TomLib

The GAP Library of Tables of Marks

1.2.9

23 October 2019

Thomas Merkwitz

Liam Naughton

Götz Pfeiffer

Thomas Merkwitz
Email: tmez2525@web.de

Liam Naughton
Email: l.naughton@wlv.ac.uk
Address: Liam Naughton
School of Mathematics and Computer Science
University of Wolverhampton
Wulfruna Street
Wolverhampton
United Kingdom
WV1 1LY

Götz Pfeiffer
Email: goetz.pfeiffer@nuigalway.ie

Copyright

Acknowledgements

This documentation was prepared with the GAPDoc package by Frank Lübeck and Max Neunhöffer.

Contents

Chapter 1

The GAP Table of Marks Library

1.1 Tables Of Marks

The concept of a *Table of Marks* was introduced by W.Burnside in his book "Theory of Groups of Finite Order" [Bur55]. Therefore a table of marks is sometimes called a *Burnside matrix*. The table of marks of a finite group G is a matrix whose rows and columns are labelled by the conjugacy classes of subgroups of G and where for two subgroups H and K the (H,K)–entry is the number of fixed points of K in the transitive action of G on the cosets of H in G. So the table of marks characterizes the set of all permutation representations of G. Moreover, the table of marks gives a compact description of the subgroup lattice of G, since from the numbers of fixed points the numbers of conjugates of a subgroup K contained in a subgroup H can be derived. For small groups the table of marks of G can be constructed directly in GAP by first computing the entire subgroup lattice of G. However, for larger groups this method is unfeasible. The GAP Table of Marks library provides access to several hundred table of marks and their maximal subgroups.

1.2 Installing The Table of Marks Library

Download the archives in your preferred format. Unpack the archives inside the pkg dirctory of your GAP installation. Load the package

Example

```
gap> LoadPackage("tomlib");
true
```

1.3 Contents

TomLib contains several hundred tables of marks. For a complete list of the contents of the library do the following.

Example

```
gap> names:=AllLibTomNames();;
gap> "A5" in names;
true
```

The current version of the tomlib contains the tables of marks of the groups listed below as well as the tables of many of their maximal subgroups and automorphism groups. The Alternating groups A_n

- for $n = 5,6,7,8,9,10,11,12,13$.

The Symmetric groups S_n

- for $n = 4,5,6,7,8,9,10,11,12,13$.

The Linear groups $L_2(n)$ for

- $n = 7,8,11,13,16,17,19,23,25,27,29,31,32,37,41,43,47,49,53$
- $n = 59,61,64,67,71,73,79,81,83,89,97,101,103,107,109,113,121,125$.

along with

- $L_3(4), L_3(3), L_3(5), L_3(7), L_3(9)$
- $L_4(3), L_3(8), L_3(11)$.

The Unitary groups

- $U_3(3), U_4(3), U_3(5), U_3(4), U_3(11), U_3(7), U_3(8)$
- $U_3(9), U_4(2), U_5(2)$

The Sporadic Groups

- $Co_3, HS, McL, He, J_1, J_2, J_3, M_{11}, M_{12}, M_{22}, M_{23}, M_{24}$

The names given to each subgroup are consistent with those used in Robert Wilson's atlas [WWT+] For example if you wish to access the table of marks of the maximal subgroup "$5:4\times A5$" of the Higman-Sims group do the following:

Example

```
gap> TableOfMarks("5:4xA5");
TableOfMarks( "5:4xA5" )
```

1.4 Administrative Functions

Here we document some of the administrative facilities for the the GAP library of tables of marks.

1.4.1 LIBTOMKNOWN

▷ `LIBTOMKNOWN` (global variable)

`LIBTOMKNOWN` is a record that controls the loading of data files of the GAP library of tables of marks. It has the following components.

