal subgroup is normal.

This function tries to set the property `IsTGroup` (4.2.1) to `true` or `false` according to its result.

Example

```
gap> g:=SmallGroup(320,152);
<pc group of size 320 with 7 generators>
gap> x:=OneSubnormalNonNormalSubgroup(g);
Group([ f2, f3, f5, f7 ])
```

```
gap> IsNormal(g,x);
false
gap> IsSubnormal(g,x);
true
```

4.1.2 OneSubnormalNonPermutableSubgroup

▷ OneSubnormalNonPermutableSubgroup(*G*) (attribute)

OneSubnormalNonPermutableSubgroup returns a subnormal subgroup which is not permutable in the group *G*, if such a subgroup exists. If such a subgroup does not exist because the group is a PT-group, it returns fail.

A group G is a PT-group when permutability is a transitive relation in G, that is, if H is a permutable subgroup of K and K is a permutable subgroup of G, then H is a permutable subgroup of G. This is equivalent in finite groups to affirming that every subnormal subgroup of G is permutable.

This function tries to set the property IsPTGroup (4.2.2) to true or false according to its result.

Since this function checks all subnormal subgroups for permutability, it may take a long time if there are many subnormal subgroups.

Example

```
gap> g:=SmallGroup(320,152);
<pc group of size 320 with 7 generators>
gap> OneSubnormalNonPermutableSubgroup(g);
fail
gap> IsPTGroup(g);
true
gap> g:=SmallGroup(8,3);
<pc group of size 8 with 3 generators>
gap> OneSubnormalNonPermutableSubgroup(g);
Group([ f1*f3 ])
```

4.1.3 OneSubnormalNonSPermutableSubgroup

▷ OneSubnormalNonSPermutableSubgroup(*G*) (attribute)

OneSubnormalNonSPermutableSubgroup returns a subnormal subgroup of defect 2 which is not S-permutable in *G*, if such a subgroup exists. If such a subgroup does not exist because the group is a PST-group, it returns fail.

A group G is a PST-group when S-permutability (Sylow permutability) is a transitive relation in G, that is, if H is an S-permutable subgroup of K and K is an S-permutable subgroup of G, then H is an S-permutable subgroup of G. This is equivalent in finite groups to affirming that every subnormal subgroup of G is S-permutable. By a result of Ballester-Bolinches, Esteban-Romero, and Ragland [BBERR07], it is enough to check this last condition for all subnormal subgroups of defect 2.

This function tries to set the property IsPSTGroup (4.2.3) to true or false according to its result.

Example

```
gap> g:=AlternatingGroup(4);
Alt( [ 1 .. 4 ] )
gap> OneSubnormalNonSPermutableSubgroup(g);
Group([ (1,2)(3,4) ])
```

4.1.4 OneSubnormalNonConjugatePermutableSubgroup

▷ `OneSubnormalNonConjugatePermutableSubgroup(`*G*`)` (attribute)

This function finds a subnormal subgroup H which does not permute with all its conjugates, if such a subgroup exist; otherwise, it returns `fail`.

Example

```
gap> g:=AlternatingGroup(4);
Alt( [ 1 .. 4 ] )
gap> OneSubnormalNonConjugatePermutableSubgroup(g);
fail
gap> g:=DihedralGroup(16);
<pc group of size 16 with 4 generators>
gap> OneSubnormalNonConjugatePermutableSubgroup(g);
Group([ f1*f4 ])
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> OneSubnormalNonConjugatePermutableSubgroup(g);
fail
gap> OneSubnormalNonPermutableSubgroup(g);
Group([ (1,2)(3,4) ])
```

4.1.5 OneSubnormalNonSNPermutableSubgroup

▷ `OneSubnormalNonSNPermutableSubgroup(`*G*`)` (attribute)

This attribute returns a subnormal subgroup H of the soluble group G such that H does not permute with a system normaliser if such a subgroup exists; otherwise, it returns `fail`. This system normaliser is obtained with the function `SystemNormalizer` (**FORMAT: SystemNormalizer**) of the Format package.

