SRGroups

Self-relicating groups acting on regular rooted trees

0.9

6 July 2021

Samuel King

Sarah Shotter

Stephan Tornier

Samuel King
Email: samuel.s.king@newcastle.edu.au
Address: University Drive, Callaghan NSW 2308

Sarah Shotter
Email: sarah.shotter@newcastle.edu.au
Address: University Drive, Callaghan NSW 2308

Stephan Tornier
Email: stephan.tornier@newcastle.edu.au
Homepage: https://www.newcastle.edu.au/profile/stephan-tornier
Address: University Drive, Callaghan NSW 2308

Abstract

SRGroups is a package for searching up self-replicating groups of regular rooted trees and performing computations on these groups. This package allows the user to generate more self-replicating groups at greater depths with its in-built functions, and is an extension of the transgrp package.

Copyright

Acknowledgements

DE210100180, FL170100032.

Contents

Chapter 1

Introduction

Let G be a subgroup of the regular rooted k-tree, $\mathrm{Aut}(T_k)$ with its group action, α, defined as $\alpha(g,x) = g(x)$, where $g \in G$ are the automorphisms of G and $x \in X$ the vertices of T_k. Let $\mathrm{stab}_G(0) = \{g \in G : \alpha(g,0) = 0\}$, and $T_0 \subset T_k$ be the set of all vertices below and including the vertex 0. Additonally, let $\varphi_0 : \mathrm{stab}_G(0) \to G$ be a group homomorphism with the mapping $g \mapsto g|_{T_0}$. Then G is called self-replicating if and only if the following two conditions, $\mathscr{R}_k$, are satisfied: G is vertex transitive on level 1 of T_k, and $\varphi_0(\mathrm{stab}_G(0)) = G$.

A group $G \leq \mathrm{Aut}(T_k)$ acting greater than level 1 is said to have sufficient rigid automorphisms if for each pair of vertices u and v on level 1 of the tree, $T_{k,1}$, there exists an automorphism $g \in G$ such that $g(u) = v$ and $g|_u = e$, where g is called (u,v)-rigid. For a self-replicating group G on level n of the tree, $T_{k,n}$, with sufficient rigid automorphisms, the maximal extension of G, $\mathscr{M}(G)$, is the largest self-replicating group (not necessarily with sufficient rigid automorphisms) on $\mathrm{Aut}(T_{k,n+1})$ that projects onto G, defined as:

$$\mathscr{M}(G) := \{x \in \mathrm{Aut}(T_{k,n+1}) : \varphi_{n+1}(x) \in G and x|_v \in G \text{for all v on level 1}\}.$$

The self-replicating property is preserved across conjugacy. For a group $H \leq \mathrm{Aut}(T_{k,n})$ with sufficient rigid automorphisms, and a self-replicating group $G \leq \mathrm{Aut}(T_{k,n+1})$, there exists a conjugate of G in $\mathrm{Aut}(T_{k,n+1})$ with sufficient rigid automorphisms. Since groups on level 1 inherently have sufficient rigid automorphisms, then self-replicating groups with sufficient rigid automorphisms can be found on all levels of the tree.

The SRGroups package serves to provide a library of these self-replicating groups to further the ongoing studies of infinite networks and group theory. By using the above definitions and conditions, several GAP methods and functions have been built to allow computations of these groups and expand the understanding of their behaviour.

First, this package acts as a library for searching currently known self-replicating groups for varying degrees and levels of regular rooted trees. This package also acts as a regular GAP package with functions that allow the expansion of the library and addition of attributes/properties relevant to self-replicating groups. Additional functions also exist in this package that are compatible with GraphViz, to plot diagrams of the extension behaviour of these self-replicating groups and their corresponding Hasse diagrams at different depths.

1.1 Purpose

Research and educational purpose. To do.

Chapter 2

Preliminaries

Introductory text. To do.

