

SmallGrp

The **GAP** Small Groups Library

1.7.0

18 August 2026

Hans Ulrich Besche
Bettina Eick
Eamonn O'Brien

Bettina Eick Email: beick@tu-bs.de
Homepage: <http://www.iaa.tu-bs.de/beick>
Address: Institut Analysis und Algebra
TU Braunschweig
Universitätsplatz 2
D-38106 Braunschweig
Germany

Eamonn O'Brien Email: obrien@math.auckland.ac.nz
Homepage: <https://www.math.auckland.ac.nz/~obrien>
Address: Department of Mathematics
University of Auckland
Private Bag 92019
Auckland
New Zealand

Contents

1	The Small Groups Library	3
1.1	Overview	3
1.2	Related packages	4
1.3	Function Reference	4
2	The Organisation of the Small Groups Library	11
2.1	Layers	11
2.2	Files and Directories	12
2.3	Layer 1	13
2.4	Layer 2	13
2.5	Layer 3	14
2.6	Layer 4	16
2.7	Layer 5	16
2.8	Layer 6	17
2.9	Layer 7	17
2.10	Layer 8	18
2.11	Layer 9	19
2.12	Layer 10	20
2.13	Layer 11	20
2.14	Concluding remarks	20
	References	22
	Index	23

Chapter 1

The Small Groups Library

1.1 Overview

The Small Groups library gives access to all groups of certain “small” orders. The groups are sorted by their orders and they are listed up to isomorphism; that is, for each of the available orders a complete and irredundant list of isomorphism type representatives of groups is given. Currently, the library contains the following groups:

- those of order at most 2000 except 1024 (423 164 062 groups);
- those of cubefree order at most 50 000 (395 703 groups);
- those of order p^7 for the primes $p = 3, 5, 7, 11$ (907 489 groups);
- those of order p^n for $n \leq 6$ and all primes p ;
- those of order $q^n \cdot p$ for q^n dividing $2^8, 3^6, 5^5$ or 7^4 and all primes p with $p \neq q$;
- those of squarefree order;
- those whose order factorises into at most 3 primes.

The first three items in this list cover an explicit range of orders; the last four provide access to infinite families of groups having orders of certain types.

The library also has an identification function: it returns the library number of a given group. This function determines library numbers using invariants of groups. The function is available for all orders in the library except for the orders 512 and 1536 and except for the orders p^5 , p^6 and p^7 above 2000.

The library is organised in 11 layers. Each layer contains the groups of certain orders and their corresponding group identification routines. It is possible to install the first n layers of the group library and the first m layers of the group identification for each $1 \leq m \leq n \leq 11$. This might be useful to save disk space. Which orders belong to which layer is described in Chapter 2, along with detailed information on how the groups were determined, how they are stored, and how they are identified.

The data in this library has been carefully checked and cross-checked. It is believed to be reliable. However, no absolute guarantees are given and users should, as always, make their own checks in critical cases.

This library has been constructed by Hans Ulrich Besche, Bettina Eick and E. A. O'Brien. A survey on this topic and an account of the history of group constructions can be found in [BEO02]. Further detailed information on the construction of this library is available in [New77], [O'B90], [O'B91], [BE99a], [BE99b], [BE01], [BEO01], [EO99a], [EO99b], [NOV04], [Gir03], [DE05], [OV05]. The Small Groups library incorporates the GAP 3 libraries TwoGroup and ThreeGroup. The data from these libraries was directly included into the Small Groups library, and the ordering there was preserved. The Small Groups library replaces the GAP 3 library of solvable groups of order at most 100. However, both the organisation and the data descriptions of these groups have changed in the Small Groups library.

As of version 1.4 of this library, the arrangement of groups is the same as in Magma, version 2.23. In earlier releases of this library, the arrangement in orders p^7 , $p = 3, 5, 7, 11$ disagreed. An attempt to fix this was made in version 1.1 of this library, but a wrong permutation was used. If you would like to refer to index numbers for these orders in older versions of the library, see SMALL_GROUPS_OLD_ORDER (1.3.15). The arrangement of all other orders has always agreed and has remained stable.

In version 1.5, the number of groups of order 1024 was corrected. For more information, refer to [Bur21].

1.2 Related packages

Several other GAP packages give access to groups of orders which are not covered by this library, or which are covered only in part. A package can make its groups available through the functions of this chapter by calling SmallGroupsAddLayer (1.3.13).

The SglPPow package provides the groups of order p^7 for primes $p > 11$ and the groups of order 3^8 . It extends this library: once it is loaded, its groups are available through the usual functions SmallGroup (1.3.1), NumberSmallGroups (1.3.5) and so on, and SmallGroupsAvailable (1.3.2) takes them into account. See <https://gap-packages.github.io/sglppow/>.

The SOTGrps package constructs and identifies the groups whose order factorises into at most 4 primes, as well as those of order p^4q for distinct primes p and q . It extends this library for those of its orders this one does not cover, and comes with its own functions SOTGroup, AllSOTGroups and IdSOTGroup for all of them. Be aware that where the orders covered by both overlap, the numbering used by SOTGrps in general differs from the one used here. See <https://gap-packages.github.io/sotgrps/>.

1.3 Function Reference

1.3.1 SmallGroup (for group order and index)

▷ SmallGroup(*order*, *i*) (function)
 ▷ SmallGroup(*pair*) (function)

returns the *i*-th group of order *order* in the catalogue. If the group is solvable, it will be given as a PcGroup; otherwise it will be given as a permutation group. If the groups of order *order* are not installed, the function reports an error and enters a break loop.

Example

```
gap> G := SmallGroup( 60, 4 );
<pc group of size 60 with 4 generators>
gap> StructureDescription( G );
```

```

"C60"
gap> G := SmallGroup( 60, 5 );
Group([ (1,2,3,4,5), (1,2,3) ])
gap> StructureDescription( G );
"A5"
gap> G := SmallGroup( 768, 1000000 );
<pc group of size 768 with 9 generators>
gap> G := SmallGroup( [768, 1000000] );
<pc group of size 768 with 9 generators>

```

1.3.2 SmallGroupsAvailable

▷ `SmallGroupsAvailable(order)` (function)

returns true if the library of groups of order *order* is installed, and false otherwise.