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> OneSubnormalNonSNPermutableSubgroup(g);
Group([ (1,3)(2,4) ])
gap> g:=Group((1,2,3)(4,5,6),(1,2));
Group([ (1,2,3)(4,5,6), (1,2) ])
gap> OneSubnormalNonSNPermutableSubgroup(g);
fail
gap> OneSubnormalNonSPermutableSubgroup(g);
Group([ (1,2,3)(4,6,5) ])
```

4.2 Group properties related to permutability

The next function names correspond to properties.

4.2.1 IsTGroup

▷ `IsTGroup(`*G*`)` (property)

This function returns `true` if `G` is a T-group, and `false` otherwise.

T-groups are the groups in which normality is a transitive relation, that is, if H is a subgroup of K and K is a subgroup of G, then H is a subgroup of G. In the finite case, they are the groups in which every subnormal subgroup is normal.

For soluble groups, the algorithm checks that for every prime p dividing its order, G is p-nilpotent and has a Dedekind Sylow p-subgroup or G has an abelian Sylow p-subgroup P and every subgroup of P is normal in $N_G(P)$.

For insoluble groups, the function checks whether the group is an SC-group with the function `IsSCGroup` (7.1.4), because PT-groups are SC-groups. Since the methods for insoluble groups depend on the computation of a chief series with the function `ChiefSeries` (**Reference: ChiefSeries**), they might not be available if the group is not given as a permutation group. Then it is checked that every subnormal subgroup of defect 2 is normal with the help of the function `OneSubnormalNonNormalSubgroup` (4.1.1). The methods based on the ideas of [BBBH03a], [BBBH03b], and [BH03] have not been implemented so far because they require the computation of quotients by all normal subgroups, which could be a time-consuming task.

Example

```
gap> g:=SmallGroup(40,4);
<pc group of size 40 with 4 generators>
gap> IsTGroup(g);
true
gap> g:=SymmetricGroup(3);
Sym( [ 1 .. 3 ] )
gap> IsTGroup(g);
true
```

4.2.2 IsPTGroup

▷ `IsPTGroup(G)` (property)

This property takes the value `true` if `G` is a PT-group, and the value `false` otherwise.

For a soluble group G, the function checks whether for all primes p, G is p-nilpotent and has an Iwasawa Sylow p-subgroup or G has an abelian Sylow p-subgroup and it satisfies the property $\mathscr{C}_p$ (that is, every subgroup of a Sylow p-subgroup P of G is normal in the Sylow normaliser $N_G(P)$).

For insoluble groups, the function checks that the group is an SC-group with the function `IsSCGroup` (7.1.4), because PT-groups are SC-groups. Since the methods for insoluble groups depend on the computation of a chief series with the function `ChiefSeries` (**Reference: ChiefSeries**), they might not be available if the group is not given as a permutation group. Then it uses the function `OneSubnormalNonPermutableSubgroup` (4.1.2) to check whether or not every subnormal subgroup is permutable. The methods based on the ideas of [BBBH03a], [BBBH03b], and [BH03] have not been implemented so far because they require the computation of quotients by all normal subgroups, which could be a time-consuming task.

Example

```
gap> g:=SmallGroup(1323,37);
<pc group of size 1323 with 5 generators>
gap> IsPTGroup(g);
true
gap> IsTGroup(g);
false
```

```
gap> OneSubnormalNonNormalSubgroup(g);
Group([ f2*f3, f4, f5 ])
```

4.2.3 IsPSTGroup

▷ `IsPSTGroup(G)` (property)

This function returns true if the group `G` is a PST-group, and false otherwise.

A finite group G is a PST-group if S-permutability (Sylow-permutability) is a transitive relation in G, that is, if H is S-permutable in K and K is S-permutable in G, then H is S-permutable in G. This is equivalent to affirming that every subnormal subgroup of G is S-permutable in G.

For a soluble group G, the function checks whether for all primes p, G is p-nilpotent, or G has an abelian Sylow p-subgroup and G satisfies the property $\mathscr{C}_p$ (that is, every subgroup of a Sylow p-subgroup P of G is normal in the Sylow normaliser $\mathrm{N}_G(P)$)

For insoluble groups, the function checks whether the group is an SC-group with the function `IsSCGroup` (7.1.4), because PST-groups are SC-groups. Since the methods for insoluble groups depend on the computation of a chief series with the function `ChiefSeries` (**Reference: ChiefSeries**), they might not be available if the group is not given as a permutation group. Then it uses the function `OneSubnormalNonSPermutableSubgroup` (4.1.3) to check whether or not every subnormal subgroup of defect 2 is S-permutable. The methods based on the ideas of [BBBH03a], [BBBH03b], and [BH03] have not been implemented so far because they require the computation of quotients by all normal subgroups, which could be a time-consuming task.

Example

```
gap> g:=SmallGroup(24,6);
<pc group of size 24 with 4 generators>
gap> IsPSTGroup(g);
true
gap> IsPTGroup(g);
false
gap> OneSubnormalNonPermutableSubgroup(g);
Group([ f1*f3, f4 ])
gap> g:=SmallGroup(24,6);
<pc group of size 24 with 4 generators>
gap> IsPSTGroup(g);
true
gap> IsPTGroup(g);
false
gap> OneSubnormalNonPermutableSubgroup(g);
Group([ f1*f3, f4 ])
gap> OneSubgroupNotPermutingWith(g,last);
Group([ f1*f2 ])
```

4.2.4 IsCPTGroup

▷ `IsCPTGroup(G)` (property)

This property returns true if every subnormal subgroup of `G` permutes with all its conjugates, and false otherwise.

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> IsCPTGroup(g);
true
gap> IsPTGroup(g);
false
gap> IsPSTGroup(g);
false
```

4.2.5 IsPSNTGroup

▷ `IsPSNTGroup(`*G*`)` (property)

This property takes the value `true` if every subnormal subgroup of the soluble group *G* permutes with every system normaliser of *G*, and `false` otherwise. If the function is applied to an insoluble group, it gives an error.

