The Group Presentations Library.

The Group Presentations Library (GPL) presents a new way to store and use presentations and words generating subgroups in a given presentation. The presentations and the words are stored as strings in a human readible way. I tried to develop and document the package in such a way that it is easy to add new words and presentations.

Presentations and short words for subgroups for sporadic groups.

The Group Presentations Library contains already 27 presentations for sporadic groups. Numerous subgroups can be created with the stored short words in the generators of the given presentations. The first chapter of this documentation describes the functions that are programmed to use the information available in the GPL library file. Besides, it will give an overview of the available presentations and it will tell for which subgroups in the available presentations words are given in the GPL library file. Several new short words that generate subgroups of sporadic groups were found by the author. Short words found by Martin Schönert were added as well to the list of words and presentations that can be found in [PS97]. This last source of information formed the first basis for this library file.

Storing presentations and words.

The code that builds the presentations and the subgroups of a group in a given presentation from the GPL library file is quite general. It is therefore possible to add other presentations and words in future releases easily. If one wishes to use a new presentation on a one-off basis, one can use the functions described in chapter 2. If a presentation or a set of presentations is needed more regularly, we could consider to add these presentations in future releases. Please contact me via email or other means. Of course it is possible to add presentations and words to your personal copy of the library file by just following the format conventions described in the first chapter and in the library file itself.

Finitely presented groups versus string presentations.

The second chapter describes an alternative approach to building finitely presented groups inside GAP. The standard approach in GAP is to use a set of abstract generators subject to a set of relations. When the relations, which are words in the abstract generators, are complicated, it is sometimes more convenient to represent some of the relations in the form of a so-called Coxeter graph. In our approach, functions are introduced that make it possible to enter generators and implicit relations via Coxeter graphs and explicit relations using strings. Functions that convert a finitely presented group in such a string presentation are added as well.

I hereby thank Anton Cox and Julian Gilbey for their comments. Almost last but not least I would like to give special thanks to my supervisor, Dr. L.H. Soicher who showed me many techniques for working with sporadic groups and GAP, and who carefully read this documentation and used procedures on an early stage.

Finally, I would like to thank the European Union, which enabled me to do this work, done during a period of eight months at Queen Mary and Westfield College, London, via an HCM grant in Computational Group Theory.

Roderik Lindenbergh
lindenbergh@math.uu.nl

1. Presentations of Sporadic Groups

This chapter describes procedures to access the information stored in the **Group Presentations Library**. Currently, all presentations stored are presentations of **Sporadic Groups**. We consider a sporadic group G to be a sporadic simple group or an automorphism group of a sporadic simple group. At the moment, the library contains 19 presentations of simple sporadic groups and 8 presentations of non-simple sporadic groups. Some of these presentations present isomorphic groups. All of the presentations currently available can be found in [PS97]. Especially, presentations are present for the seven smallest sporadic simple groups and their automorphism groups. For every **maximal** subgroup in these eleven groups, short words are given that generate the maximal subgroup in the given presentation. Maximal is considered here as up to the action of the automorphism group of the group presented.