2.1 Regular rooted tree groups

2.1.1 IsRegularRootedTreeGroup (for IsPermGroup)

▷ `IsRegularRootedTreeGroup(arg)` (filter)

Returns: `true` or `false`

Groups acting on the regular rooted trees $T_{k,n}$ are stored together with their degree $k \in \mathbb{N}_{\geq 2}$ (see `RegularRootedTreeGroupDegree` (2.1.2)) and depth $n \in \mathbb{N}$ (see `RegularRootedTreeGroupDepth` (2.1.3)) as well as other attributes and properties in this category.

2.1.2 RegularRootedTreeGroupDegree (for IsRegularRootedTreeGroup)

▷ `RegularRootedTreeGroupDegree(G)` (attribute)

Returns: The degree of `G`.

The argument of this attribute is a regular rooted tree group `G`.

Example

```
gap> RegularRootedTreeGroupDegree(AutT(2,3));
2
```

2.1.3 RegularRootedTreeGroupDepth (for IsRegularRootedTreeGroup)

▷ `RegularRootedTreeGroupDepth(G)` (attribute)

Returns: The depth of `G`.

The argument of this attribute is a regular rooted tree group `G`.

Example

```
gap> RegularRootedTreeGroupDepth(AutT(2,3));
3
```

2.1.4 RegularRootedTreeGroup (for IsInt, IsInt, IsPermGroup)

▷ `RegularRootedTreeGroup(k, n, G)` (operation)

Returns: The regular rooted tree group `G` as an object of the category `IsRegularRootedTreeGroup` (2.1.1), together with its degree `k` (see `RegularRootedTreeGroupDegree` (2.1.2)) and its depth `n` (see `RegularRootedTreeGroupDepth` (2.1.3)).

The arguments of this method are a permutation group $G \leq \mathrm{Aut}(T_{k,n})$, its degree `k` and its depth `n`.

2.2 Auxiliary functions

2.2.1 BelowAction

▷ `BelowAction(k, n, aut, i)` (function)

Returns: The restriction of `aut` to the subtree below the level 1 vertex `i`, as an element of $\mathrm{Aut}(T_{k,n-1})$.

The arguments of this function are a degree $k \in \mathbb{N}_{\geq 2}$, a depth $n \in \mathbb{N}$, an element `aut` of $\mathrm{Aut}(T_{k,n})$ (see `AutT` (3.2.1)), and a depth 1 vertex $i \in \{1, \cdots, k\}$ of $T_{k,n}$.

Example

```
gap> BelowAction(2,2,(1,2)(3,4),2);
(1,2)
```

2.2.2 RemoveConjugates

▷ `RemoveConjugates(G, subgroups)` (function)

Returns: Removes `G`-conjugates from the list `subgroups`.

The arguments of this function are a group `G` and a list `subgroups` of subgroups of `G`.

Example

```
gap> G:=SymmetricGroup(3);
Sym( [ 1 .. 3 ] )
gap> subgroups:=[Group((1,2)),Group((2,3)),Group((1,3))];
[ Group([ (1,2) ]), Group([ (2,3) ]), Group([ (1,3) ]) ]
gap> RemoveConjugates(G,subgroups);
gap> subgroups;
[ Group([ (1,2) ]) ]
```

Chapter 3

Self-replicating groups

Introductory text. To do.

3.1 Properties and Attributes

3.1.1 IsSelfReplicating (for IsRegularRootedTreeGroup)

▷ `IsSelfReplicating(G)` (property)

Returns: `true` if G is self-replicating, and `false` otherwise.
The argument of this property is aa regular rooted tree group G.

Example

```
gap> subgroups:=AllSubgroups(AutT(2,2));
[ Group(()), Group([ (3,4) ]), Group([ (1,2) ]), Group([ (1,2)(3,4) ]),
  Group([ (1,3)(2,4) ]), Group([ (1,4)(2,3) ]), Group([ (3,4), (1,2) ]),
  Group([ (1,3)(2,4), (1,2)(3,4) ]), Group([ (1,3,2,4), (1,2)(3,4) ]),
  Group([ (3,4), (1,2), (1,3)(2,4) ]) ]
gap> Apply(subgroups,G->RegularRootedTreeGroup(2,2,G));
gap> Apply(subgroups,G->IsSelfReplicating(G));
gap> subgroups;
[ false, false, false, false, false, false, false, true, true, true ]
```