Example

```
gap> g:=Group((1,2,3)(4,5,6),(1,3));
Group([ (1,2,3)(4,5,6), (1,3) ])
gap> IsPSTGroup(g);
false
gap> IsPSNTGroup(g);
true
gap> IsCPTGroup(g);
true
gap> g:=SmallGroup(16,7);
<pc group of size 16 with 4 generators>
gap> IsPSTGroup(g);
true
gap> IsCPTGroup(g);
false
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> IsPSNTGroup(g);
false
gap> IsCPTGroup(g);
true
```

Chapter 5

Local Functions in the PERMUT Package

In the study of permutability, the usage of local characterisations has become a useful tool to describe the classes of T-groups, PT-groups, and PST-groups. In this chapter we present some local characterisations of these classes and some functions which allow to check whether or not a group given in GAP satisfies these conditions.

A *local* description of group-theoretical property consists of expressing it as the conjunction of some properties depending on a prime p, usually related to the behaviour of p-elements, p-subgroups, or p-chief factors, for all primes p.

5.1 A Local Function for Supersolubility

The GAP library does not contain methods to check whether a group G is p-supersoluble, where p is a prime number. We include such a function in the PERMUT package.

5.1.1 IsPSupersolvable

▷ `IsPSupersolvable(G, p)` (function)
▷ `IsPSupersoluble(G, p)` (function)

This function returns `true` if the group `G` is p-supersoluble, and `false` otherwise, where `p` is a prime number. This function is not defined in GAP. The method we have implemented for finite groups includes checking whether the group is supersoluble (in this case, it must return `true`). If the group is not soluble, it computes a chief series and checks whether all chief factors have order p or have order not divisible by p.

Example

```
gap> g:=Group((1,2,3,4,5,6,7), (8,9,10,11,12,13,14), (15,16,17,18,19,20,21),
> (22,23,24,25,26,27,28), (29,30,31,32,33,34,35),
> (1,8,15,22,29)(2,9,16,23,30)(3,10,17,24,31)(4,11,18,25,32)(5,12,19,26,
>     33)(6,13,20,27,34)(7,14,21,28,35),
> (1,8)(2,9)(3,10)(4,11)(5,12)(6,13)(7,14)); #C7 wr S5
<permutation group with 7 generators>
gap> IsPSupersolvable(g,7);
false
gap> IsPSupersolvable(g,11);
true
```

Example

```
gap> g:=DirectProduct(PSL(2,7),
>     Group((1,2,3,4,5,6,7,8,9,10,11), (2,5,6,10,4)(3,9,11,8,7)));
Group([ (3,7,5)(4,8,6), (1,2,6)(3,4,8), (9,10,11,12,13,14,15,16,17,18,19),
  (10,13,14,18,12)(11,17,19,16,15) ])
gap> IsPNilpotent(g,5);
true
gap> IsPNilpotent(g,11);
false
gap> IsPSupersolvable(g,11);
true
gap> IsPNilpotent(g,3);
false
```

5.2 Local functions for T-groups, PT-groups, and PST-groups

The following functions correspond to local description of the classes of soluble T-groups, PT-groups, and PST-groups. Most of the known useful local characterisations of these classes of groups can be seen to be equivalent to one of them, either in the universe or all finite groups or in the universe of all finite p-soluble groups. By a local characterisation of a group-theoretical property $\mathscr{R}$ we mean a group-theoretical property $\mathscr{R}_p$ for each prime p such that a group satisfies $\mathscr{R}$ if and only if it satisfies $\mathscr{R}_p$ for all primes p.

5.2.1 IsCp

▷ `IsCp(G, p)` (function)

This function returns `true` if the group `G` satisfies the property $\mathscr{C}_p$, where p is a prime number, and `false` otherwise.

A group G satisfies $\mathscr{C}_p$ when every subgroup H of a Sylow p-subgroup P of G is normal in the corresponding Sylow normaliser $\mathrm{N}_G(P)$. This property was introduced by Robinson in [Rob68]. A group G is a soluble PST-group if and only if it satisfies $\mathscr{C}_p$ for all primes p.