Information about some particular sporadic group can be obtained by using its name. The names that appear in the following example are the names that are used to identify the sporadic groups. These names correspond with the names used in GAP to identify character tables.

```
gap> Read("gpl.g");
gap> av := AvailableGroupPresentationsGPL();
[ GroupPresentationGPL( "M11", 1 ), GroupPresentationGPL( "M12", 1 ),
  GroupPresentationGPL( "M12:2", 1 ), GroupPresentationGPL( "J1", 1 ),
  GroupPresentationGPL( "M22", 1 ), GroupPresentationGPL( "M22:2", 1 ),
  GroupPresentationGPL( "J2", 1 ), GroupPresentationGPL( "J2:2", 1 ),
  GroupPresentationGPL( "M23", 1 ), GroupPresentationGPL( "HS", 1 ),
  GroupPresentationGPL( "HS:2", 1 ), GroupPresentationGPL( "M24", 1 ),
  GroupPresentationGPL( "M24", 2 ), GroupPresentationGPL( "McL", 1 ),
  GroupPresentationGPL( "He", 1 ), GroupPresentationGPL( "He:2", 1 ),
  GroupPresentationGPL( "Suz", 1 ), GroupPresentationGPL( "Suz:2", 1 ),
  GroupPresentationGPL( "ON", 1 ), GroupPresentationGPL( "Co3", 1 ),
  GroupPresentationGPL( "Co2", 1 ), GroupPresentationGPL( "Fi22", 1 ),
  GroupPresentationGPL( "Fi22:2", 1 ),
  GroupPresentationGPL( "Fi23", 1 ), GroupPresentationGPL( "Co1", 1 ),
  GroupPresentationGPL( "Co1", 2 ),
  GroupPresentationGPL( "F3+:2", 1 ) ]
gap> for i in [1..27] do InitializeGroupPresentationGPL( av[i] ); od;
gap> List(av, p -> AvailableSubgroupsGPL( p ));
[ [ A6.2_3, L2(11), A6, 3^2:Q8.2, A5.2, 3^2:8, 2.S4 ],
  [ M11, A6.2^2, PGL2(9), S6, L2(11), L2(11), 3^2.2.S4, 2xS5, M8.S4,
    4^2:D12, A4xS3 ],
  [ L2(11):2, L2(11):2, (2^2xA5):2, 3_{+}^{1+2}:D8, (M8.S4).2,
    (4^2:D12).2, S4xS3, S5 ],
  [ L2(11), 2^3.7.3, 2xA5, 19:6, 11:10, D6xD10, 7:6 ],
```

```
[ L3(4), 2^4:a6, A7, 2^4:s5, 2^4:L2(5), 2^3:sl(3,2), A6.2_3,
  L2(11) ],
[ L3(4):2_2, 2^4:s6, 2^5:s5, 2^5:L2(5), 2^3:sl(3,2)x2, A6.2^2,
  L2(11):2 ],
[ U3(3), 3.A6.2_2, 2^1+4b:a5, 2^2+4.3xs3, a4xa5, a5xd10, L3(2).2,
  5^2:d12, A5 ],
[ U_3(3):2, 3.A6.2^2, 2^1+4b.S5, 2^2+4:3xS3.2, (A4xA5):2, (A5xD10).2,
  L3(2):2x2, 5^2:(4xS3), S5 ],
[ M22, L3(4).2_2, 2^4:a7, A8, M11, 2^4:(3xa5):2, 23:11 ],
[ M22, U3(5).2, U3(5), L3(4).2_1, A8.2, 2^4.s6, 4^3:psl(3,2), M11,
  4.2^4:s5, 2xa6.2^2, 5:4xa5 ],
[ M22:2, L3(4).2_2, A8.2x2, 2^5.s6, 4^3:(psl(3,2)x2), 2_+^(1+6):s5,
  (2xa6.2^2).2, 5_+^{1+2}:[2^5], 5:4xs5 ],
[ M23, M22:2, 2^4:A8, L3(4):S3 ],
[ M12:2, 2^6:3:S6, M12, 2^6:3:A6, 2^6:(L3(2)xS3) ],
[ U4(3), M22, U3(5), 3^{1+4}_+:2.S5 ], [ S4(4):2 ], [ S4(4):4 ],
[ G2(4), 3_2.U4(3):2_3 ], [ 3.U4(3).(2^2) ], [ L3(7):2 ],
[ McL:2, McL, HS, M23, 2.S6(2) ],
[ U6(2):2, U6(2), 2^{10}:M22:2, McL, 2^{1+8}:S6(2), HS:2 ],
[ 2.U6(2), O7(3), 2^{10}:M22, 2^6:S6(2) ], [ O8^+(2):S3x2 ],
[ 2.Fi22, O8^+(3):S3, O8^+(3):3, O8^+(3):2 ], [ Co2, 2^{11}:M24,
  Co3 ],
[ 2x3.Suz:2 ], [ Fi23x2 ] ]
```

Note that to be able to work with a presentation it has to be initialized first using the procedure `InitializeGroupPresentationGPL`. The subgroups are ordered according to their size. In case of equal size, maximal subgroups are listed before non-maximal ones.