3.1.2 HasSufficientRigidAutomorphisms (for IsRegularRootedTreeGroup)

▷ `HasSufficientRigidAutomorphisms(G)` (property)

Returns: `true` if G has sufficient rigid automorphisms, and `false` otherwise.
The argument of this property is a regular rooted tree group G

Example

```
gap> subgroups:=AllSubgroups(AutT(2,2));
[ Group(()), Group([ (3,4) ]), Group([ (1,2) ]), Group([ (1,2)(3,4) ]),
  Group([ (1,3)(2,4) ]), Group([ (1,4)(2,3) ]), Group([ (3,4), (1,2) ]),
  Group([ (1,3)(2,4), (1,2)(3,4) ]), Group([ (1,3,2,4), (1,2)(3,4) ]),
  Group([ (3,4), (1,2), (1,3)(2,4) ]) ]
gap> Apply(subgroups,G->RegularRootedTreeGroup(2,2,G));
gap> Apply(subgroups,G->HasSufficientRigidAutomorphisms(G));
gap> subgroups;
[ false, false, false, false, true, false, false, true, true, true ]
```

3.1.3 ParentGroup (for IsRegularRootedTreeGroup)

▷ `ParentGroup(G)` (attribute)

Returns: The restriction of G to $\mathrm{Aut}(T_{k,n-1})$.

The argument of this attribute is a regular rooted tree group $G \leq \mathrm{Aut}(T_{k,n})$.

Example

```
gap> G:=AutT(2,3);;
gap> ParentGroup(G);
Group([ (1,2), (1,3)(2,4), (3,4) ])
gap> last=AutT(2,2);
true
```

3.1.4 MaximalExtension (for IsRegularRootedTreeGroup)

▷ `MaximalExtension(G)` (attribute)

Returns: The maximal extension of $M(G) \leq \mathrm{Aut}(T_{k,n+1})$ of G.

The argument of this attribute is a self-replicating regular rooted tree group $G \leq \mathrm{Aut}(T_{k,n})$ with sufficient rigid automorphisms.

Example

```
gap> G:=AutT(2,3);;
gap> MaximalExtension(G);
<permutation group with 11 generators>
gap> last=AutT(2,4);
true
```

3.1.5 RepresentativeWithSufficientRigidAutomorphisms (for IsRegularRootedTreeGroup)

▷ `RepresentativeWithSufficientRigidAutomorphisms(G)` (attribute)

Returns: A self-replicating $\mathrm{Aut}(T_{k,n})$-conjugate of G with sufficient rigid automorphisms. If the parent group of G has sufficient rigid automorphisms then the output group has the same parent group (see `ParentGroup` (3.1.3)) as G.

The argument of this attribute is a self-replicating regular rooted tree group $G \leq \mathrm{Aut}(T_{k,n})$.

Example

```
gap> G:=SRGroup(2,3,6);;
gap> conjugates:=ShallowCopy(AsList(G^AutT(2,3)));
[ Group([ (1,5)(2,6)(3,7)(4,8), (1,3)(2,4)(5,7)(6,8), (1,2)(3,4) ]),
  Group([ (1,5)(2,6)(3,8)(4,7), (1,3)(2,4)(5,8)(6,7), (1,2)(3,4) ]) ]
gap> Apply(conjugates,H->RegularRootedTreeGroup(2,3,H));
gap> for H in conjugates do Print(HasSufficientRigidAutomorphisms(H),"\n"); od;
true
false
gap> H:=conjugates[2];
Group([ (1,5)(2,6)(3,8)(4,7), (1,3)(2,4)(5,8)(6,7), (1,2)(3,4) ])
gap> IsSelfReplicating(H);
true
gap> RepresentativeWithSufficientRigidAutomorphisms(H);
Group([ (1,5)(2,6)(3,7)(4,8), (1,3)(2,4)(5,7)(6,8), (1,2)(3,4) ])
gap> last=conjugates[1];
true
```

3.2 Examples

AutT. More to come. Grigorchuk, Hanoi, ...