1.3.3 AllSmallGroups

▷ `AllSmallGroups(arg)` (function)

returns all groups with certain properties as specified by *arg*. If *arg* is a number *n*, then this function returns all groups of order *n*. However, the function can also take several arguments which then must be organised in pairs function and value. In this case the first function must be `Size` (**Reference:** `Size`) and the first value an order or a range of orders. If a value is a list then it is considered a list of possible function values to include. The function returns those groups of the specified orders having those properties specified by the remaining functions and their values.

Some selection criteria are *indexed*: their value follows from the position a group has in the library, so a selection using only such criteria constructs no group at all. Any other criterion is checked group by group, at a cost proportional to the number of groups of the given orders.

For the orders whose factorisation has more than three prime factors, `IsNilpotentGroup` (**Reference:** `IsNilpotentGroup`), `IsSupersolvableGroup` (**Reference:** `IsSupersolvableGroup`) and `IsSolvableGroup` (**Reference:** `IsSolvableGroup`) are indexed, except `IsSupersolvableGroup` (**Reference:** `IsSupersolvableGroup`) for the cubefree orders above 2000 which are not squarefree, and so is `IsAbelian` (**Reference:** `IsAbelian`), except for p^5 with $p \geq 7$. For the prime powers p^n with $n \geq 4$, `RankPGroup` (**Reference:** `RankPGroup`) is indexed except for p^6 with $p \geq 11$ and for 7^7 and 11^7 , and `PClassPGroup` (**Reference:** `PClassPGroup`) except for those and for p^4 with $p \geq 11$ and p^5 with $p \geq 7$. `LGLength` (**Reference:** `LGLength`), `FrattinifactorSize` and `FrattinifactorId` are indexed for the orders of at most 2000 with more than three prime factors, except 512, 768, 1024, 1152, 1536, 1920 and $p^n \cdot q > 1000$ with $n > 2$.

Orders with at most three prime factors are not indexed; no such order below 50000 has more than 13 groups.

`SmallGroupsInformation` (1.3.12) reports which criteria are indexed for a given order.

`IdsOfAllSmallGroups` (1.3.10) returns the ids of these groups instead of the groups themselves, and `NumberSmallGroups` (1.3.5) just their number.

Example

```

gap> AllSmallGroups( 6 );
[ <pc group of size 6 with 2 generators>,
  <pc group of size 6 with 2 generators> ]

```

```
gap> AllSmallGroups( 60, IsNilpotentGroup );
[ <pc group of size 60 with 4 generators>,
  <pc group of size 60 with 4 generators> ]
```

1.3.4 OneSmallGroup

▷ OneSmallGroup(*arg*)

(function)

returns one group with certain properties as specified by *arg*. The permitted arguments are those supported by AllSmallGroups (1.3.3).

Example

```
gap> G := OneSmallGroup( 6, IsAbelian );
<pc group of size 6 with 2 generators>
gap> StructureDescription( G );
"C6"
gap> G := OneSmallGroup( 6, IsAbelian, false );
<pc group of size 6 with 2 generators>
gap> StructureDescription( G );
"S3"
gap> G := OneSmallGroup( Size, [1..1000], IsSolvableGroup, false );
Group([ (1,2,3,4,5), (1,2,3) ])
```

1.3.5 NumberSmallGroups

▷ NumberSmallGroups(*order*[, *func1*, *val1*, ...])

(function)

▷ NrSmallGroups(*order*[, *func1*, *val1*, ...])

(function)

returns the number of groups of order *order*.

Example

```
gap> NumberSmallGroups( 512 );
10494213
gap> NumberSmallGroups( 2^8 * 23 );
1083472
gap> NumberSmallGroups( 4096 );
Error, the library of groups of size 4096 is not available
```

The function also accepts the arguments of AllSmallGroups (1.3.3) and counts the groups that function would return.

Example

```
gap> NumberSmallGroups( [ 1 .. 100 ] );
1048
gap> NumberSmallGroups( 96, IsAbelian );
7
```

Orders alone are answered from the numbers of groups. Otherwise the cost is that of IdsOfAllSmallGroups (1.3.10): the indexed criteria are applied first, whatever order the criteria are given in, and only the groups a remaining criterion must be tested on are constructed.

1.3.6 NumberSmallGroupsAvailable

▷ `NumberSmallGroupsAvailable(order)` (function)

returns true if the number of groups of order *order* is known, and false otherwise.

Example

```
gap> NumberSmallGroupsAvailable( 100 );
true
gap> NumberSmallGroups( 100 );
16
gap> NumberSmallGroupsAvailable( 4096 );
false
gap> NumberSmallGroups( 4096 );
Error, the library of groups of size 4096 is not available
```

1.3.7 SelectSmallGroups

▷ `SelectSmallGroups(argl, all, id)` (function)

universal function for `AllSmallGroups`, `OneSmallGroup` and `IdsOfAllSmallGroups`.

1.3.8 IdSmallGroup

▷ `IdSmallGroup(G)` (attribute)

▷ `IdGroup(G)` (attribute)

returns the library number of *G*; that is, the function returns a pair [*order*, *i*] where *G* is isomorphic to `SmallGroup( order, i )`.

Example

```
gap> IdSmallGroup( GL( 2,3 ) );
[ 48, 29 ]
gap> IdSmallGroup( Group( (1,2,3,4), (4,5) ) );
[ 120, 34 ]
```

1.3.9 IdGroupsAvailable

▷ `IdGroupsAvailable(order)` (function)

returns true if the identification routines for groups of order *order* are installed, and false otherwise.

1.3.10 IdsOfAllSmallGroups

▷ `IdsOfAllSmallGroups(arg)` (function)

similar to `AllSmallGroups` but returns ids instead of groups. This may prevent workspace overflows if a large number of groups are expected in the output.