The first section in this chapter describes the function `GroupRecordGPL` that returns a record containing all available presentations of a group. The next sections describe functions that initialize and access the presentations themselves. The final sections describe functions that extract certain subgroups in a given presentation from the library.

1.1 GroupRecordGPL

`GroupRecordGPL(` *grname* `)`

`GroupRecordGPL` returns a record that contains information about the underlying group G with name *grname* together with all available presentations for G.

```
gap> GroupRecordGPL( "J2");
GroupRecordGPL( "J2" )
```

Such a record contains the following components:

`.info` contains a list of two entries: the name of the group and its size.

`.presentations` contains the number of different presentations of G. If this number n is bigger than zero then the components `.1` ,..., `.n` contain these presentations.

`.operations` contains a function that takes care of the output to the screen when `GroupRecordGPL` is called.

1.2 GroupPresentationGPL

`GroupPresentationGPL(` *grname*`,` *number* `)`

`GroupPresentationGPL` returns a record containing the presentation with number *number* of the group with name *grname*. *grname* is expected to be a string.

```
gap> mcl := GroupPresentationGPL( "McL", 1);
GroupPresentationGPL( "McL", 1 )
```

This record contains the following components:

`.name` contains a descriptive name for the underlying group.

```
gap> mcl.name;
"McL"
```

`.number` is the same as the number *number* above.

```
gap> mcl.number;
1
```

`.stringPresentation` contains presentation number *number* of group *name* in string format. The string presentation is a list of maximal four entries. If all four entries are bound they will have the following contents:

1. A string representing a Coxeter graph.
2. A string representing generators that are not introduced by the Coxeter graph.
3. A string representing relators, necessary for the definition of the presentation.
4. A string representing relators, not necessary for the definition of the presentation, but helpful for a coset enumeration process.

It is not required that all entries are bound, but at least one of the entries *1* or *2* should be. It is important that an entry is on that position of the string presentation list as is indicated by the scheme above. Refer to chapter 2 for information on the formats required for the mentioned strings.

```
gap> mcl.stringPresentation;
[ "a3b5c3d3f6e,a4e3c4f",, "a=(cf)^2,b=(ef)^3,(eab)^3,(bce)^5,(aecd)^4,
  (cef)^7", "(bcd)^5" ]
```

`.subgroups` when present, contains a list of strings defining subgroups of the group in the given presentation. The first entry contains a descriptive name of the subgroup. The names correspond with the names used by GAP to identify character tables as much as possible. The second entry consist of a string representing words, generating the subgroup. The third entry is of the form `"_A**"`. This string indicates the source of the words. (`A` stands for Author, see the source file for definitions of the used abbrevations for names of authors.) The existence of a string `"_MT"` (resp. `"_MF"`) indicates that the subgroup is maximal (resp. non-maximal). Eventually, some other common names for the subgroup are added.

```
gap> mcl.subgroups;
[ [ "U4(3)", "cef,dfcd", "_APS", "_MT" ],
  [ "M22", "a,b,c,d,e", "_APS", "_MT" ],
```

```
[ "U3(5)", "b,(de)^(cbac),acdcbeace,(cdfbeaecd)^3", "_APS", "_MT" ],
[ "3^{1+4}_+:2.S5", "cdfbeaecd,cbadecbcbadcecbaecd", "_APS", "_MT" ] ]
```

`.text` describes the source of the presentation.

```
gap> mcl.text;
"origin: Low Rank Representations and Graphs for Sporadic Groups"
```

`.operations` contains a function that takes care of the output to the screen when `GroupPresentationGPL` is called.

