FILTERS PACKAGE

JOSHUA MAGLIONE

Contents

1. Overview

The goal of this package is to provide Magma [BCP] with the basic tools to do computations with filters. Proofs and details of algorithms are found in [M1, M2, W1, W2].

A *filter* for a group G is a function $\phi : M \to 2^G$ from a commutative monoid $M = \langle M, +, 0, \preceq \rangle$ into the normal subgroups of G satisfying the following: for all $s, t \in M$

$$[\phi_s, \phi_t] \leq \phi_{s+t} \qquad \& \qquad s \preceq t \text{ implies } \phi_t \leq \phi_s.$$

Associated to every filter ϕ is a *boundary filter* $\partial\phi : M \to 2^G$ such that for all $s \in M$,

$$\partial\phi_s = \langle \phi_{s+t} \mid t \in M - \{0\} \rangle.$$

Wilson proves [W1, Theorem 3.1] that each filter has an associated Lie ring:

$$L(\phi) = \bigoplus_{s \in M - \{0\}} \phi_s / \partial\phi_s.$$

This chapter makes assumptions on the monoid M to facilitate computations. Throughout we assume that $M = \mathbb{N}^d$ for some d and that $\preceq$ is a total ordering of M. Therefore, all filters are series.

We define a new type for filters with `Flt`.

Date: May 11, 2016.

2. Filters

2.1. **Constructors.** There are a few ways to construct filters. The first two intrinsics are the most broad.

```
Filter(X, D, I, P) : Any, SeqEnum, SeqEnum, UserProgram -> Flt
TotallyOrderedFilter(X, D, I, P : \parameters) : Any, SeqEnum, SeqEnum, UserProgram -> Flt
Lex : BoolElt : true
```

Given a group X, a sequence of maximal indices D for the groups in the image I, and a function $P(x, y)$ such that `true` implies that $x \preceq y$, construct the filter associated to this data. The only difference between `Filter` and `TotallyOrderedFilter` is that the filter is flagged as totally ordered. The algorithm does not perform a check, so we assume the user believes it. If the total order is a lex order, then set `Lex` equal to `true`. This will only make a difference if the filter gets refined.

```
LowerCentralFilter(G) : Grp -> Flt
```

Returns the filter constructed from the lower central series of G.

```
pCentralFilter(G, p) : Grp, RngIntElt -> Flt
pCentralFilter(G) : GrpPC -> Flt
```

Returns the filter constructed from the exponent-p central series of G.

```
JenningsFilter(G) : Grp -> Flt
```

Returns the filter constructed from the Jennings series of G.

Example `Filt_Const`

We construct a filter.

```
> G := SmallGroup(2^4*3^4,100);
> F := LowerCentralFilter(G);
> F;
Filter from N into the subsets of GrpPC.
> 3@F;
GrpPC of order 162 = 2 * 3^4
PC-Relations:
    $.1^2 = Id($),
    $.2^3 = $.4,
    $.3^3 = Id($),
    $.4^3 = Id($),
    $.5^3 = Id($),
    $.3^$.2 = $.3 * $.5
```

2.2. **Operations.** We descibe the operations on filters.

```
Length:
  (F::Flt) -> RngIntElt
```

Returns the number of subgroups in the image of the filter. One can also use # as well.

```
Image:
  (F::Flt) -> SeqEnum
```

Returns the sequence of subgroups in the image of the filter.

```
MaximalIndices:
  (F::Flt) -> SeqEnum
```

Returns the sequence of maximal indices for each of the subgroups in the image of the filter.

```
Preorder:
  (F::Flt) -> UserProgram
```

Returns the pre-order of the filter. If $P(x, y)$ is the pre-order, then `true` implies that $x \preceq y$.

```
IsSeries:
  (F::Flt) -> BoolElt
```

Returns `true` if the filter is a series. (Note this does not imply that $\preceq$ is a total order for $\mathbb{N}^d$.)

```
Lattice:
  (F::Flt) -> GrphDir
```

Returns the lattice of the subgroups in the image of the filter.

```
BoundaryFilter:
  (F::Flt) -> Flt
```