Example

```
gap> IdsOfAllSmallGroups( 60, IsNilpotentGroup );
[ [ 60, 4 ], [ 60, 13 ] ]
gap> IdsOfAllSmallGroups( 60, IsNilpotentGroup, false );
[ [ 60, 1 ], [ 60, 2 ], [ 60, 3 ], [ 60, 5 ], [ 60, 6 ], [ 60, 7 ],
  [ 60, 8 ], [ 60, 9 ], [ 60, 10 ], [ 60, 11 ], [ 60, 12 ] ]
gap> IdsOfAllSmallGroups( Size, 60, IsSupersolvableGroup, true );
[ [ 60, 1 ], [ 60, 2 ], [ 60, 3 ], [ 60, 4 ], [ 60, 6 ], [ 60, 7 ],
  [ 60, 8 ], [ 60, 10 ], [ 60, 11 ], [ 60, 12 ], [ 60, 13 ] ]
```

1.3.11 IdGap3SolvableGroup

- ▷ IdGap3SolvableGroup(G) (attribute)
- ▷ Gap3CatalogueIdGroup(G) (attribute)

returns the catalogue number of G in the GAP 3 catalogue of solvable groups; that is, the function returns a pair $[order, i]$ meaning that G is isomorphic to the group SolvableGroup($order, i$) in GAP 3.

1.3.12 SmallGroupsInformation

- ▷ SmallGroupsInformation($order$) (function)

prints information on the groups of the specified order.

Example

```
gap> SmallGroupsInformation( 32 );

There are 51 groups of order 32.
They are sorted by their ranks.
  1 is cyclic.
  2 - 20 have rank 2.
 21 - 44 have rank 3.
 45 - 50 have rank 4.
 51 is elementary abelian.

The following selection criteria are indexed for this size:
  IsAbelian, IsNilpotentGroup, IsSupersolvableGroup, IsSolvableGroup,
  PClassPGroup, RankPGroup, FrattinifactorSize and FrattinifactorId.

This size belongs to layer 2 of the SmallGroups library.
IdSmallGroup is available for this size.
```

1.3.13 SmallGroupsAddLayer

- ▷ SmallGroupsAddLayer($desc$) (function)

adds a further layer to the library, described by the record $desc$. Its groups then become available through SmallGroup (1.3.1), NumberSmallGroups (1.3.5), AllSmallGroups (1.3.3) and the other functions of this chapter. The components of $desc$ are:

name

a string naming the layer. Must be unique. Can be used by other layers to refer to this one.

available

a function taking an order and either returning `fail`, to indicate this layer does not handle the given order; or else a record which is handed to the functions below as `inforec`. The component number of the returned record, if present, is the number of groups of that order.

group

a function (`order` , `i` , `inforec`) returning the *i*-th group of that order.

idAvailable

optional, a function taking an order and either returning `fail`, to indicate this layer does not handle identification of groups of the given order; or else a record which is handed to the function `id` below as `idrec`. Defaults to `available`, so it is only needed where the identification covers other orders than the construction, or wants a record of its own. If `idAvailable` is defined then `id` must also be provided.

id optional, a function (`G` , `idrec`) returning the index of *G* among the groups of its order. Without it `IdSmallGroup` (1.3.8) stays unavailable for these orders.

number

optional, a function (`order` , `inforec`) returning the number of groups of that order. Only needed if `available` does not report this count.

information

optional, a function (`order` , `inforec` , `number`) printing what `SmallGroupsInformation` (1.3.12) should say about that order beyond the number of groups.

properties

optional, a function (`order` , `inforec`) reporting which selection criteria the `select` and `count` functions of this layer can deduce from the position of the groups alone, so that `SelectSmallGroups` (1.3.7) and `NumberSmallGroups` (1.3.5) need not construct the groups to decide these criteria.

select, count

optional, how `SelectSmallGroups` (1.3.7) and `NumberSmallGroups` (1.3.5) are to work through this layer. By default the groups of the order are constructed one by one and tested.

before, after

optional lists of names of other layers, to be consulted after respectively before this one. A name that is not registered is ignored, so wishing about a package that is not loaded does no harm. The layers of this package are themselves one layer, named "SmallGrp", which without a wish to the contrary comes first.

Where two layers cover an order, the one consulted first wins.

Example

```
SmallGroupsAddLayer( rec(
  name := "SOTGrps",
  available := function( order )
```

```

        if not IsSOTAvailable( order ) then
            return fail;
        fi;
        return rec( number := NumberOfSOTGroups( order ) );
    end,
    group := { order, i, inforec } -> SOTGroup( order, i ),
    id := { G, inforec } -> IdSOTGroup( G )[2] );

```

1.3.14 UnloadSmallGroupsData

▷ UnloadSmallGroupsData()

(function)

GAP loads all necessary data from the library automatically, but it does not delete the data from the workspace again. Usually, this will not be necessary, since the data is stored in a compressed format. However, if a large number of groups from the library have been loaded, then the user might wish to remove the data from the workspace and this can be done by the above function call.

Example

```
gap> UnloadSmallGroupsData();
```

1.3.15 SMALL_GROUPS_OLD_ORDER

▷ SMALL_GROUPS_OLD_ORDER

(global variable)

If set to true, then groups of order 3^7 , 5^7 , 7^7 , and 11^7 are ordered in the way they were ordered up to version 1.0 of the package. If this variable is set to false, which is the default as of version 1.4, then a different ordering, that agrees with the one in Magma version 2.23, is used. The functions SMALLGP_PERM $_x$, with $x = 3, 5, 7, 11$, give the old index position corresponding to a new index position. In releases 1.1-1.3 a misunderstood ordering, based on the old ordering and the permutations $(2, 30083)(3, 30084)(4, 30085)(5, 30086)$, $(2, 104599)(3, 104600)(4, 104601)(5, 104602)$, and $(2, 721053)(3, 721054)(4, 721055)(5, 721056)$ respectively, was used.