1.3 InitializeGroupPresentationGPL

`InitializeGroupPresentationGPL(` *pres* `)`

`InitializeGroupPresentationGPL` initializes the GPL presentation *pres*. This function returns nothing but adds one or two components to the GPL record *pres*:

`.group` contains the GAP group built from the presentation.

```
gap> InitializeGroupPresentationGPL( mcl );
gap> mcl.group;
McL
```

`.stringGenerators`, when bound, contains a string that gives the correspondance between the generators used in the finitely presented GAP group and the generators used in the string presentation. The letter on position i of the string corresponds with the generator *group.i*.

```
gap> mcl.stringGenerators;
"abcdfe"
```

1.4 GroupPresentationsGPL

`GroupPresentationsGPL(` *grname* `)`

`GroupPresentationsGPL(` *grname* `)` returns a list of all presentations of the group with name *grname*. *grname* is expected to be a string.

```
gap> m24pres := GroupPresentationsGPL( "M24");
[ GroupPresentationGPL( "M24", 1 ), GroupPresentationGPL( "M24", 2 ) ]
```

1.5 AvailableGroupPresentationsGPL

`AvailableGroupPresentationsGPL( )`

`AvailableGroupPresentationsGPL()` returns a list of all available presentations in the GPL library file. See the preamble for the current list.

1.6 SubgroupGPL

`SubgroupGPL(` *pres*, *number* `)`
`SubgroupGPL(` *pres*, *sgrname* `)`

`SubgroupGPL(` *pres*, *number* `)` returns the subgroup labeled by entry number *number* in the list *pres*`.subgroups`. *pres* must be a presentation. Note that a presentation is **not** a group:

```
gap> m12 := GroupPresentationGPL( "M12", 1);
GroupPresentationGPL( "M12", 1 )
gap> InitializeGroupPresentationGPL( m12 );
gap> m11 := SubgroupGPL( m12, 1);
M11
gap>  Index(m12, m11);
Error, <G> must be a group in
Index( m12, m11 ) called from
main loop
brk>
gap> gm12 := m12.group;
M12
gap> Index(gm12, m11);
12
```

`SubgroupGPL(` *pres*, *sgrname* `)` returns the subgroup of name *sgrname* defined in the list *pres*`.subgroups` :

```
gap> m12 := GroupPresentationGPL( "M12", 1);
GroupPresentationGPL( "M12", 1 )
gap> InitializeGroupPresentationGPL( m12 );
gap> s6 := SubgroupGPL(m12, "S6");
S6
gap> gm12 := m12.group;
gap> opsm12s6 :=  OperationCosetsFpGroup( gm12, s6 );;
gap> Size(opsm12s6);
95040
```

1.7 AvailableSubgroupsGPL

`AvailableSubgroupsGPL(` *pres* `)`

`AvailableSubgroupGPL(` *pres* `)` returns a list of all the subgroups defined in the list *pres*`.subgroups`, where *pres* must be a presentation. Note that in general not a complete list of subgroups of the group indicated by *name* will be returned.

```
gap> hig := GroupPresentationGPL( "HS", 1);
gap> InitializeGroupPresentationGPL( hig );
gap> AvailableSubgroupsGPL( hig );
[ M22, U3(5).2, U3(5), L3(4).2_1, A8.2, 2^4.s6, 4^3:psl(3,2), M11,
  4.2^4:s5, 2xa6.2^2, 5:4xa5 ]
```

1.8 AvailableMaximalSubgroupsGPL

`AvailableMaximalSubgroupsGPL(` *name* `)`

`AvailableMaximalSubgroupGPL(` *name* `)` works the same as `SubgroupGPL(` *name* `)` but will only return the listed maximal subgroups of the group indicated by *name*. Again, this will in general not be a complete list of maximal subgroups. For the available presentations of the sporadic simple groups from M_{11} up to HS and their automorphism groups, however, words for every maximal subgroup are present.

```
gap>  m11 := GroupPresentationGPL( "M11",1);;
gap> InitializeGroupPresentationGPL( m11 );
gap> AvailableSubgroupsGPL( m11 );
[ A6.2_3, L2(11), A6, 3^2:Q8.2, A5.2, 3^2:8, 2.S4 ]
gap> AvailableMaximalSubgroupsGPL( m11 );
[ A6.2_3, L2(11), 3^2:Q8.2, A5.2, 2.S4 ]
```

2. Finitely Presented Groups Via Strings

A **finitely presented group** is a group generated by a finite set of **abstract generators** subject only to a finite set of **relations** that these generators must satisfy.