Example

```
gap> g:=AlternatingGroup(5);
Alt( [ 1 .. 5 ] )
gap> IsCp(g,3);
true
gap> IsCp(g,5);
true
gap> IsCp(g,7);
true
gap> IsCp(g,2);
false
gap> g:=SmallGroup(200,44); # semidirect product of Q8 with C5xC5
<pc group of size 200 with 5 generators>
gap> IsCp(g,5);
false
gap> IsCp(g,2);
true
```

5.2.2 IsXp

▷ `IsXp(G, p)` (function)

This function returns `true` if `G` satisfies $\mathscr{X}_p$, where p is a prime, and `false` otherwise.

A group G satisfies $\mathscr{X}_p$ when for every subgroup H of a Sylow p-subgroup P of G, H is permutable in $N_G(P)$. This property was introduced by Beidleman, Brewster, and Robinson in [BBR99]. A group G is a soluble PT-group if and only if G satisfies $\mathscr{X}_p$ for all primes p.

Example

```
gap> g:=SmallGroup(189,7);
<pc group of size 189 with 4 generators>
gap> IsXp(g,3);
true
gap> IsXp(g,7);
true
gap> IsPTGroup(g);
true
gap> IsCp(g,3);
false
gap> IsTGroup(g);
false
```

5.2.3 IsYp

▷ `IsYp(G, p)` (function)

This function returns `true` if `G` satisfies $\mathscr{Y}_p$, where p is a prime, and `false` otherwise.

A group G satisfies $\mathscr{X}_p$ when for every two subgroups H and K with $H \leq K$, H is S-permutable in $N_G(P)$. This property was introduced by Ballester-Bolinches and Esteban-Romero in [BBER02]. A group G is a soluble PST-group if and only if G satisfies $\mathscr{Y}_p$ for all primes p.

Example

```
gap> g:=SmallGroup(200,43); # semidirect product of D8 with C5xC5
<pc group of size 200 with 5 generators>
gap> IsCp(g,2);
false
gap> IsXp(g,2);
false
gap> IsYp(g,2);
true
gap> g:=Group((1,2,3)(4,5,6),(1,2));
Group([ (1,2,3)(4,5,6), (1,2) ])
gap> IsYp(g,3);
false
gap> IsYp(g,2);
true
```

5.3 Auxiliary Functions for T-groups, PT-groups, and PST-groups

The following functions are used to check whether or not a group is a soluble T-group, PT-group, or PST-group.

5.3.1 IsAbCp

▷ `IsAbCp(G, p)` (function)

This function returns `true` if `G` has an abelian Sylow `p`-subgroup `p` such that every subgroup of P is normal in the Sylow normaliser $\mathrm{N}_G(P)$, and `false` otherwise.

This function is used to characterise soluble PST-groups: a group G is a soluble PST-group if and only if G satisfies $\mathscr{Y}_p$ for all primes p, and a group G satisfies $\mathscr{Y}_p$ if and only if G is p-nilpotent or G has an abelian Sylow p-subgroup and satisfies $\mathscr{C}_p$. A group G satisfies $\mathscr{C}_p$ if and only if every subgroup of a Sylow p-subgroup P of G is normal in the Sylow normaliser $\mathrm{N}_G(P)$. Therefore this function checks whether G has an abelian Sylow p-subgroup and G satisfies $\mathscr{C}_p$.

Example

```
gap> g:=AlternatingGroup(5);
Alt( [ 1 .. 5 ] )
gap> IsAbCp(g,5);
true
```

5.3.2 IsDedekindSylow

▷ `IsDedekindSylow(G, p)` (function)

This function returns `true` if a Sylow p-subgroup of `G` is Dedekind, else it returns `false`.

A group G is Dedekind when every subgroup of G is normal. If p is a prime, a Dedekind p-group (see for example 2.3.12 in [Sch94]) is abelian or a direct product of a quaternion group of order 8 and an elementary abelian 2-group. Obviously, a p-group is Dedekind if and only if it is a T-group.

The algorithm used in this function to test whether a non-abelian 2-group satisfies this condition checks that the Frattini subgroup of a Sylow 2-subgroup P of G has order 2 and that the centre of P has exponent 2 and index 4. In this case, it computes the natural epimorphism from P to $P/\mathrm{Z}(P)$ and it checks that the preimages of the generators of $P/\mathrm{Z}(P)$ under the natural epimorphism have order 4. If all these conditions hold, then the Sylow 2-subgroup is Dedekind, otherwise it is not.

This function tries to set the property `IsTGroup` (4.2.1) to `true` or `false` for the Sylow p-subgroup.

Example

```
gap> g:=DirectProduct(SmallGroup(8,4),CyclicGroup(5));
<pc group of size 40 with 4 generators>
gap> IsDedekindSylow(g,2);
true
```

5.3.3 IwasawaTripleWithSubgroup

▷ `IwasawaTripleWithSubgroup(G, X, p)` (function)

This function returns an Iwasawa triple for a p-group `G` such that `X` is a member of it, if such a triple exists, and `fail` otherwise. This function is used as an auxiliary function to compute an Iwasawa triple for a group `G`.