Chapter 2

The Organisation of the Small Groups Library

This chapter describes how the Small Groups library is organised. It includes information on the determination of the groups, on the encoding used to store them, and on the algorithms used by the identification routines. It is aimed at readers interested in the internals of the library; it is not needed in order to use it.

2.1 Layers

The Small Groups library is organised in 11 layers. Each layer contains the groups of certain orders:

Layer 1

the orders with at most 3 prime factors; that is, the orders of type p , p^2 , pq , p^3 , pq^2 and pqr .

Layer 2

all remaining orders at most 1000 except 512 and 768.

Layer 3

all remaining orders of type $2^n \cdot p$ with $n \leq 8$ and p an arbitrary odd prime.

Layer 4

the orders 7^4 and 5^5 and all remaining orders of type $q^n \cdot p$ with q^n dividing 3^6 , 5^5 or 7^4 and p a prime different from q .

Layer 5

all remaining orders at most 2000 except 512, 1024, 1152, 1536 and 1920.

Layer 6

the orders 1152 and 1920.

Layer 7

the order 512.

Layer 8

the order 1536.

Layer 9

the remaining groups of order p^4 , p^5 and p^6 .

Layer 10

the remaining groups of squarefree order and of cubefree order at most 50 000.

Layer 11

the orders p^7 with $p = 3, 5, 7, 11$.

This setup has been chosen for two reasons. First, the organisation in layers permits adding new layers (and thus new group orders) without changes to the existing catalogue. Secondly, it is possible to install a part of the layers only. This might be interesting for computer systems with little hard disk space available.

For each layer i there are two internal functions `SMALL_AVAILABLE_FUNCS[i]( order )` and `ID_AVAILABLE_FUNCS[i]( order )`. Both functions return `fail` if the given order is not contained in this layer. Otherwise the functions return an information record on the layer and the order. This record contains the information needed to handle the groups of the specified order internally. If the layer i is not installed, then the entries `SMALL_AVAILABLE_FUNCS[i]` and `ID_AVAILABLE_FUNCS[i]` are unbound.

Moreover, there are two internal header functions `SMALL_AVAILABLE( order )` and `ID_AVAILABLE( order )`. These two functions are used to loop over the layer functions and find the first layer containing the given order. If this layer is found, then an information record on the order is returned. If there is no installed layer containing the groups of this order, then the functions return `fail`.

2.2 Files and Directories

For each layer x there are either one or two directories in the package directory: `smallx`, or `smallx` and `idx`. The directory `smallx` contains the groups of layer x and the directory `idx` contains the corresponding identification routine, if available. An exception to this rule is the first layer; here all the code for the groups is contained in the files `gap/smlgp1.g` and `gap/idgrp1.g`. For the layers 7, 8 and 11 there is no identification routine available at the moment and thus there are no directories `id7`, `id8` and `id11`.

The other relevant files of the package are:

`read.g`

for loading the small groups catalogue into GAP.

`gap/small.gd`, `gap/small.gi`

these are the files where the main functions for the small groups catalogue are defined.

`gap/gap3cat.g`

identification of a small group in the GAP 3 SolvableGroup catalogue.

`gap/smlinfo.gi`

the code for the function `SmallGroupsInformation` (1.3.12).

2.3 Layer 1

The groups whose order factorises into at most 3 primes have been classified by Hölder, see [Höl93]. An algorithm to find the isomorphism type of such groups was first implemented in GAP 3 by Frank Celler and Hans-Georg Esser.

Hölder’s classification distinguishes the groups of the following order types. Let p, q and r be different primes with $p < q < r$.

- p 1 cyclic group.
- p^2 1 cyclic and 1 elementary abelian group.
- p^3 3 abelian groups and 2 extraspecial groups.
- pq 1 cyclic group; 1 group of type $q : p$ if $q \equiv 1 \pmod{p}$.
- p^2q 2 abelian groups; 1 group of type $q : p \times p$ if $q \equiv 1 \pmod{p}$; 1 group of type A_4 if $p = 2$ and $q = 3$; 1 group of type $(p \times q).p$ if $q \equiv 1 \pmod{p}$; 1 group of type $q : C_{p^2}$ if $q \equiv 1 \pmod{p^2}$.
- pq^2 2 abelian groups; 1 group of type $q : p \times q$ if $q \equiv 1 \pmod{p}$; 1 group of type $C_{q^2} : p$ if $q \equiv 1 \pmod{p}$; 1 group of type $(q \times q) : p$ if $p > 2$ and $q + 1 \equiv 0 \pmod{p}$; groups of type $q : p + q : p$ (1 group for $p = 2$ and $(p + 1)/2$ groups otherwise).
- pqr 1 cyclic group; 1 group of type $q : p \times r$ if $q \equiv 1 \pmod{p}$; 1 group of type $r : p \times q$ if $r \equiv 1 \pmod{p}$; 1 group of type $r : q \times p$ if $r \equiv 1 \pmod{q}$; 1 group of type $r : (p \times q)$ if $r \equiv 1 \pmod{pq}$; $p - 1$ groups of type $q : p + r : p$.

The groups in this layer are stored “generically”; that is, using functions instead of explicit presentations. The identification routine follows the classification as well.

2.4 Layer 2

2.4.1 Determination

The nilpotent groups of this layer have been constructed using p -group generation, see [O’B90] and the ANUPQ package. The 2- and 3-groups of this layer have also been available in the TwoGroup and ThreeGroup libraries of GAP 3, see [JNO90] and [O’B91]. The non-nilpotent groups in this layer have been determined using the Frattini extension method, the cyclic split extension method and cyclic extension for the non-solvable groups, see [BE99a], [BE99b] and [BE01].

2.4.2 Storing

The solvable groups in this layer are stored by a single long integer describing a pc presentation of the group (see PcGroupCode (**Reference: PcGroupCode**) or [BE99a]). For p -groups the encoded pc presentations are standard presentations and they are equal to the presentations known in the 2- and 3-groups libraries of GAP 3. The non-solvable groups are stored by generating permutations.