A finitely presented group can be constructed in GAP as described in the chapter **Finitely Presented Groups** of the GAP manual, see [GAP]. Here an alternative method of construction is described, closer to notation used for presentations in [PS97] and in [ATLAS]. One can construct a finitely presented group via a **Coxeter graph**, which introduces generators and describes relations. If necessary extra generators or relations can be added. In this chapter procedures are described that create finitely presented groups via strings representing graphs, explicit generators and relations. Two procedures to form such a **string presentation** from an arbitrary finitely presented group are added as well.

For example from a string presentation to a finitely presented group:

```
gap>  cox := "A3b5c3d3e4c";
"A3b5c3d3e4c"
gap>  rel := "A=(ce)^2";
"A=(ce)^2"
gap> extrarel := "_PCE,(Abc)^5,(bcd)^5";
"_PCE,(Abc)^5,(bcd)^5";
gap> m11 :=  FpGroupViaStrings( cox, rel, extrarel );
## The map is: "Abcde"
Group( f.1, f.2, f.3, f.4, f.5 )
gap> map := "Abcde";
gap> m10 := SubgroupViaStrings( m11, "b,c,d,AbcbAe", map );
Subgroup( Group( f.1, f.2, f.3, f.4, f.5 ),
[ f.2, f.3, f.4, f.1*f.2*f.3*f.2*f.1*f.5 ] )
gap> Index(m11,m10);
11
```

From a finitely presented group to a string presentation:

```
gap> F5 := FreeGroup( "x1", "x2", "x3", "x4", "x5" );;
gap> E1 := F5 / [ F5.1^2, F5.2^2, F5.3^2, F5.4^2, F5.5^2,
> ( F5.1 * F5.3 )^2, ( F5.2 * F5.4 )^2, ( F5.1 * F5.2 )^3,
> ( F5.2 * F5.3 )^3, ( F5.3 * F5.4 )^3, ( F5.4 * F5.1 )^3,
> ( F5.1 * F5.5 )^3, ( F5.2 * F5.5 )^2, ( F5.3 * F5.5 )^3,
> ( F5.4 * F5.5 )^2,
> ( F5.1 * F5.2 * F5.3 * F5.4 * F5.3 * F5.2 )^2 ];;
gap> CoxeterStringPresentation( E1 );
[ "A3B3C3D3A3E3C",, "(ABCDCB)^2" ]
```

The first section in this chapter describes **Coxeter graphs**. The next sections describe how to construct finitely presented groups and subgroups using strings representing Coxeter graphs, generators and words. The final sections describe procedures to create string presentations from arbitrary finitely presented groups.

2.1 Coxeter graphs

Each node x of a Coxeter graph specifies a generator x and the relation $1 = x^2$. If two distinct nodes x, y in a Coxeter graph are joined by an edge with label m, where m is a positive integer, then this denotes the relation $1 = (xy)^m$. All the edges in our Coxeter graphs are labelled. If two distinct nodes x, y in a Coxeter graph are not joined by an edge, then this denotes the relation $1 = (xy)^2$. We specify a Coxeter graph Φ by a set of paths in Φ, each containing at least one edge, and which together contain all the edges of Φ. A typical path is of the form

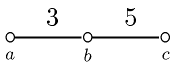
$$xmynz\ldots,$$

which means that x is joined to y with an edge labelled m, y is joined to z with an edge labelled n, and so on.