`ACTUAL`
: the name of the file to be read at the moment (set by `SetActualLibFileName`),

`LOADSTATUS`
: a record whose components are names of files in the library of tables of marks, with value a list whose first entry is one of `"loaded"`, `"unloaded"`, `"userloaded"` and whose second entry is an integer that controls when the corresponding tables of marks can be removed from GAP,

`MAX` GAP remembers at most `MAX` of the previously loaded library files (the default value is 10),

`UNLOAD`
is it allowed to remove previously loaded library files (is set to `true` by default),

`STDGEN`
a list describing standard generators of almost simple groups in the table of marks library.

Additionally the names of the files already loaded occur as components of `LIBTOMKNOWN`; the corresponding values are given by the data of the files.

1.4.2 IsLibTomRep

▷ `IsLibTomRep(`*`obj`*`)` (Representation)

Library tables of marks have their own representation. `IsLibTomRep` tests if *`obj`* is a library representation.

1.4.3 TableOfMarksFromLibrary

▷ `TableOfMarksFromLibrary(`*`string`*`)` (function)
Returns: the table of marks with name *`string`*.

1.4.4 ConvertToLibTom

▷ `ConvertToLibTom(`*`record`*`)` (function)

`ConvertToLibTom` converts a record with components from `TableOfMarksComponents` into a library table of marks object with the corresponding attribute values set.

1.4.5 SetActualLibFileName

▷ `SetActualLibFileName(`*`filename`*`)` (function)

Sets the file name for *`filename`*.

1.4.6 LIBTOM

▷ `LIBTOM(`*`arg`*`)` (function)
▷ `AFLT(`*`source, destination, fusion`*`)` (function)
▷ `ACLT(`*`identifier, component, value`*`)` (function)

The library format of a library table of marks is a call to the function `LIBTOM`, with the arguments sorted as in `TableOfMarksComponents` .

`AFLT` adds a fusion map *`value`* from the table of marks with name *`source`* to the table of marks with name *`destination`*. The fusion map is a list of positive integers, storing at position *i* the position of the class in *`destination`* that contains the subgroups in the *i*-th class of *`source`*.

`ACLT` adds the value *`value`* of a component with name *`component`* to the table of marks with identifier value *`identifier`* in the library of tables of marks.

1.4.7 AllLibTomNames

▷ `AllLibTomNames()` (function)

Returns: a list containing all names of available library tables of marks.

1.4.8 NamesLibTom

▷ `NamesLibTom(`*`string`*`)` (attribute)
▷ `NamesLibTom(`*`tom`*`)` (attribute)

Returns: all names of the library table *tom* or of the library table with name *string*

1.4.9 NotifiedFusionsOfLibTom

▷ `NotifiedFusionsOfLibTom(`*`tom`*`)` (attribute)
▷ `NotifiedFusionsOfLibTom(`*`string`*`)` (attribute)
▷ `FusionsOfLibTom(`*`tom`*`)` (attribute)
▷ `FusionsOfLibTom(`*`string`*`)` (attribute)

Are there any fusions from the library table of marks *tom* or the table of marks with name *string* into other library tables marks?

`NotifiedFusionsOfLibTom` returns the names of all such tables of marks. `FusionsOfLibTom` returns the complete fusion maps. For that the corresponding library file has to be loaded.

1.4.10 NotifiedFusionsToLibTom

▷ `NotifiedFusionsToLibTom(`*`tom`*`)` (attribute)
▷ `NotifiedFusionsToLibTom(`*`string`*`)` (attribute)
▷ `FusionsToLibTom(`*`tom`*`)` (attribute)
▷ `FusionsToLibTom(`*`string`*`)` (attribute)

Are there any fusions from other library table of marks to *tom* or the table of marks with name *string*.

`NotifiedFusionsToLibTom` returns the names of all such tables of marks. `FusionsToLibTom` returns the complete fusion maps. For that, the correponding library files have to be loaded.

1.4.11 UnloadTableOfMarksData

▷ `UnloadTableOfMarksData()` (function)

`UnloadTableOfMarksData` clears the cache of tables of marks. This can be used to free up to several hundred megabytes of space in the current GAP session.