2.4.3 Selection arguments

For the selection functions the values of the following attributes are precomputed and stored:

for p -groups

IsAbelian (**Reference:** IsAbelian), PClassPGroup (**Reference:** PClassPGroup), RankPGroup (**Reference:** RankPGroup), FrattinifactorSize and FrattinifactorId.

for non- p -groups

IsAbelian (**Reference:** IsAbelian), IsNilpotentGroup (**Reference:** IsNilpotentGroup), IsSupersolvableGroup (**Reference:** IsSupersolvableGroup), IsSolvableGroup (**Reference:** IsSolvableGroup), LGLength, FrattinifactorSize and FrattinifactorId.

2.4.4 Identification

The identification routine uses invariants of groups to identify a given group. The invariants are organised as a tree and stored in ID_GROUP_TREE, which contains a leaf for every group. In some cases this invariants tree is not sufficient to distinguish all groups. In this case the tree is used to reduce to a small number of possible groups and then a special random isomorphism test finds the correct group among the possible ones.

2.5 Layer 3

2.5.1 Determination

These groups have been determined using the generic variation of the cyclic split extension method and the Frattini extension method, see [BE01].

2.5.2 Storing: nilpotent groups

For each order in this layer we first list the nilpotent groups $P \times Q$ where P is the cyclic group of order p and Q is a group of order 2^n . These groups are sorted by the catalogue number of Q and the group Q is stored in layer 2.

2.5.3 Storing: groups with normal Sylow p -subgroup

Then we consider the remaining groups with normal Sylow p -subgroup. These are split extensions of type $P : Q$. The isomorphism type of these groups depends on the isomorphism type of Q and the action homomorphism $Q \rightarrow \text{Aut}(P)$. Let K be the kernel of this homomorphism with $[Q : K] = 2^i$ for some i . We encode the homomorphisms for fixed Q and fixed i as follows.

Consider a generator g of P and let $g_1, \dots, g_r$ be a generating set of Q (we choose the first r pc generators of Q for r the rank of Q). Consider a homomorphism $a : Q \rightarrow \text{Aut}(P)$ and let a_i be the image of g_i . Clearly, a_i is of the form $a_i : g \rightarrow g^{l_i}$ with $0 \leq l_i \leq 2^i - 1$. Thus we describe the homomorphism a by the sequence $l_1, \dots, l_r$. We encode this sequence as a 2^i -adic number of length r ; that is, we encode this sequence as $l_1 q^{r-1} + l_2 q^{r-2} + \dots + l_r q^0$ where $q = 2^i$. Hence each homomorphism a is encoded as an integer and for fixed Q and i we write all these integers into a list.

A special case occurs if there are two homomorphisms a and b such that $a_i = b_i^j$ for some integer j ; that is, the images of a are powers of the images of b . In this case a is stored directly after b and we only store the integer $-j$ instead of the encoding for the sequence $l_1, \dots, l_r$ for a .

Now we consider the case that for certain Q and i we have a list L consisting of non-negative integers only; that is, the special power-case did not occur for this group Q and index i . In some cases we encode such a list L further. Recall that the integers in L correspond to different homomorphisms. Thus the integers are all different and we may sort them in increasing order. Then we can store this sorted list as a binary number; that is, the list $c_1, \dots, c_s$ is encoded as $2^{c_1-1} + \dots + 2^{c_s-1}$. This is only useful if the list L does not contain large numbers.

Thus for Q and i we have either a list or an integer stored to encode the corresponding homomorphisms. For each index i we write all lists or integers into a list such that at position $\text{Id}(Q)$ the corresponding list or integer to Q and i is found.

For $i = 1$ this list has no holes, since there exists at least one group $P : Q$ with K of index 2 for each group Q . Often the entry at position j in this list is equal to the entry at position $j - 1$. In this case we delete the entry at position j and thus create a hole in the list.

For $i > 1$ there might be holes in the list if there is no group $P : Q$ with K of index 2^i . Hence the previous idea cannot be applied for $i > 1$. However, in all the lists we have equal entries in various places. If this is the case, then we substitute an entry in a list by a negative integer $-j$ meaning that the entry at this position is equal to the entry at position j .

Finally, if the list for some index i has only few entries left, then we substitute the list by a record containing two lists: first the non-empty positions and secondly their entries.

2.5.4 Storing: groups with normal Sylow 2-subgroup

Next we consider the remaining groups of type $Q : P$. These exist for a few primes p only. These groups are described by operation homomorphisms of the type $P \rightarrow \text{Aut}(Q)$. Here we store the catalogue number of Q and a long integer for each such homomorphism $a : P \rightarrow \text{Aut}(Q)$. Consider a generator g of P . Clearly, we just need to store the image of g for each homomorphism a . The image g^a is an automorphism of Q and thus we store the images under g^a of a fixed generating set of Q . For this purpose we consider a canonical numbering of the elements of Q and store the generator images by storing their number in the element list. The sequence of these numbers is encoded as a $(|Q| + 1)$ -adic number; that is, the sequence $e_1, \dots, e_r$ is stored as $e_1q^{r-1} + e_2q^{r-2} + \dots + e_r$ where $q = |Q| + 1$.

2.5.5 Storing: groups without normal Sylow subgroup

Now there are the groups of order $2^n \cdot p$ without any normal Sylow subgroup left. These exist for very few primes p only. They have been computed as in layer 2 using the Frattini extension method. They are stored as described in layer 2 using one long integer for each group.

2.5.6 Identification

The identification of nilpotent groups $P \times Q$ relies on the identification of Q only and this is done as in layer 2. Similarly, the groups without normal Sylow subgroup are identified as described in layer 2.

For the groups $Q : P$ with normal Sylow 2-subgroup we first determine the prime p and the isomorphism type of Q . If this is not sufficient, then we use the methods as described in layer 2. (Recall that there are only finitely many primes p possible here.)

For the groups $P : Q$ with normal Sylow p -subgroup we first determine the prime p and the isomorphism type of Q as well. Moreover, we compute the centralizer K of P in Q and consider its index $[Q : K] = 2^i$.