3.2.1 AutT

▷ `AutT(k, n)` (function)

Returns: The regular rooted tree group $\mathrm{Aut}(T_{k,n})$ as a permutation group of the k^n leaves of $T_{k,n}$.

The arguments of this function are a degree $k \in \mathbb{N}_{\geq 2}$ and a depth $n \in \mathbb{N}$.

Example

```
gap> G:=AutT(2,2);
Group([ (1,2), (3,4), (1,3)(2,4) ])
gap> Size(G);
8
```

3.2.2 ConjugacyClassRepsMaxSelfReplicatingSubgroups

▷ `ConjugacyClassRepsMaxSelfReplicatingSubgroups(G)` (function)

Returns: A list of $\mathrm{Aut}(T_{k,n})$-conjugacy class representatives of all self-replicating, maximal subgroups of G.

The argument of this function is a regular rooted tree group G.

Example

```
gap> Size(ConjugacyClassRepsMaxSelfReplicatingSubgroups(AutT(2,2)));
2
NrSRGroups(2,2);
3
```

3.2.3 ConjugacyClassRepsSelfReplicatingSubgroupsWithConjugateProjection

▷ `ConjugacyClassRepsSelfReplicatingSubgroupsWithConjugateProjection(G)` (function)

Returns: A list of $\mathrm{Aut}(T_{k,n+1})$-conjugacy class representatives of all self-replicating subgroups of $\mathrm{Aut}(T_{k,n+1})$ whose parent group is conjugate to G.

The argument of this function is a regular rooted tree group $G \leq \mathrm{Aut}(T_{k,n})$.

Example

```
gap> A3:=RegularRootedTreeGroup(3,1,AlternatingGroup(3));;
gap> S3:=RegularRootedTreeGroup(3,1,SymmetricGroup(3));;
gap> A3_extn:=ConjugacyClassRepsSelfReplicatingSubgroupsWithConjugateProjection(A3);;
gap> S3_extn:=ConjugacyClassRepsSelfReplicatingSubgroupsWithConjugateProjection(S3);;
gap> Size(A3_extn);
5
gap> Size(S3_extn);
11
gap> NrSRGroups(3,2);
16
```

Chapter 4

The library of self-replicating groups

Introductory text. To do.

4.1 Availability functions

Introductory text. To do. Similarities with transitive groups library.

4.1.1 SRGroupsAvailable

▷ `SRGroupsAvailable(k, n)` (function)

Returns: Whether the self-replicating groups of degree, k, and depth, n, are available.
The argument of this function is a degree, k, and a depth, n.

Example

```
gap> SRGroupsAvailable(2,5);
true
gap> SRGroupsAvailable(5,2);
false
```

4.1.2 NrSRGroups

▷ `NrSRGroups(k, n)` (function)

Returns: The number of self-replicating groups of degree, k, and depth, n.
The argument of this function is a degree, k, and a depth, n.

Example

```
gap> NrSRGroups(2,3);
15
gap> NrSRGroups(2,5);
2436
```

4.1.3 SRDegrees

▷ `SRDegrees()` (function)

Returns: All of the degrees currently stored in the SRGroups library.
There are no inputs to this function.

Example

```
gap> SRDegrees();
[ 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17 ]
```

4.1.4 SRLevels

▷ SRLevels(*k*) (function)

Returns: All of the levels currently stored in the SRGroups library for an input RegularRootedTreeGroupDegree, *deg*.

The input to this function is the degree of the regular rooted tree, *k*.

Example

```
gap> SRLevels(2);
[ 1, 2, 3, 4 ]
```

4.2 Selection functions

4.2.1 SRGroup

▷ SRGroup(*k, n, num*) (function)

Returns: The *num*th self-replicating group of degree *k* and depth *n* stored in the SRGroups library.

The argument of this function is a degree, *k*, a depth, *n*, and a designated number of the stored self-replicating group, *num*.

Example

```
gap> SRGroup(2,3,1);
SRGroup(2,3,1)
gap> Size(last);
8
```

4.2.2 AllSRGroups

▷ AllSRGroups(*Input1, val1, Input2, val2, ...*) (function)

Returns: A list of self-replicating groups matching the input arguments as RegularRootedTreeGroup objects.