1.5 Standard Generators of Groups

An s-tuple of *standard generators* of a given group G is a vector $(g_1, g_2, \ldots, g_s)$ of elements $g_i \in G$ satisfying certain conditions (depending on the isomorphism type of G) such that

1. $\langle g_1, g_2, \ldots, g_s \rangle = G$ and

2. the vector is unique up to automorphisms of G, i.e., for two vectors $(g_1, g_2, \ldots, g_s)$ and $(h_1, h_2, \ldots, h_s)$ of standard generators, the map $g_i \mapsto h_i$ extends to an automorphism of G.

For details about standard generators, see [Wil96].

1.5.1 StandardGeneratorsInfo (for groups)

▷ `StandardGeneratorsInfo(G)` (attribute)

When called with the group `G`, `StandardGeneratorsInfo` returns a list of records with at least one of the components `script` and `description`. Each such record defines *standard generators* of groups isomorphic to `G`, the i-th record is referred to as the i-th set of standard generators for such groups. The value of `script` is a dense list of lists, each encoding a command that has one of the following forms.

A *definition* $[i, n, k]$ **or** $[i, n]$
means to search for an element of order n, and to take its k-th power as candidate for the i-th standard generator (the default for k is 1),

a *relation* $[i_1, k_1, i_2, k_2, \ldots, i_m, k_m, n]$ **with** $m > 1$
means a check whether the element $g_{i_1}^{k_1} g_{i_2}^{k_2} \cdots g_{i_m}^{k_m}$ has order n; if g_j occurs then of course the j-th generator must have been defined before,

a *relation* $[[i_1, i_2, \ldots, i_m], \texttt{slp}, n]$
means a check whether the result of the straight line program `slp` (see (**Reference: Straight Line Programs**)) applied to the candidates $g_{i_1}, g_{i_2}, \ldots, g_{i_m}$ has order n, where the candidates g_j for the j-th standard generators must have been defined before,

a *condition* $[[i_1, k_1, i_2, k_2, \ldots, i_m, k_m], f, v]$
means a check whether the GAP function in the global list `StandardGeneratorsFunctions` (1.5.3) that is followed by the list f of strings returns the value v when it is called with G and $g_{i_1}^{k_1} g_{i_2}^{k_2} \cdots g_{i_m}^{k_m}$.

Optional components of the returned records are

`generators`
a string of names of the standard generators,

`description`
a string describing the `script` information in human readable form, in terms of the `generators` value,

`classnames`
a list of strings, the i-th entry being the name of the conjugacy class containing the i-th standard generator, according to the Atlas character table of the group (see `ClassNames` (**Reference: ClassNames**)), and

`ATLAS`
a boolean; `true` means that the standard generators coincide with those defined in Rob Wilson's Atlas of Group Representations (see [WWT+]), and `false` means that this property is not guaranteed.

`standardization`
: a positive integer *i*; Whenever `ATLAS` returns `true` the value of *i* means that the generators stored in the group are standard generators w.r.t. standardization *i*, in the sense of Rob Wilson's Atlas of Group Representations.

There is no default method for an arbitrary isomorphism type, since in general the definition of standard generators is not obvious.

The function `StandardGeneratorsOfGroup` (1.5.5) can be used to find standard generators of a given group isomorphic to *G*.

The `generators` and `description` values, if not known, can be computed by `HumanReadableDefinition` (1.5.2).

Example

```
gap> StandardGeneratorsInfo( TableOfMarks( "L3(3)" ) );
[ rec( ATLAS := true,
    description := "|a|=2, |b|=3, |C(b)|=9, |ab|=13, |ababb|=4",
    generators := "a, b",
    script := [ [ 1, 2 ], [ 2, 3 ], [ [ 2, 1 ], [ "|C(",, ")|" ], 9 ],
        [ 1, 1, 2, 1, 13 ], [ 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 4 ] ],
    standardization := 1 ) ]
```

1.5.2 HumanReadableDefinition

▷ `HumanReadableDefinition(`*info*`)` (function)

▷ `ScriptFromString(`*string*`)` (function)

Let *info* be a record that is valid as value of `StandardGeneratorsInfo` (1.5.1). `HumanReadableDefinition` returns a string that describes the definition of standard generators given by the `script` component of *info* in human readable form. The names of the generators are taken from the `generators` component (default names `"a"`, `"b"` etc. are computed if necessary), and the result is stored in the `description` component.