An Iwasawa triple for a p-group G is a triple (X,b,s) such that X is an abelian normal subgroup of G with cyclic quotient, b is a generator of a supplement to X in G, and b induces a power automorphism in X of the form $x \to x^{1+p^s}$. A theorem of Iwasawa states that a p-group G has a modular subgroup lattice (or, equivalently, G has all subgroups permutable) if and only if G is a direct product of a quaternion group of order 8 and an elementary abelian 2-group or G has an Iwasawa triple (X,b,s) with $s \geq 2$.

The construction of the Iwasawa triple takes a generator b of a cyclic supplement to X in G. Then we consider a generator a of X of the largest possible order and find an element c of $\langle b \rangle$ and an element s such that $a^c = a^{1+p^s}$. If such an element does not exist, the function returns `fail`. For this element, it checks whether for all generators t of X, the equality $t^c = t^{1+p^s}$ holds. If this holds, it returns the triple (X,c,s); otherwise it returns `fail`.

Example

```
gap> e:=ExtraspecialGroup(27,9);
<pc group of size 27 with 3 generators>
gap> IwasawaTripleWithSubgroup(e,Subgroup(e,[e.1,e.3]),3);
[ Group([ f1, f3 ]), f2, 1 ]
```

5.3.4 IwasawaTriple

▷ `IwasawaTriple(G)` (attribute)

This function computes an Iwasawa triple for the p-group `G`, if it exists. If `G` is not Iwasawa, the function returns `fail`. If `G` is a direct product of an elementary abelian 2-group and a quaternion group of order 8, it returns an empty list. If `G` is Iwasawa, then the function returns an Iwasawa triple for `G`. An Iwasawa triple for a group `G` is a triple (X,b,s) where X is an abelian normal subgroup of G such that G/X is cyclic, b is a generator of a cyclic supplement to X in G, and s is an integer such that for all $x \in X$, $x^b = x^{1+p^s}$. A theorem of Iwasawa states that a p-group G has a modular subgroup lattice (or, equivalently, G has all subgroups permutable) if and only if G is a direct product of an elementary abelian 2-group and a quaternion group of order 8 or G has an Iwasawa triple (X,b,s) with $s \geq 2$ if $p = 2$.

The method followed to find an Iwasawa triple for non-abelian non-Dedekind groups begins with the whole group G. If the group is abelian, it returns the Iwasawa triple $(G, 1, \log_p \exp(G))$. If it is not abelian, it constructs a list l formed by G. For every element N of l, it takes the maximal subgroups of N which are normal in G and give cyclic quotient. If any of these subgroups is a member of an Iwasawa triple, it is computed with the function `IwasawaTripleWithSubgroup` (5.3.3) and the value is returned. If not, N is removed from the l and these maximal subgroups of N are added to l. This follows until an Iwasawa triple is found or the list l is empty. Since normal subgroups with cyclic quotient are contained in a unique maximal chain, no subgroup appears twice in this algorithm.

The algorithm also takes into account the fact that a Iwasawa group of exponent 4 must be abelian or a direct product of a quaternion group of order 8 and an elementary abelian 2-group.

For the trivial group, it returns the triple composed by the trivial group, its identity element, and the prime 3.

Example

```
gap> e:=ExtraspecialGroup(27,3);
<pc group of size 27 with 3 generators>
```

```
gap> IwasawaTriple(e);
fail
gap> e:=ExtraspecialGroup(27,9);
<pc group of size 27 with 3 generators>
gap> IwasawaTriple(e);
[ Group([ f1, f3 ]), f2, 1 ]
```

5.3.5 IsIwasawaSylow

▷ `IsIwasawaSylow(G, p)` (function)

This function returns `true` if G has an Iwasawa (modular) Sylow p-subgroup, and `false` otherwise.

Recall that a p-group P has a modular subgroup lattice, or is an Iwasawa group, when all subgroups of P are permutable. It is clear that a p-group has a modular subgroup lattice if and only if it is a T-group.

The implementation of this function begins by searching for a pair of subgroups that do not permute. In this case, the function returns `false`. The maximum number of pairs to be checked here is controlled by the variable `PermutMaxTries` (3.2.1), which is assigned to 10 by default. If no such pair is found, the algorithm looks for an Iwasawa triple for a Sylow p-subgroup P of G. If there exists one such triple (X,b,s) with $s \geq 2$ when $p = 2$ or the group is a direct product of a quaternion group of order 8 and an elementary abelian 2-group, then it returns `true`; else it returns `false`.

The values of the attributes `IsPTGroup` (4.2.2) and `IsTGroup` (4.2.1) for P are set by the function.