Main library search function that acts analogously as the AllTransitiveGroups function from the transgrp library. Has several possible input arguments such as *Degree*, *Depth* (or *Level*), *Number*, *Projection*, *IsSubgroup*, *Size*, *NumberOfGenerators*, and *IsAbelian*. Order of the arguments do not matter. List inputs and singular inputs can be provided. The argument definitions are as follows: *Degree* (int > 1) := degree of tree *Depth*/*Level* (int > 0) := level of tree *Number* (int > 0) := self-replicating group number *Projection* (int > 0) := groups whose projected image are the group number on the level above *IsSubgroup* (int > 0) := groups that are a subgroup of the group number provided *Size* (int >= degree^depth or int > 1) := size of group *MinimalGeneratingSet* (int > 0) := size of the group's minimal generating set *IsAbelian* (boolean) := all groups that are abelian if true, and not abelian if false

Example

```
gap> AllSRGroups(Degree, 2, Level, 4, IsAbelian, true);
[ SRGroup(2,4,2), SRGroup(2,4,9), SRGroup(2,4,12), SRGroup(2,4,14) ]
```

```
gap> AllSRGroups(Degree,[2..5],Depth,[2..5],IsSubgroup,[1..5],Projection,[1..3]);
Restricting degrees to [ 2, 3 ]
[ SRGroup(2,1,1), SRGroup(2,1,1), SRGroup(2,2,1), SRGroup(2,3,1),
  SRGroup(2,3,2), SRGroup(2,4,1), SRGroup(2,4,1), SRGroup(2,4,2),
  SRGroup(2,4,2), SRGroup(2,4,2), SRGroup(3,1,1), SRGroup(3,1,1),
  SRGroup(3,1,1), SRGroup(3,1,1) ]
```

4.2.3 AllSRGroupsInfo

▷ `AllSRGroupsInfo(`*`Input1, val1, Input2, val2, ...`*`)` (function)

Returns: Information about the self-replicating group(s) satisfying all of the provided input arguments in list form: [*`Generators`*, *`Name`*, *`Parent Name`*, *`Children Name(s)`*]. If the *`Position`* input is provided, only the corresponding index of this list is returned.

Inputs work the same as the main library search function `AllSRGroups` (4.2.2), with one additional input: *`Position`* (or *`Index`*). Position/Index := (int in [1..4])

Example

```
gap> AllSRGroupsInfo(Degree, 2, Depth, [2,3], IsAbelian, true);
[ [ [ (1,2)(3,4), (1,3,2,4) ], "SRGroup(2,2,1)", "SRGroup(2,1,1)",
      [ "SRGroup(2,3,1)", "SRGroup(2,3,2)" ] ],
  [ [ (1,2)(3,4), (1,3)(2,4) ], "SRGroup(2,2,2)", "SRGroup(2,1,1)",
      [ "SRGroup(2,3,3)", "SRGroup(2,3,4)", "SRGroup(2,3,5)",
          "SRGroup(2,3,6)" ] ],
  [ [ (1,5,4,8,2,6,3,7), (1,4,2,3)(5,8,6,7), (1,2)(3,4)(5,6)(7,8) ]
        , "SRGroup(2,3,1)", "SRGroup(2,2,1)",
      [ "SRGroup(2,4,1)", "SRGroup(2,4,2)" ] ],
  [
      [ (1,5,2,6)(3,7,4,8), (1,3)(2,4)(5,7)(6,8),
          (1,2)(3,4)(5,6)(7,8) ], "SRGroup(2,3,4)",
      "SRGroup(2,2,2)",
      [ "SRGroup(2,4,8)", "SRGroup(2,4,9)", "SRGroup(2,4,10)" ] ],
  [ [ (1,3)(2,4)(5,7)(6,8), (1,5)(2,6)(3,7)(4,8),
          (1,2)(3,4)(5,6)(7,8) ], "SRGroup(2,3,5)",
      "SRGroup(2,2,2)",
      [ "SRGroup(2,4,11)", "SRGroup(2,4,12)", "SRGroup(2,4,13)",
          "SRGroup(2,4,14)", "SRGroup(2,4,15)" ] ] ]
gap> AllSRGroupsInfo(Degree, 2, Level, [2,3], IsAbelian, true, Position, 1);
[ [ (1,2)(3,4), (1,3,2,4) ], [ (1,2)(3,4), (1,3)(2,4) ],
  [ (1,5,4,8,2,6,3,7), (1,4,2,3)(5,8,6,7), (1,2)(3,4)(5,6)(7,8) ],
  [ (1,5,2,6)(3,7,4,8), (1,3)(2,4)(5,7)(6,8), (1,2)(3,4)(5,6)(7,8) ],
  [ (1,3)(2,4)(5,7)(6,8), (1,5)(2,6)(3,7)(4,8), (1,2)(3,4)(5,6)(7,8) ] ]
```