`ScriptFromString` does the converse of `HumanReadableDefinition`, i.e., it takes a string *string* as returned by `HumanReadableDefinition`, and returns a corresponding `script` list.

If "condition" lines occur in the script (see `StandardGeneratorsInfo` (1.5.1)) then the functions that occur must be contained in `StandardGeneratorsFunctions` (1.5.3).

Example

```
gap> scr:= ScriptFromString( "|a|=2, |b|=3, |C(b)|=9, |ab|=13, |ababb|=4" );
[ [ 1, 2 ], [ 2, 3 ], [ [ 2, 1 ], [ "|C(",, ")|" ], 9 ], [ 1, 1, 2, 1, 13 ],
  [ 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 4 ] ]
gap> info:= rec( script:= scr );
rec( script := [ [ 1, 2 ], [ 2, 3 ], [ [ 2, 1 ], [ "|C(",, ")|" ], 9 ],
      [ 1, 1, 2, 1, 13 ], [ 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 4 ] ] )
gap> HumanReadableDefinition( info );
"|a|=2, |b|=3, |C(b)|=9, |ab|=13, |ababb|=4"
gap> info;
rec( description := "|a|=2, |b|=3, |C(b)|=9, |ab|=13, |ababb|=4",
  generators := "a, b",
  script := [ [ 1, 2 ], [ 2, 3 ], [ [ 2, 1 ], [ "|C(",, ")|" ], 9 ],
      [ 1, 1, 2, 1, 13 ], [ 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 4 ] ] )
```

1.5.3 StandardGeneratorsFunctions

▷ StandardGeneratorsFunctions (global variable)

StandardGeneratorsFunctions is a list of even length. At position $2i-1$, a function of two arguments is stored, which are expected to be a group and a group element. At position $2i$ a list of strings is stored such that first inserting a generator name in all holes and then forming the concatenation yields a string that describes the function at the previous position; this string must contain the generator enclosed in round brackets (and).

This list is used by the functions StandardGeneratorsInfo (1.5.1)), HumanReadableDefinition (1.5.2), and ScriptFromString (1.5.2). Note that the lists at even positions must be pairwise different.

Example

```
gap> StandardGeneratorsFunctions{ [ 1, 2 ] };
[ function( G, g ) ... end, [ "|C(",, ")|" ] ]
```

1.5.4 IsStandardGeneratorsOfGroup

▷ IsStandardGeneratorsOfGroup(*info*, *G*, *gens*) (function)

Let *info* be a record that is valid as value of StandardGeneratorsInfo (1.5.1), *G* a group, and *gens* a list of generators for *G*. In this case, IsStandardGeneratorsOfGroup returns true if *gens* satisfies the conditions of the script component of *info*, and false otherwise.

Note that the result true means that *gens* is a list of standard generators for *G* only if *G* has the isomorphism type for which *info* describes standard generators.

1.5.5 StandardGeneratorsOfGroup

▷ StandardGeneratorsOfGroup(*info*, *G*[, *randfunc*]) (function)

Let *info* be a record that is valid as value of StandardGeneratorsInfo (1.5.1), and *G* a group of the isomorphism type for which *info* describes standard generators. In this case, StandardGeneratorsOfGroup returns a list of standard generators of *G*, see Section 1.5.

The optional argument *randfunc* must be a function that returns an element of *G* when called with *G*; the default is PseudoRandom (**Reference: PseudoRandom**).

In each call to StandardGeneratorsOfGroup, the script component of *info* is scanned line by line. *randfunc* is used to find an element of the prescribed order whenever a definition line is met, and for the relation and condition lines in the script list, the current generator candidates are checked; if a condition is not fulfilled, all candidates are thrown away, and the procedure starts again with the first line. When the conditions are fulfilled after processing the last line of the script list, the standard generators are returned.