Returns the boundary filter.

```
> G := SmallGroup(3^6,100);
> F := JenningsFilter(G);
> Lattice(F);
Digraph
Vertex  Neighbours

1       2 ;
2       3 ;
3       4 ;
4       5 ;
5       6 ;
6       ;

> #F;
6
> Image(F);
[
    GrpPC : G of order 729 = 3^6
    PC-Relations:
        G.1^3 = G.6,
        G.2^3 = G.4^2 * G.5,
        G.3^3 = G.5^2 * G.6,
        G.4^3 = G.6^2,
        G.2^G.1 = G.2 * G.3,
        G.3^G.1 = G.3 * G.4,
        G.3^G.2 = G.3 * G.6,
        G.4^G.1 = G.4 * G.5,
        G.5^G.1 = G.5 * G.6,

    GrpPC of order 81 = 3^4
    PC-Relations:
        $.1^3 = $.3^2,
        $.2^3 = $.4^2,
```

```
    GrpPC of order 27 = 3^3
    PC-Relations:
        $.1^3 = $.3^2,

    GrpPC of order 9 = 3^2
    PC-Relations:
        $.1^3 = Id($),
        $.2^3 = Id($),

    GrpPC of order 3
    PC-Relations:
        $.1^3 = Id($),

    GrpPC of order 1
    PC-Relations:
]
> B := BoundaryFilter(F);
> #B;
6
> Image(B);
[
    GrpPC : G of order 729 = 3^6
    PC-Relations:
        G.1^3 = G.6,
        G.2^3 = G.4^2 * G.5,
        G.3^3 = G.5^2 * G.6,
        G.4^3 = G.6^2,
        G.2^G.1 = G.2 * G.3,
        G.3^G.1 = G.3 * G.4,
        G.3^G.2 = G.3 * G.6,
        G.4^G.1 = G.4 * G.5,
        G.5^G.1 = G.5 * G.6,

    GrpPC of order 81 = 3^4
    PC-Relations:
        $.1^3 = $.3^2,
        $.2^3 = $.4^2,

    GrpPC of order 27 = 3^3
    PC-Relations:
        $.1^3 = $.3^2,

    GrpPC of order 9 = 3^2
    PC-Relations:
        $.1^3 = Id($),
        $.2^3 = Id($),

    GrpPC of order 3
    PC-Relations:
        $.1^3 = Id($),

    GrpPC of order 1
```

```
    PC-Relations:
]
```

3. Graded Lie algebras

We have created a new attribute for Lie algebras which is a designated spot to store a record based on the graded information. This record is only assigned to Lie algebras that come from filters.

```
AlgLie:
  GradedInfo:
    Bimaps                      HomogeneousIndices
    BimapIndices                Projections
    HomogeneousComponents       StructureConstants
```

We allow the standard Lie algebra function to accept filters, and we make the information contained `GradedInfo` accessible to the user. All the intrinsics that require a Lie algebra will assume that `GradedInfo` has been assigned, and therefore, comes from a filter.

```
LieAlgebra:
  (F::Flt) -> AlgLie, Map
```

Returns the Lie algebra associated to the filter and a map from the group to the Lie algebra.

```
GradedProducts:
  (L::AlgLie) -> List
```

Returns the bimaps given by the graded product on the associated Lie algebra. Only bimaps with positive dimensional domains and codomains are returned. The indices of the domain of the bimaps are also included.

```
FilterToBimap:
  (F::Flt, s::RngIntElt, t::RngIntElt) -> TenSpcElt
  (F::Flt, s::SeqEnum, t::SeqEnum) -> TenSpcElt
```

Returns the bimap of the associated Lie algebra corresponding to $L_s \times L_t \rightarrowtail L_{s+t}$. If any L_i are 0-dimensional, then `false` is returned.

```
HomogeneousComponents:
  (L::AlgLie) -> List, List
```

Returns the list of homogeneous components of L as vector spaces, and for each homogeneous space, a projection from L into the component.

```
StructureConstants
  (L::AlgLie) -> SeqEnum
```