For example, the graph specified by the two paths

$$a3b3c3d3a3c, b3d$$

is the complete graph on a, b, c, d, with all edge-labels 3. The Coxeter graph



is denoted by

$$a3b5c,$$

which also denotes the presentation

$$\langle a, b, c \mid 1 = a^2 = b^2 = c^2 = (ab)^3 = (bc)^5 = (ac)^2 \rangle$$

for $2 \times A_5$.

Note that we demand a Coxeter graph to have at least one edge.

2.2 FpGroupViaStrings

```
FpGroupViaStrings( cox )
FpGroupViaStrings( cox, relstring )
FpGroupViaStrings( cox, genstring, relstring )
FpGroupViaStrings( genstring, relstring )
FpGroupViaStrings( ... , extrarelstring )
FpGroupViaStrings( [ ... ] )
```

`FpGroupViaStrings` returns the finitely presented group generated by the generators present in *cox* and/or in *genstring*, and subject to the relations defined by *cox* and/or by *relstring*.

The Coxeter graph *cox* is expected to be represented as a string where every part between commas corresponds to a path in the Coxeter graph in the obvious way: for example, the graph specified by the two paths $a3b3c3d3a3c, b3d$ should be represented as

```
gap> cox := "a3b3c3d3a3c,b3d";;
```

Generators can be given only by single capital or small letters. A small letter denotes a different generator from a capital letter. This limits the number of different generators that can be used in a string presentation to 52. When all relations are present in the Coxeter graph one can build the corresponding group:

```
gap> 2dira5 := FpGroupViaStrings( "K3L5M" );
## The map is: "KLM"
Group( f.1, f.2, f.3 )
gap> 2dira5.relators;
[ f.1*f.2*f.1*f.2*f.1*f.2, f.2*f.3*f.2*f.3*f.2*f.3*f.2*f.3*f.2*f.3,
  f.1^2, f.2^2, f.3^2, f.1*f.3*f.1*f.3 ]
```

Besides of creating and returning the group, `FpGroupViaStrings` will print out a message that indicates how the string generators correspond with the generators of the finitely presented group. The first string generator found while analyzing the string presentation will correspond with the first generator of the finitely presented group, etcetera. This map has to be used for building subgroups via strings, see section 2.3. When generators are entered in a different order this will only result in a relabeling of the underlying generators used by GAP.

Generators can also be introduced in an explicit way: a string of letters like `"a,b,c"` will be interpreted as a list of generators. Note that for generators introduced in this way it is necessary to enter explicitly an order relation, as nothing will be assumed about the order of a generator.

```
gap> gen := "K,L,M";
"K,L,M"
gap> rel := "K^2,L^2,M^2,(KL)^3,(LM)^5,(KM)^2";
"K^2,L^2,M^2,(KL)^3,(LM)^5,(KM)^2"
gap> alt2dira5 := FpGroupViaStrings( gen, rel );
## The map is: "KLM"
Group( f.1, f.2, f.3 )
gap> alt2dira5.relators;
[ f.1^2, f.2^2, f.3^2, f.1*f.2*f.1*f.2*f.1*f.2,
f.2*f.3*f.2*f.3*f.2*f.3*f.2*f.3*f.2*f.3, f.1*f.3*f.1*f.3 ]
```

Relations should be presented as a string. Within the string, relations are separated by commas. A relation should be presented as a word in capital or small letters of the Latin alphabet. Besides letters the symbols '^', '(', ')' and '=' will be recognised. So, for example blanks and commutator signs are not recognised yet and may cause error messages. The symbol '^' stands for the power operation when the right hand side is a number and for conjugation when the right hand represents a word. Symbols should be grouped using normal brackets '(.)'. Single symbols do not need to be grouped. Some examples :

string	**meaning**
`"a"`	a
`"ab"`	ab
`"a^2"`	aa
`"a^b"`	$b^{-1}ab$
`"(ab)^(cd)"`	$d^{-1}c^{-1}abcd$
`"((bcd)^2e)^a"`	$a^{-1}bcdbcdea$
`"(ab)^(cd)=(ce)^2"`	$d^{-1}c^{-1}b^{-1}a^{-1}cdcece$

where the expression in 'meaning' is equal to 1.