Note that if *G* has the wrong isomorphism type then StandardGeneratorsOfGroup returns a list of elements in *G* that satisfy the conditions of the script component of *info* if such elements exist, and does not terminate otherwise. In the former case, obviously the returned elements need not be standard generators of *G*.

Example

```
gap> a5:= AlternatingGroup( 5 );
Alt( [ 1 .. 5 ] )
```

```
gap> info:= StandardGeneratorsInfo( TableOfMarks( "A5" ) )[1];
rec( ATLAS := true, description := "|a|=2, |b|=3, |ab|=5",
generators := "a, b", script := [ [ 1, 2 ], [ 2, 3 ], [ 1, 1, 2, 1, 5 ] ],
standardization := 1 )
gap> IsStandardGeneratorsOfGroup( info, a5, [ (1,3)(2,4), (3,4,5) ] );
true
gap> IsStandardGeneratorsOfGroup( info, a5, [ (1,3)(2,4), (1,2,3) ] );
false
gap> s5:= SymmetricGroup( 5 );;
gap> RepresentativeAction( s5, [ (1,3)(2,4), (3,4,5) ],
>        StandardGeneratorsOfGroup( info, a5 ), OnPairs ) <> fail;
true
```

1.5.6 StandardGeneratorsInfo (for tables of marks)

▷ `StandardGeneratorsInfo(`*tom*`)` (attribute)

For a table of marks *tom*, a stored value of `StandardGeneratorsInfo` equals the value of this attribute for the underlying group (see `UnderlyingGroup` (**Reference: UnderlyingGroup for tables of marks**)) of *tom*, cf. Section 1.5.

In this case, the `GeneratorsOfGroup` (**Reference: GeneratorsOfGroup**) value of the underlying group G of *tom* is assumed to be in fact a list of standard generators for G; So one should be careful when setting the `StandardGeneratorsInfo` value by hand.

There is no default method to compute the `StandardGeneratorsInfo` value of a table of marks if it is not yet stored.

Example

```
gap> a5:=TableOfMarks("A5");
gap> std:= StandardGeneratorsInfo( a5 );
[ rec( ATLAS := true, description := "|a|=2, |b|=3, |ab|=5",
    generators := "a, b", script := [ [ 1, 2 ], [ 2, 3 ], [ 1, 1, 2, 1, 5 ]
      ], standardization := 1 ) ]
gap> # Now find standard generators of an isomorphic group.
gap> g:= SL(2,4);;
gap> repeat
>   x:= PseudoRandom( g );
> until Order( x ) = 2;
gap> repeat
>   y:= PseudoRandom( g );
> until Order( y ) = 3 and Order( x*y ) = 5;
gap> # Compute a representative w.r.t. these generators.
gap> RepresentativeTomByGenerators( a5, 4, [ x, y ] );
Group([ [ [ 0*Z(2), Z(2)^0 ], [ Z(2)^0, 0*Z(2) ] ],
  [ [ Z(2^2), Z(2^2)^2 ], [ Z(2^2)^2, Z(2^2) ] ] ])
gap> # Show that the new generators are really good.
gap> grp:= UnderlyingGroup( a5 );;
gap> iso:= GroupGeneralMappingByImages( grp, g,
>              GeneratorsOfGroup( grp ), [ x, y ] );;
gap> IsGroupHomomorphism( iso );
true
gap> IsBijective( iso );
```

```
true
```

References

[Bur55] W. Burnside. *Theory of groups of finite order*. Dover Publications Inc., New York, 1955. Unabridged republication of the second edition, published in 1911. 4

[Wil96] R. A. Wilson. Standard generators for sporadic simple groups. *J. Algebra*, 184(2):505–515, 1996. 8

[WWT+] R. A. Wilson, P. Walsh, J. Tripp, I. Suleiman, R. A. Parker, S. P. Norton, S. Nickerson, S. Linton, J. Bray, and R. Abbott. ATLAS of Finite Group Representations. http://brauer.maths.qmul.ac.uk/Atlas/. 5, 8

Index