Returns the structure constants of L.

```
> G := SmallGroup(3^6,100);
> F := pCentralFilter(G);
> L, phi := LieAlgebra(F);
> L;
Lie Algebra of dimension 6 with base ring GF(3)
> phi;
Mapping from: GrpPC: G to AlgLie: L given by a rule
> GradedProducts(L);
[*
```

```
    <<[ 1 ], [ 1 ]>, Tensor V2 x V1 >-> V0
    V2 : Full Vector space of degree 2 over GF(3)
    V1 : Full Vector space of degree 2 over GF(3)
    V0 : Full Vector space of degree 1 over GF(3)>,
    <<[ 1 ], [ 2 ]>, Tensor V2 x V1 >-> V0
    V2 : Full Vector space of degree 2 over GF(3)
    V1 : Full Vector space of degree 1 over GF(3)
    V0 : Full Vector space of degree 1 over GF(3)>,
    <<[ 1 ], [ 3 ]>, Tensor V2 x V1 >-> V0
    V2 : Full Vector space of degree 2 over GF(3)
    V1 : Full Vector space of degree 1 over GF(3)
    V0 : Full Vector space of degree 1 over GF(3)>,
    <<[ 1 ], [ 4 ]>, Tensor V2 x V1 >-> V0
    V2 : Full Vector space of degree 2 over GF(3)
    V1 : Full Vector space of degree 1 over GF(3)
    V0 : Full Vector space of degree 1 over GF(3)>,
    <<[ 2 ], [ 2 ]>, Tensor V2 x V1 >-> V0
    V2 : Full Vector space of degree 1 over GF(3)
    V1 : Full Vector space of degree 1 over GF(3)
    V0 : Full Vector space of degree 1 over GF(3)>,
    <<[ 2 ], [ 3 ]>, Tensor V2 x V1 >-> V0
    V2 : Full Vector space of degree 1 over GF(3)
    V1 : Full Vector space of degree 1 over GF(3)
    V0 : Full Vector space of degree 1 over GF(3)>
*]
> FilterToBimap(F,1,3);
Tensor V2 x V1 >-> V0
V2 : Full Vector space of degree 2 over GF(3)
V1 : Full Vector space of degree 1 over GF(3)
V0 : Full Vector space of degree 1 over GF(3)
> HomogeneousComponents(L);
[*
    Full Vector space of degree 2 over GF(3),

    Full Vector space of degree 1 over GF(3),

    Full Vector space of degree 1 over GF(3),

    Full Vector space of degree 1 over GF(3),

    Full Vector space of degree 1 over GF(3)
*] [*
    Mapping from: AlgLie: L to Full Vector space of degree 2
    over GF(3),
    Mapping from: AlgLie: L to Full Vector space of degree 1
    over GF(3),
    Mapping from: AlgLie: L to Full Vector space of degree 1
    over GF(3),
    Mapping from: AlgLie: L to Full Vector space of degree 1
    over GF(3),
    Mapping from: AlgLie: L to Full Vector space of degree 1
    over GF(3)
*]
```

```
> StructureConstants(L);
[
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0],

    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0],

    [0 2 0 0 0 0]
    [1 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0],

    [0 0 2 0 0 0]
    [0 0 0 0 0 0]
    [1 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0],

    [0 0 0 2 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [1 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0],

    [0 0 0 0 2 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [0 0 0 0 0 0]
    [1 0 0 0 0 0]
    [0 0 0 0 0 0]
]
```

4. Refining filters

We provide a number of different ways to refine a filter using methods of linear algebra. See [W1, W2] for details. Throughout, we assume that the filters are totally ordered.

```
AdjointRefinement:
  (F::Flt) -> Flt, SeqEnum
```

```
CentroidRefinement:
  (F::Flt) -> Flt, SeqEnum
DerivationRefinement:
  (F::Flt) -> Flt, SeqEnum
LeftScalarRefinement:
  (F::Flt) -> Flt, SeqEnum
RightScalarRefinement:
  (F::Flt) -> Flt, SeqEnum
```

Returns the resulting filter after applying the appropriate refinement. The algorithm starts at the first bimap in the associated Lie algebra, and continues to the last bimap until a refinement is found. Once a refinement is found, it creates a prefilter containing the given filter and the new subgroup and generates a new filter. If no refinements are found, then the given filter is returned. A sequence of strings is already returned detailing which bimap and refinement were used.

```
Refine:
  (F::Flt : Heuristics:=["all"], Iterations:=0) -> Flt, SeqEnum
```

Returns the resulting filter after applying refinements. Similar to the above refinements, the algorithm searchs the first bimap for a refinement depending on the heuristics chosen. If one is found, then it creates a prefilter and generates a filter. If `Iterations` is set to a positive integer, then the algorithm iterates this process no more than `Iterations` times. On the other hand, if all the heuristics have been exhausted and no refinement have been found, then algorithm continues to search through the rest of the bimaps. If no refinements are found the given filter is returned. After the appropriate number of iterations, the resulting filter is returned along with a sequence of strings describing the origin of the refinement: both the bimap and refinement used.

The parameter `Heuristics` must be set to sequence of strings. Acceptable strings are `"all"`, `"adjoint"`, `"centroid"`, `"derivation"`, `"left scalar"`, and `"right scalar"`. In general, the order of refinement matters, so the algorithm will refine in the order given. The default is

`["all"] = ["derivation","adjoint","left scalar","right scalar","centroid"]`.