A list of strings that, besides the normal relations, contains the string `_PCE` will be recognised as a list of relations that are not necessary to define the group in question but are meant to speed up a **Coset Enumeration** process. Whether these relations will be added to the group relations during the construction of the Group Record, is dependent on the value of the global boolean variable `GPL_PCE`. Here `_PCE` stands for 'Purpose of Coset Enumeration'. For example:

```
gap>  cox := "a3b5c3d3e4c";; rel := "a=(ce)^2";;
gap> extrarel := "_PCE,(abc)^5,(bcd)^5";
"_PCE,(abc)^5,(bcd)^5"
gap> m11 :=  FpGroupViaStrings( cox, rel, extrarel );;
## The map is: "abcde"
gap> m11.relators;
[ f.1*f.2*f.1*f.2*f.1*f.2, f.2*f.3*f.2*f.3*f.2*f.3*f.2*f.3*f.2*f.3,
f.3*f.4*f.3*f.4*f.3*f.4, f.4*f.5*f.4*f.5*f.4*f.5,
f.5*f.3*f.5*f.3*f.5*f.3*f.5*f.3, f.1^2, f.2^2, f.3^2, f.4^2, f.5^2,
f.1*f.3*f.1*f.3, f.1*f.4*f.1*f.4, f.1*f.5*f.1*f.5, f.2*f.4*f.2*f.4,
f.2*f.5*f.2*f.5, f.1^-1*f.3*f.5*f.3*f.5,
f.1*f.2*f.3*f.1*f.2*f.3*f.1*f.2*f.3*f.1*f.2*f.3*f.1*f.2*f.3,
f.2*f.3*f.4*f.2*f.3*f.4*f.2*f.3*f.4*f.2*f.3*f.4*f.2*f.3*f.4 ]
gap> GPL_PCE := false;;
gap> m11 :=  FpGroupViaStrings( cox, rel, extrarel );;
## The map is: "abcde"
gap> m11.relators;
[ f.1*f.2*f.1*f.2*f.1*f.2, f.2*f.3*f.2*f.3*f.2*f.3*f.2*f.3*f.2*f.3,
f.3*f.4*f.3*f.4*f.3*f.4, f.4*f.5*f.4*f.5*f.4*f.5,
f.5*f.3*f.5*f.3*f.5*f.3*f.5*f.3, f.1^2, f.2^2, f.3^2, f.4^2, f.5^2,
f.1*f.3*f.1*f.3, f.1*f.4*f.1*f.4, f.1*f.5*f.1*f.5, f.2*f.4*f.2*f.4,
f.2*f.5*f.2*f.5, f.1^-1*f.3*f.5*f.3*f.5 ]
```

`FpGroupViaStrings` will also accept a **list** containing entries as described above.

```
gap> list := [cox, rel, extrarel];;
gap> Size( FpGroupViaStrings( list ));
## The map is: "abcde"
7920
```