In a large number of cases the catalogue numbers $\text{Id}(Q)$ and $\text{Id}(K)$ are sufficient to determine the group $P : Q$ uniquely. Moreover, this feature is independent of p . We can recognise these cases using the tree of invariants: if we can determine $P : Q$ uniquely with $\text{Id}(Q)$ and $\text{Id}(K)$ only, then there is no node corresponding to this case in the tree.

If there is a node in the tree, then $\text{Id}(Q)$ and $\text{Id}(K)$ are not sufficient to determine the group uniquely. In this case we apply methods similar to layer 2; that is, we use invariants. Since there are infinitely many primes p which might turn up here, we need an additional idea to use the invariant tree for this purpose. In the invariant tree we have invariants stored for groups of type $R : Q$ of order $2^n \cdot r$ for the primes $r = 3, 5, 17$ and 97 . Now we first choose the smallest prime r such that r is congruent to 1 modulo $[Q : K]$. The cases that $[Q : K] = 2^i$ for $i = 6, 7$ or 8 cannot occur here, because in these cases $\text{Id}(Q)$ and $\text{Id}(K)$ are always sufficient to determine the group.

Then we use that the groups $P : Q$ behave essentially similarly to the groups $R : Q$ for the chosen r . Thus we can construct a group $R : Q$ corresponding to $P : Q$ and identify $R : Q$ instead of $P : Q$.

2.6 Layer 4

2.6.1 Determination

The groups of order $2401 = 7^4$ and $3125 = 5^5$ have been computed using p -group generation as described in layer 2 and they are stored and handled similarly. The other groups in this layer have been determined as outlined in layer 3 and, again, they are organised similarly.

2.6.2 Storing

There is a difference from layer 3 in the compression of the group data; that is, the groups in this layer are not compressed as efficiently as the groups in layer 3. We consider the groups of type $P : Q$ for P the cyclic group of order p and Q a group of order q^n . These groups are determined by Q and an operation homomorphism $Q \rightarrow \text{Aut}(P)$. As in layer 3 we first compute one integer for each operation homomorphism and then proceed to compress the lists of integers for each Q and each i where q^i is the index of the centralizer of P in Q . Unlike in layer 3, we do not compress the lists of non-negative integers to a single integer and we do not reduce the list for $i = 1$ further, since in both cases the obtained compression is very small.

2.6.3 Identification

The group identification is essentially similar to the procedure described in layer 3. However, for the groups with normal Sylow p -subgroup we additionally use a special random isomorphism test to identify groups. It is a “special” version of the general algorithm: in the general algorithm we use certain sets to choose generating sets from. In this special version we restrict these sets suitably. (We use non-trivial elements of P and certain elements of Q .) This is necessary, since the groups which appear here can be too large to compute explicitly the elements sorted in conjugacy classes.

2.7 Layer 5

This layer is similar to layer 2.

2.8 Layer 6

2.8.1 Determination

The nilpotent groups in this layer are determined as direct products of p -groups. Most of the groups of orders 1152 and 1920 have a normal Sylow 3-subgroup or Hall $\{3, 5\}$ -subgroup, respectively. They are constructed using the coprime split extension method, see [BEO02]. The remaining groups of these orders (without normal 2-complement) have been determined using the Frattini extension method and the cyclic split extension method, see also [BE99a].

2.8.2 Storing

Only the non-nilpotent groups with normal Sylow 3-subgroup or Hall $\{3, 5\}$ -subgroup are stored in a special way. The remaining groups are organised similarly to layer 2.

For the groups of type $P : Q$ where P is the normal Sylow 3-subgroup or the Hall $\{3, 5\}$ -subgroup and Q is a group of order 2^7 we split a pc presentation in two parts. The first part are the relators of Q . These are stored within the groups of order 2^7 and thus can be considered as known. The second part of the relators determine P and the operation of Q on P . These second parts are stored as long integers. If we consider these second parts for all the groups, then we find that they are highly redundant. There are only 921 (1722) different long integers for the groups of order 1152 and 1920, respectively. Thus we store these two sets of long integers only.

Then for each group Q of order 2^7 we store for each extension $P : Q$ the position of the corresponding long integer. Hence we obtain a list of integers for Q . Again we note that the lists of integers obtained for all the groups Q of order 2^7 are highly redundant. There are only 1298 (722) different lists for the groups of order 1152 (1920). Thus we can compress the lists using the same idea again.

2.8.3 Identification

The identification of groups of these orders is similar to layer 2.

2.9 Layer 7

2.9.1 Determination

The groups of order 512 have first been enumerated, see [EO99b] and [EO99a]. Later they have been determined explicitly using p -group generation, see [O'B90] and [BEO02]. There are 10 494 213 groups of this order.

2.9.2 Storing

We introduce a special method to store such groups. Consider a pc presentation of a given group of order 512. If we concatenate the exponent vectors of the right hand sides of such a presentation, then we obtain a bit vector of length 120. This can be used as encoding of a group of order 512. We compress the list of such encodings further using the following method.

First we split the list of all groups of order 512 in sublists containing 1000 groups each. Let L be such a sublist. We suppose that L consists of 1000 bitlists of length 120 and we describe a method to encode this list of bitlists in a single vector V using an alphabet of 83 symbols (81 symbols for the encoding and 2 special symbols as signs).

First we find those indices i in the range 1 to 120 such that for all lists in L the entries at position i are fixed. Thus we note for each of the 120 entries whether the entries are all 0, all 1 or variable in L . There are 3^{120} combinations possible, which we store in a vector of length 30 using 81 symbols. This vector of length 30 is the initial segment of V . Now we can reduce to consider the variable positions only and thus work with bitlists of shorter length.

We split the shorter bitlists in blocks such that the bitlists in each block have at most 6 variable entries. For this purpose we start at the beginning of L and add bitlists to the current block until the next bitlist would lead to more than 6 variable entries for this block. Each block can contain from 1 up to 64 bitlists. Now we determine a “headcode” and a “tailcode” to describe the block. The head and tail codes are then concatenated to V . The head contains an encoding of the variable bits for this block as above. The tail then describes the vectors in the variable bits. Since there are at most 64 possible combinations arising here, we need only one letter in the 81-alphabet to describe a single vector of variable bits.