Example

```
gap> e:=ExtraspecialGroup(27,9);
<pc group of size 27 with 3 generators>
gap> IsIwasawaSylow(e,3);
true
```

Chapter 6

Totally and Mutually Permutable Products

In recent years, many authors have considered totally and mutually permutable subgroups. Recall that two subgroups A and B of a group G are *totally permutable* if every subgroup of A permutes with every subgroup of B, and they are *mutually permutable* if every subgroup of A permutes with B and every subgroup of B permutes with A.

We have defined some "One" functions which give a pair of subgroups which do not permute and prove that two subgroups fail to have a certain property.

We have also defined some functions to work with totally and mutually f-permutable subgroups, where f is a subgroup embedding functor.

The functions of this chapter are defined in a preliminary state.

6.1 Functions for Mutually and Totally Permutable Products

6.1.1 AreMutuallyPermutableSubgroups

▷ `AreMutuallyPermutableSubgroups([G, ]A, B)` (function)

This function returns `true` if the subgroups A and B of G are mutually permutable subgroups, that is, every subgroup of A permutes with B and every subgroup of B permutes with A, and `false` otherwise. The method used here checks only that A permutes with all cyclic subgroups of B and that B permutes with all cyclic subgroups of A.

The method with two arguments assume that A and B have a common supergroup.

6.1.2 OnePairShowingNotMutuallyPermutableSubgroups

▷ `OnePairShowingNotMutuallyPermutableSubgroups([G, ]A, B)` (function)

This function returns a pair of the form [`A`, `V`] with `V` a subgroup of `B` or of the form [`W`, `B`] with `W` a subgroup of `A` in which both subgroups do not permute, or `fail` if this pair does not exist because the product is mutually permutable.

6.1.3 AreTtotallyPermutableSubgroups

▷ `AreTotallyPermutableSubgroups(`*`[G, ]A, B`*`)` (function)

This function returns `true` if the subgroups A and B of G are totally permutable, that is, every subgroup of A permutes with every subgroup of B, and `false` otherwise. The method used here checks only that every cyclic subgroup of A permutes with every cyclic subgroup of B.

The method with two arguments assume that A and B have a common supergroup.

6.1.4 OnePairShowingNotTotallyPermutableSubgroups

▷ `OnePairShowingNotTotallyPermutableSubgroups(`*`[G, ]A, B`*`)` (function)

This function returns a pair of the form [*V*, *W*], with *V* a subgroup of *A* and *W* a subgroup of *B*, such that both subgroups do not permute, or `fail` if this pair does not exist because the product is totally permutable.

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> a:=AlternatingGroup(4);
Alt( [ 1 .. 4 ] )
gap> b:=Subgroup(g,[(1,2,3,4),(1,3)]);
Group([ (1,2,3,4), (1,3) ])
gap> AreMutuallyPermutableSubgroups(g,a,b);
true
gap> AreTotallyPermutableSubgroups(g,a,b);
false
gap> OnePairShowingNotTotallyPermutableSubgroups(g,a,b);
[ Group([ (2,3,4) ]), Group([ (1,2)(3,4) ]) ]
gap> c:=Subgroup(g,[(1,2,3)]);
Group([ (1,2,3) ])
gap> AreMutuallyPermutableSubgroups(g,a,c);
false
gap> OnePairShowingNotMutuallyPermutableSubgroups(g,a,c);
[ Group([ (2,3,4) ]), Group([ (1,2,3) ]) ]
gap> AreMutuallyPermutableSubgroups(a,c);
false
gap> g:=SymmetricGroup(3);
Sym( [ 1 .. 3 ] )
gap> a:=AlternatingGroup(3);
Alt( [ 1 .. 3 ] )
gap> b:=Subgroup(g,[(1,2)]);
Group([ (1,2) ])
gap> AreTotallyPermutableSubgroups(g,a,b);
true
```

6.1.5 AreMutuallyFPermutableSubgroups

▷ `AreMutuallyFPermutableSubgroups(`*`[G, ]A, B, fA, fB`*`)` (function)

This function returns `true` if the subgroups `A` and `B` are mutually `f`-permutable, and `false` otherwise. Here `A` and `B` are subgroups of `G` and `fA` and `fB` are, respectively, lists of subgroups of `A` and `B`, respectively.

In the version with four arguments, A and B are assumed to be subgroups of a common supergroup.

6.1.6 OnePairShowingNotMutuallyFPermutableSubgroups

▷ `OnePairShowingNotMutuallyFPermutableSubgroups(`*`G, A, B, fA, fB`*`)` (function)

This function returns a pair of the form [`A`, `V`] with `V` a subgroup in `fB` or `B` or of the form [`W`, `B`] with `W` a subgroup in `fA` or `A` in which both subgroups do not permute, or `fail` if this pair does not exist. Here `A` and `B` are subgroups of `G` and `fA` and `fB` are lists of subgroups of `A` and `B`, respectively.

In the version with four arguments, `A` and `B` are assumed to be subgroups of a common supergroup.

6.1.7 AreTotallyFPermutableSubgroups

▷ `AreTotallyFPermutableSubgroups(`*`G, A, B, fA, fB`*`)` (function)

This function returns `true` if the subgroup `A` permutes with all subgroups in the list `fB` and `B` permutes with all subgroups in the list `fA`, and `false` otherwise. Here `A` and `B` are subgroups of `G`, `fA` is a list of subgroups of `A` and `fB` is a list of subgroups of `B`.

In the version with four arguments, `A` and `B` are assumed to be subgroups of a common supergroup.

6.1.8 OnePairShowingNotTotallyFPermutableSubgroups

▷ `OnePairShowingNotTotallyFPermutableSubgroups(`*`G, A, B, fA, fB`*`)` (function)

This function returns a pair of the form [`U`, `V`] with `U` a subgroup in `fA` or `A` and `V` a subgroup in `fB` or `B` in which both subgroups do not permute, or `fail` if this pair does not exist. Here `A` and `B` are subgroups of `G`, `fA` is a list of subgroups of `A` and `fB` is a list of subgroups of `B`.

In the version with two arguments, `A` and `B` are assumed to be subgroups of a common supergroup.

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> a:=AlternatingGroup(4);
Alt( [ 1 .. 4 ] )
gap> b:=Subgroup(g,[(1,2,3,4),(1,3)]);
Group([ (1,2,3,4), (1,3) ])
gap> AreTotallyFPermutableSubgroups(g,a,b,
>      MaximalSubgroups(a),MaximalSubgroups(b));
false
gap> OnePairShowingNotTotallyFPermutableSubgroups(g,a,b,
>      MaximalSubgroups(a),MaximalSubgroups(b));
[ Group([ (1,2,3) ]), Group([ (2,4), (1,3)(2,4) ]) ]
gap> AreTotallyFPermutableSubgroups(g,a,b,DerivedSeries(a),DerivedSeries(b));
true
```