The other parameter, `Iterations`, can be set to any nonnegative integer. Set `Iterations` to 0 if you want the algorithm to refine the filter until it can no longer be refined.

```
> G := SmallGroup(3^6,100);
> F := pCentralFilter(G);
> Refine(F);
Filter from N^2 into the subsets of GrpPC.
[
    <<[ 1 ], [ 2 ]>, "derivation: radical">
]
> AdjointRefinement(F);
Filter from N^2 into the subsets of GrpPC.
[
    <<[ 1 ], [ 2 ]>, "adjoint: radical">
]
```

```
> G := SmallGroup(2^9,3000000);
```

```
> F := pCentralFilter(G);
> Refine(F);
Filter from N^3 into the subsets of GrpPC.
[
    <<[ 1 ], [ 1 ]>, "derivation: radical">,
    <<[ 1, 0 ], [ 1, 1 ]>, "derivation: radical">
]
> Refine(F : Iterations:=1);
Filter from N^2 into the subsets of GrpPC.
[
    <<[ 1 ], [ 1 ]>, "derivation: radical">
]
> Refine(F : Heuristics:=["adjoint","centroid"], Iterations:=0);
Filter from N into the subsets of GrpPC.
```

```
> G := SylowSubgroup(Sym(50),2);
> G := PCGroup(G);
> F := pCentralFilter(G);
> Refine(F);
Filter from N^7 into the subsets of GrpPC.
[
    <<[ 1 ], [ 1 ]>, "derivation: radical">,
    <<[ 1, 0 ], [ 1, 1 ]>, "derivation: semisimple">,
    <<[ 1, 0, 0 ], [ 1, 1, 0 ]>, "derivation: radical">,
    <<[ 1, 1, 0, 0 ], [ 2, 1, 0, 0 ]>, "derivation: radical">,
    <<[ 1, 1, 0, 0, 1 ], [ 4, 2, 0, 0, 1 ]>, "derivation: radica
    l">,
    <<[ 1, 1, 1, 0, 0, 0 ], [ 2, 1, 1, 1, 0, 0 ]>, "derivation:
    radical">
]
```

5. Generating filters

We now discuss the intrinsics which will generate filters by inserting a subgroup. Again, we assume that the filter is totally ordered.

```
GenerateFilter:
  (F::Flt, H::Grp, i::SeqEnum, P::UserProgram : Lex:=false) -> Flt,
  (F::Flt, S::SeqEnum, I::SeqEnum, P::UserProgram : Lex:=false) -> Flt
```

Returns the filter generated by inserting the subgroup H (or the sequence of subgroups S) in the filter with index i (or the sequenece of indices I). If the pre-order P is lexicographical, set `Lex` to `true`. If the domain of F is $\mathbb{N}^d$, it is assumed that $i \in \mathbb{N}^{d+1}$.

```
ConstructFilter:
  (F::Flt, X::[GrpElt]) -> Flt,
  (F::Flt, H::Grp) -> Flt
```

Returns a filter generated by including the subgroup $\langle X \rangle$ or H into the filter. The given filter is returned if its location cannot be determined.

```
> G := SmallGroup(5^6,500);
> O := Omega(G,1);
> O;
```

```
GrpPC : O of order 3125 = 5^5
PC-Relations:
    O.2^O.1 = O.2 * O.4^2,
    O.3^O.2 = O.3 * O.5
> F := pCentralFilter(G);
> 2@F;
GrpPC of order 125 = 5^3
PC-Relations:
    $.1^5 = Id($),
    $.2^5 = Id($),
    $.3^5 = Id($)
> P := function(x,y) // Lex-order
function>   i := 1;
function>   while i le #x do
function|while>     if x[i] lt y[i] then
function|while|if>       return true;
function|while|if>     elif x[i] gt y[i] then
function|while|if>       return false;
function|while|if>     end if;
function|while>     i +:= 1;
function|while>   end while;
function>   return true;
function> end function;
>
> FF := GenerateFilter(F, O, [1,1], P : Lex:=true);
> FF;
Filter from N^2 into the subsets of GrpPC.
> #F;
4
> #FF;
6
```

```
> G := SmallGroup(2^9,1000000);
> F := pCentralFilter(G);
> H := sub< G | G.2,G.3*G.5>^G;
> ConstructFilter(F,H);
Filter from N^2 into the subsets of GrpPC.
```

```
> G := SmallGroup(2^9,1000000);
> F := pCentralFilter(G);
> H := sub< G | G.1,G.2 >^G;
> ConstructFilter(F,H);
Filter from N^2 into the subsets of GrpPC.
```

References

[BCP] Wieb Bosma, John Cannon, and Catherine Playoust, *The Magma algebra system. I. The user language*, J. Symbolic Comput. **24** (1997), no. 3-4, 235–265. Computational algebra and number theory (London, 1993). MR1484478

[M1] Joshua Maglione, *Efficient refinements for finite groups*, submitted.

[M2] ———, *Longer nilpotent series for classical unipotent subgroups*, J. Group Theory **18** (2015), no. 4, 569–585. MR3365818

[W1] James B. Wilson, *More characteristic subgroups, Lie rings, and isomorphism tests for p-groups*, J. Group Theory **16** (2013), no. 6, 875–897, DOI 10.1515/jgt-2013-0026. MR3198722

[W2] ———, *New Lie products for groups and their automorphisms*, submitted. `arXiv:1501.04670`.

Index

Department of Mathematics, Colorado State University, Fort Collins, CO 80523, USA

E-mail address: maglione@math.colostate.edu