Finally, we note that some of the tailcodes arise for many blocks. These tailcodes are stored in a special list. If such a tailcode is occurring in V , then only the reference to the special list is stored instead of the full tailcode. This special list contains 1890 different tailcodes, so 2 letters in the alphabet are sufficient to refer to a tailcode.

To distinguish references and actual tailcodes we use a special sign. Each new head begins with a special symbol in V . There are two special symbols available for this purpose. One is used if the corresponding tailcode is referenced, and the other if it is an ordinary tailcode.

This compression allows the groups of order 512 to be stored in 4.4 MB.

2.10 Layer 8

2.10.1 Determination

The 408 641 062 groups of order 1536 have been determined using the cyclic split extension method and the improved version of the Frattini extension method, see [BE99a] and [BE01].

2.10.2 Storing

We store these groups in a special way as well and we obtain a compression to 6.1 MB. The groups are stored using references to the groups of order 512 of layer 7.

The nilpotent groups $C_3 \times Q$ for Q a group of order 512 are based on the groups of order 512.

The major part of the groups are the 398 032 384 non-nilpotent groups with normal Sylow 3-subgroup. The compression of these groups follows the one outlined in layer 3. As described there, for each group Q of order 512 we compute an integer which determines all groups of type $C_3 : Q$. There are only 15 249 different integers of this type. In a first compression we create a list of references to the different integers.

Now we note that a group often has the same reference as the previous one. Thus we split the groups of order 512 in sets of 100 000 groups and create a record for each set. The record contains two lists which are used to recall the blocks in a set having the same reference. Thus the first list contains the length of the blocks and the second list contains the corresponding references.

Again, in both lists certain patterns occur often. In the first list the length 1 appears often. Thus the 1 is not stored explicitly, but a blank in this list has to be considered as 1. In the second list often an entry is the same as the one 2 places before. Thus these entries are deleted as well and blanks can be reconstructed by this rule.

Additionally we store the number of resulting groups for each integer to speed up the method to find the group with a given catalogue number.

The 18 028 groups with normal Sylow 2-subgroup are stored similarly to layer 3. As described for layer 3 we determine a long integer for each homomorphism $Q \rightarrow \text{Aut}(P)$ for Q of order 512 and P the cyclic group of order 3. The occurring integers are highly redundant – there are 6774 different ones. Thus we use a list of references again.

The 96 437 groups without normal Sylow subgroup are determined using the Frattini extension method. As described in layer 2 they can be stored by encoding the relators of a pc presentation as long integer. However, the relators corresponding to the Frattini factors of the groups are known. Thus it is sufficient to store the relators of the Frattini factors once for all groups with this factor and encode the remaining relators only. Note that there are only 16 different Frattini factors and only 12 of them have order less than 1536. The groups are stored in lists corresponding to the 12 Frattini factors.

2.11 Layer 9

This layer contains the generic construction of the groups of order p^4 , p^5 and p^6 greater than 3125 (those of order up to 3125 are contained in the lower layers of the Small Groups library).

2.11.1 Determination

The groups of order p^4 were first determined by Hölder (1893), see [Höl93]. The groups of order p^5 were first determined by Bagnara (1898), see [Bag98]. The groups of order p^6 were first (correctly) determined by Newman, O’Brien and Vaughan-Lee (2004), see [NOV04].

2.11.2 Storing

The groups of order p^4 ($p \geq 11$) are given by pc presentations which are produced at run-time by a function `SMALL_GROUP_FUNCS[19]` provided by Newman.

The groups of order p^5 ($p \geq 7$) are given by pc presentations which are produced at run-time by a function `SMALL_GROUP_FUNCS[20]` provided by Gyrat, see [Gir03].

The groups of order p^6 ($p \geq 5$) are given by pc presentations which are produced at run-time by a function `SMALL_GROUP_FUNCS[21]` provided by Newman, O’Brien and Vaughan-Lee, see [NOV04].

The groups of order p^6 are given as a list of partially repetitive structures. These are compressed into the file `sm11.z`. At run-time, this compressed structure will be expanded, but restricted to those parts of the structure relevant for the given p when needed. It is cached into `SMALL_GROUP_LIB[1]`. As parts of this structure contain long ($O(p^2)$) lists of groups which are classified by additional parameters and these lists are not dense, it might be necessary to set up the complete list of indices to find the presentation of a single group.

2.11.3 Identification

A group of order p^4 can be identified and its catalogue number found by the function `ID_GROUP_FUNCS[19]` based on a function provided by Newman and modified by Besche.

There is no identification available for the groups of order p^5 and p^6 at present.

2.12 Layer 10

2.12.1 Groups of squarefree order

These groups have first been determined by Hölder (1895), see [Höl95]. The implemented construction is based on the determination given in [DE05]. The key invariants are the socle and socle factor.

The groups of squarefree order which are not contained in lower layers are given by pc presentations which are produced at run-time by a function `SMALL_GROUP_FUNCS` [24] written specially for this library. Such a group can be identified and its catalogue number found by the function `ID_GROUP_FUNCS` [24].

2.12.2 Groups of cubefree but not squarefree order

These groups were first determined in [DE05]. Every group with cubefree order is either solvable or has a direct decomposition into a solvable group and $PSL(2, p)$ for a suitable prime.

The groups of cubefree order at most 50 000 which are not contained in lower layers and are not squarefree are given by pc presentations which are produced at run-time by a function `SMALL_GROUP_FUNCS` [25] written specially for this library. Such a group can be identified and its catalogue number found by the function `ID_GROUP_FUNCS` [25].

2.13 Layer 11

2.13.1 Determination

The groups of order p^7 have been determined by O'Brien and Vaughan-Lee, see [OV05]. This layer contains these groups for some small primes: $p = 3, 5, 7, 11$.