Chapter 7

Other Functions in the PERMUT Package

In this chapter we define some miscellaneous functions which have appeared in the context of permutability, or some functions which have been used for some of the functions of the package.

7.1 Functions

7.1.1 AllSubnormalSubgroups

▷ `AllSubnormalSubgroups(G)` (attribute)

This function computes all subnormal subgroups of `G`. The method used to obtain this list consists in beginning with the list of all normal subgroups of `G` and by adding all normal subgroups of the subgroups in the list until no new subnormal subgroups appear. This computes the complete list of subgroups, not only a representative of each conjugacy class as other functions do.

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> AllSubnormalSubgroups(g);
[ Sym( [ 1 .. 4 ] ), Group([ (2,4,3), (1,4)(2,3), (1,3)(2,4) ]), Group([ (1,4)
  (2,3), (1,3)(2,4) ]), Group(()), Group([ (1,3)(2,4) ]), Group([ (1,2)
  (3,4) ]), Group([ (1,4)(2,3) ]) ]
```

7.1.2 PrimesDividingSize

▷ `PrimesDividingSize(G)` (attribute)

This attribute gives a list of primes dividing the size of the finite group `G`, without repetitions. Its code has been borrowed from the GAP manual.

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> PrimesDividingSize(g);
[ 2, 3 ]
```

7.1.3 SylowSubgroups

▷ `SylowSubgroups(`*G*`)` (attribute)

This attribute returns a list composed by one Sylow subgroup for every prime dividing the size of *G*. If *G* is soluble, then it returns a Sylow system or Sylow basis of *G* by means of the function `SylowSystem` (**Reference: SylowSystem**) (a set composed of a Sylow subgroup for each prime dividing the order of *G* permuting in pairs).

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> SylowSubgroups(g);
[ Group([ (1,2), (3,4), (1,3)(2,4) ]), Group([ (1,2,3) ]) ]
gap> s5:=SymmetricGroup(5);
Sym( [ 1 .. 5 ] )
gap> SylowSubgroups(s5);
[ Group([ (1,2), (3,4), (1,3)(2,4) ]), Group([ (1,2,3) ]), Group([ (1,2,3,4,
   5) ]) ]
```

7.1.4 IsSCGroup

▷ `IsSCGroup(`*G*`)` (property)

This property is `true` if *G* is an SC-group, and `false` otherwise. A group *G* is an SC-group if all its chief factors are simple. Note that a soluble group *G* is an SC-group if and only if *G* is supersoluble. The method used to check this property uses the chief series if its is available or the group is not soluble.

Since the methods for insoluble groups might on the computation of a chief series with the function `ChiefSeries` (**Reference: ChiefSeries**), they might not be available if the group is not given as a permutation group.

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> IsSCGroup(g);
false
gap> g:=GL(2,5);
GL(2,5)
gap> IsSCGroup(g);
true
```

7.1.5 IsSylowTowerGroup

▷ `IsSylowTowerGroup(`*G*`)` (property)

This property takes the value `true` if *G* has a Sylow tower of supersoluble type, and `false` otherwise.