4.3 Extending the library

4.3.1 SRGroupFile

▷ `SRGroupFile(`*`k`*`)` (function)

The arguments of this function are a degree, *`k`*, or *`0`*. If the argument is non-zero, this function creates the file containing all self-replicating groups of the regular rooted k-tree at the lowest level

not stored in the SRGroups library. If the argument is *0*, this function creates the file containing all self-replicating groups of the regular rooted tree at the level 1 for the lowest degree not stored in the SRGroups library. The file naming convention is "sr_k_n.grp", and they are stored in the "data" folder of the SRGroups package. Level 1 groups are calculated using the transgrp library. If the argument is non-zero and there is a gap between files (i.e. if "sr_k_n.grp" and "sr_k_n+2.grp" exists, but "sr_k_n+1.grp" does not exist), then this function creates the files in this gap.

Example

```
gap> SRGroupFile(2);
You have requested to make group files for degree 2.
Creating level 3 file.
Evaluating groups extending from:
SRGroup(2,2,1)  (1/3)
SRGroup(2,2,2)  (2/3)
SRGroup(2,2,3)  (3/3)
SRGroup(2,2,4)  (4/3)
Formatting file sr_2_3.grp now.
Reordering individual files.
Done.
gap> SRGroupFile(0);
Creating degree 5 file on level 1.
Done.
gap> SRGroupFile(2);
You have requested to make group files for degree 2.
Gap found; missing file from level 2. Creating the missing file now.
Creating files:
sr_2_2.grp
Done.
```

4.3.2 ExtendSRGroup

▷ `ExtendSRGroup(`*arg*`)` (function)

The arguments of this function are: arg[1]: degree of tree (int > 1), *k*, arg[2]: highest level of tree where the file "sr_k_n.grp" exists (int > 1), *n*, (arg[3],arg[4],...): sequence of group numbers to extend from using the files "temp_k_n_arg[3]_arg[4]_...arg[Length(arg)-1].grp". This function creates the file of the group number arg[Length(arg)] stored in the file "temp_k_n_arg[3]_arg[4]_...arg[Length(arg)-1].grp", and saves it as "temp_k_n_arg[3]_arg[4]_...arg[Length(arg)].grp".

4.3.3 CombineSRFiles

▷ `CombineSRFiles(`*k, n*`)` (function)

The arguments of this function are a degree, *k*, and a level, *n*, of a regular rooted tree, *n-1* is the highest level stored as the file "sr_k_n-1.grp" in the SRGroups library, and all of the files "temp_k_n-1_i_proj.grp" for every SRGroup(k,n-1,i) are stored in the "data/temp_k_n" folder of the SRGroups library. This function combines each of the "temp_k_n-1_i_proj.grp" files into the complete "temp_k_n.grp" file to be used by the `SRGroupFile` (4.3.1) function.

4.3.4 CheckSRProjections

▷ `CheckSRProjections(k, n)` (function)

Returns: Whether all of the self-replicating groups of degree `k` and level `n` project correctly to level `n-1`. This is mainly used after obtaining new data to check that it has been formatted correctly (see `SRGroupFile` (4.3.1)).

The arguments of this function are a degree, `k`, and a level, `n`.

Example

```
gap> CheckSRProjections(2,4);
All groups project correctly.
```

References

Index