2.13.2 Storing

The groups are encoded by `PcGroupCode` (**Reference: `PcGroupCode`**) and then the various codes are stored in compressed form. See the groups of order 512 for details. Pc presentations for the groups are produced at run-time for these groups.

2.13.3 Identification

There is no identification function available for the groups in this layer.

2.14 Concluding remarks

The compression of group data for layer 2 has been developed long ago. A first version has been used in the p -group generation algorithm and thus to encode the 2- and 3-groups library of GAP 3. This compression is described in [BE99a].

The remaining compression methods for layers 3 – 8 have been introduced by Hans Ulrich Besche specially for this library. They are designed to store the groups in as little space as possible. For different layers there are different methods, since certain approaches had been successful for certain groups but not for others. Moreover, already published layers have not been changed again and thus useful concepts sometimes have not been applied to earlier layers.

References

- [Bag98] G. Bagnara. La composizione dei gruppi finiti il cui grado è la quinta potenza di un numero primo. *Annali di Matematica Pura ed Applicata*, 1:137–228, 1898. [19](#)
- [BE99a] H. U. Besche and B. Eick. Construction of finite groups. *J. Symbolic Comput.*, 27(4):387–404, 1999. [4](#), [13](#), [17](#), [18](#), [20](#)
- [BE99b] H. U. Besche and B. Eick. The groups of order at most 1000 except 512 and 768. *J. Symbolic Comput.*, 27(4):405–413, 1999. [4](#), [13](#)
- [BE01] H. U. Besche and B. Eick. The groups of order $q^n \cdot p$. *Comm. Algebra*, 29(4):1759–1772, 2001. [4](#), [13](#), [14](#), [18](#)
- [BEO01] H. U. Besche, B. Eick, and E. A. O’Brien. The groups of order at most 2000. *Electron. Res. Announc. Amer. Math. Soc.*, 7:1–4 (electronic), 2001. [4](#)
- [BEO02] H. U. Besche, B. Eick, and E. A. O’Brien. A millennium project: constructing small groups. *Internat. J. Algebra Comput.*, 12(5):623–644, 2002. [4](#), [17](#)
- [Bur21] D. Burrell. On the number of groups of order 1024. *Communications in Algebra*, page 1–3, 2021. [4](#)
- [DE05] H. Dietrich and B. Eick. On the groups of cube-free order. *J. Algebra*, 292(1):122–137, 2005. [4](#), [20](#)
- [EO99a] B. Eick and E. A. O’Brien. Enumerating p -groups. *J. Austral. Math. Soc. Ser. A*, 67(2):191–205, 1999. Group theory. [4](#), [17](#)
- [EO99b] B. Eick and E. A. O’Brien. The groups of order 512. In B. H. Matzat, G.-M. Greuel, and G. Hiss, editors, *Algorithmic algebra and number theory (Heidelberg, 1997)*, page 379–380. Springer, Berlin, 1999. Proceedings of Abschlusstagung des DFG Schwerpunktes Algorithmische Algebra und Zahlentheorie in Heidelberg. [4](#), [17](#)
- [Gir03] B. Girnat. Klassifikation der Gruppen bis zur Ordnung p^5 . Staatsexamensarbeit, TU Braunschweig, Braunschweig, Germany, 2003. [4](#), [19](#)
- [Höl93] O. Hölder. Die gruppen der ordnungen p^3 , pq^2 , pqr , p^4 . *Mathematische Annalen*, 43:301–412, 1893. [13](#), [19](#)
- [Höl95] O. Hölder. Die gruppen mit quadratfreier ordnungszahl. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, page 211–229, 1895. [20](#)

- [JNO90] R. James, M. F. Newman, and E. A. O'Brien. The groups of order 128. *Journal of Algebra*, 129(1):136–158, 1990. [13](#)
- [New77] M. F. Newman. Determination of groups of prime-power order. In R. A. Bryce, J. Cossey, and M. F. Newman, editors, *Group theory (Proc. Miniconf., Australian Nat. Univ., Canberra, 1975)*, volume 573 of *Lecture Notes in Math.*, pages 73–84. Lecture Notes in Math., Vol. 573, Berlin, 1977. Springer. Lecture Notes in Mathematics, Vol. 573. [4](#)
- [NOV04] M. F. Newman, E. A. O'Brien, and M. R. Vaughan-Lee. Groups and nilpotent Lie rings whose order is the sixth power of a prime. *J. Algebra*, 278(1):383–401, 2004. [4](#), [19](#)
- [O'B90] E. A. O'Brien. The p -group generation algorithm. *J. Symbolic Comput.*, 9(5-6):677–698, 1990. Computational group theory, Part 1. [4](#), [13](#), [17](#)
- [O'B91] E. A. O'Brien. The groups of order 256. *J. Algebra*, 143(1):219–235, 1991. [4](#), [13](#)
- [OV05] E. A. O'Brien and M. R. Vaughan-Lee. The groups with order p^7 for odd prime p . *J. Algebra*, 292(1):243–258, 2005. [4](#), [20](#)

Index

AllSmallGroups, [5](#)

Gap3CatalogueIdGroup, [8](#)

IdGap3SolvableGroup, [8](#)

IdGroup, [7](#)

IdGroupsAvailable, [7](#)

IdSmallGroup, [7](#)

IdsOfAllSmallGroups, [7](#)

NrSmallGroups, [6](#)

NumberSmallGroups, [6](#)

NumberSmallGroupsAvailable, [6](#)

OneSmallGroup, [6](#)

SelectSmallGroups, [7](#)

SglPPow package, [4](#)

SmallGroup

- for a pair [order, index], [4](#)
- for group order and index, [4](#)

SmallGroupsAddLayer, [8](#)

SmallGroupsAvailable, [5](#)

SmallGroupsInformation, [8](#)

SMALL_GROUPS_OLD_ORDER, [10](#)

SOTGrps package, [4](#)

ThreeGroup library, [4](#)

TwoGroup library, [4](#)

UnloadSmallGroupsData, [10](#)