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
```

```
gap> IsSylowTowerGroup(g);
false
gap> g:=SmallGroup(75,1);
<pc group of size 75 with 3 generators>
gap> IsSylowTowerGroup(g);
true
```

7.1.6 Permutizer

▷ Permutizer(*G*, *U*) (function)
▷ Permutiser(*G*, *U*) (function)

The permutiser of a subgroup *U* of a group *G* is the subgroup generated by all cyclic subgroups of *G* which permute with *U*. If *U* is permutable in *G* (in particular, if *U* is normal in *G*), then its permutizer coincides with *G*.

Example

```
gap> g:=SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> Permutizer(g,Subgroup(g,[(1,2,3)]));
Group([ (1,2,3), (2,3) ])
gap> Size(last);
6
```

7.1.7 AllGeneratorsCyclicPGroup

▷ AllGeneratorsCyclicPGroup(*g*, *p*) (function)

This auxiliary function returns the list of all generators of the cyclic p-group generated by the p-element g. Here p is a prime number. Since this function is not intended to be used in interactive mode, no check is done that the argument is a p-element.

Example

```
gap> AllGeneratorsCyclicPGroup((1,2,3,4,5,6,7,8,9),3);
[ (1,2,3,4,5,6,7,8,9), (1,3,5,7,9,2,4,6,8), (1,5,9,4,8,3,7,2,6),
  (1,6,2,7,3,8,4,9,5), (1,8,6,4,2,9,7,5,3), (1,9,8,7,6,5,4,3,2) ]
```

References

[BBBC+09] A. Ballester-Bolinches, J. C. Beidleman, J. Cossey, R. Esteban-Romero, M. F. Ragland, and J. Schmidt. Permutable subnormal subgroups of finite groups. *Arch. Math.*, 92:549–557, 2009. 3

[BBBH03a] A. Ballester-Bolinches, J. C. Beidleman, and H. Heineken. Groups in which Sylow subgroups and subnormal subgroups permute. *Illinois J. Math*, 47(1-2):63–69, 2003. 18, 19

[BBBH03b] A. Ballester-Bolinches, J. C. Beidleman, and H. Heineken. A local approach to certain classes of finite groups. *Comm. Algebra*, 31(12):5931–5942, 2003. 18, 19

[BBCLER13] A. Ballester-Bolinches, E. Cosme-Llópez, and R. Esteban-Romero. Algorithms for permutability in finite groups. *Cent. Eur. J. Math.*, 11(11):1914–1922, 2013. 3

[BBER02] A. Ballester-Bolinches and R. Esteban-Romero. Sylow permutable subnormal subgroups of finite groups. *J. Algebra*, 251(2):727–738, 2002. 23

[BBERA10] A. Ballester-Bolinches, R. Esteban-Romero, and M. Asaad. *Products of finite groups*. Walter de Gruyter, Berlin, 2010. 3

[BBERR07] A. Ballester-Bolinches, R. Esteban-Romero, and M. Ragland. A note on finite PST-groups. *J. Group Theory*, 10(2):205–210, 2007. 3, 16

[BBERR09] A. Ballester-Bolinches, R. Esteban-Romero, and M. Ragland. Corrigendum: A note on finite PST-groups. *J. Group Theory*, 12(6):961–963, 2009. 3

[BBR99] J. C. Beidleman, B. Brewster, and D. J. S. Robinson. Criteria for permutability to be transitive in finite groups. *J. Algebra*, 222(2):400–412, 1999. 23

[BH03] J. C. Beidleman and H. Heineken. Finite soluble groups whose subnormal subgroups permute with certain classes of subgroups. *J. Group Theory*, 6(2):139–158, 2003. 18, 19

[EW03] B. Eick and C. R. B. Wright. *GAP package FORMAT — Computing with formations of finite solvable groups v.˜1.3*, 2003. Available on `https://www.uoregon.edu/~wright/RESEARCH/format/` (last visited 30th July 2015). 3, 5

[Fog97] T. Foguel. Conjugate-permutable subgroups. *J. Algebra*, 191:235–239, 1997. 11

[Rob68] D. J. S. Robinson. A note on finite groups in which normality is transitive. *Proc. Amer. Math. Soc.*, 19:933–937, 1968. 22

[Sch94] R. Schmidt. *Subgroup lattices of groups*, volume 14 of *De Gruyter Expositions in Mathematics*. Walter de Gruyter, Berlin, 1994. 24

Index