2.3 SubgroupViaStrings

`SubgroupViaStrings(` *G* `,` *elstring*`,` *map* `)`

Let G be a finitely presented group and *elstring* a string representing a list of words $w_1, ..., w_n$ in the string generators of G. *map* should be a string containing every string generator of G as printed out by the procedure 'FpGroupViaStrings'. `SubgroupViaStrings` returns the subgroup U generated by $w_1, ..., w_n$ with parent group G. *elstring* should be a string of the form `"w_1,w_2, ... ,w_n"` with w_i a word in the string generators of G.

```
gap> cox := "a3b5c3d3e4c";;
gap> rel := "a=(ce)^2" ;;
gap> extrarel := "_PCE,(abc)^5,(bcd)^5";;
gap> m11 := FpGroupViaStrings( cox, rel, extrarel );;
## The map is: "abcde"
gap> map := "abcde";;
gap> a6 := SubgroupViaStrings( m11, "b,c,d,d^(abcbae)", map );
Subgroup( M11,
[ f.2, f.3, f.4, f.5^-1*f.1^-1*f.2^-1*f.3^-1*f.2^-1*f.1^-1*f.4*f\
.1*f.2*f.3*f.2*f.1*f.5 ] )
gap> OperationCosetsFpGroup( m11, a6 );
Group( ( 1, 2)( 3, 5)( 6, 9)(10,15)(11,16)(17,19)(18,20)(21,22),
( 2, 4)( 5, 8)( 7,11)( 9,14)(10,15)(12,17)(13,18)(21,22),
( 3, 5)( 4, 7)( 6,10)( 8,13)( 9,15)(11,16)(17,21)(19,22),
( 3, 6)( 5, 9)( 7,12)( 8,14)(10,15)(11,17)(16,19)(21,22),
( 1, 3)( 2, 5)( 4, 8)( 7,13)(10,15)(11,18)(16,20)(21,22) )
```

2.4 StringPresentation

`StringPresentation(` *G* `)`

Let G be a finitely presented group. `StringPresentation(` *G* `)` will return the presentation of G as a list of strings. The first string lists the string generators of G, the second string the relators as words in the generators.

```
gap> F2 := FreeGroup( "A", "B" );
Group( A, B )
gap> A5 := F2 / [ F2.1^2, F2.2^3, (F2.1*F2.2)^5 ];
Group( A, B )
gap> StringPresentation( A5 );
[ "A,B", "AA,BBB,ABABABABAB" ]
```

2.5 CoxeterStringPresentation

`CoxeterStringPresentation(` *G* `)`

Let G be a finitely presented group. `CoxeterStringPresentation(` *G* `)` will return the presentation of G as a list of strings. It will try to represent as much as possible of the presentation as a Coxeter graph. For this purpose the algorithm will first detect the generators of order two. Then it will call the package `Grape`, see [So93] to compute the maximal complete subgraphs of the graph spanned by the relators of the form `(xy)^n` for all generators

x and y of order 2. It will select one such subgraph that contains a maximal number of points.

```
gap> F5 := FreeGroup( 5 );
Group( f.1, f.2, f.3, f.4, f.5 )
gap> rel := [ (F5.1*F5.2)^3, (F5.2*F5.3)^5, (F5.3*F5.4)^3,
  > (F5.4*F5.5)^5, F5.1^2, F5.2^2, F5.3^2, F5.4^2, F5.5^2,
  > (F5.1*F5.3)^2, (F5.1*F5.4)^2, (F5.1*F5.5)^2, (F5.2*F5.4)^2,
  > (F5.2*F5.5)^2, (F5.3*F5.5)^2, (F5.1*F5.2*F5.3)^3,
  > F5.1^-1*(F5.3*F5.4*F5.5)^5, (F5.2*F5.3*F5.4)^5 ];;
gap> Janko1 := F5/rel;
Group( f.1, f.2, f.3, f.4, f.5 );
gap> CoxeterStringPresentation( Janko1 );
[ "A3B5C3D5E",, "(ABC)^3,A(CDE)^5,(BCD)^5" ]
```

Bibliography

[ATLAS] J.H. Conway, R.T. Curtis, S.P. Norton, R.A. Parker and R.A. Wilson, *An ATLAS of Finite Groups*, Clarendon Press, Oxford, 1985.

[GAP] M. Schönert, et al., GAP: Groups, Algorithms and Programming, Lehrstuhl D für Mathematik, RWTH Aachen, 1995.
Manual: `http://www.math.rwth-aachen.de/LDFM/GAP/Info/manual.html`

[PS97] C.E. Praeger and L.H. Soicher, *Low rank representations and graphs for sporadic groups*, Cambridge University Press, 1997.

[So93] L.H. Soicher, GRAPE: a system for computing with graphs and groups, in *Groups and Computation* (L. Finkelstein and W.M. Kantor, eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science **11**, A.M.S., 1993, pp. 287–291.