Smallsemi

A GAP package

Version 0.6.8

Andreas Distler
J. D. Mitchell

Andreas Distler Email: a.distler@tu-bs.de
Address: Institut Computational Mathematics
Rebenring 31 (A14)
38106 Braunschweig
Germany

J. D. Mitchell Email: jdm3@st-and.ac.uk
Homepage: http://www-groups.mcs.st-and.ac.uk/~jamesm
Address: Mathematical Institute
North Haugh
St Andrews, Fife
KY16 9SS
Scotland, UK

Copyright

Acknowledgements

We would like to thank Tom Kelsey for making this library possible by running all the initial computations in Minion [GJM06].

The first author acknowledges financial support of the University of St Andrews. The second author acknowledges support of EPSRC grant number GR/S/56085/01.

Colophon

If you use Smallsemi, please tell us by sending an email to a.distler@tu-bs.de or jdm3@st-and.ac.uk.

If you find any bugs or have any suggestions or comments, we would very much appreciate it if you would let us know. Also, we would like to hear about applications of this software.

Contents

Chapter 1

Introduction

This manual describes the Smallsemi package (Version 0.6.8) for GAP.

The Smallsemi package is a data library of semigroups of small size. It provides all semigroups with at most 8 elements as well as various information about these objects. The reason that semigroups of higher orders are not included is the huge number of such objects. The numbers of semigroups of sizes 1 to 9 are given in the table below (orders not included in the library are in italics). The number of semigroups of size 10 is not known at the time of writing.

Size	Number of semigroups
1	1
2	4
3	18
4	126
5	1 160
6	15 973
7	836 021
8	1 843 120 128
9	*52 989 400 714 478*

The initial idea for Smallsemi was developed out of the wish for an extensive number of examples of semigroups of moderate size. This lead to the idea of an electronical database. As an existing example the SmallGroups Library [BEO02] was an inspiration on how such a project could be established. Unfortunately the number of semigroups is so much bigger, and most of them bare so little structure, that new techniques to store and handle the semigroups had to be developed. Of course, the first step was to actually construct all the semigroups.

In the remainder of the introduction we explain what you need to do to install and optimize Smallsemi; see Subsections 1.1 and 1.2.

In Chapter 2 we explain how the semigroups where obtained, what exactly is stored and how, and which additional properties have been precomputed.

The the types of use Smallsemi is intended for and its limitations are described in Chapter 3. The extensive examples can be used as a quick-start guide and as something to come back to after reading the technical details about available functionality in the subsequent sections.

Chapter 4 has all the information about available functions.

1.1 Requirements

This software is written for GAP 4. It requires an existing installation of GAP in version 4.5 or higher.

It is recommended but not necessary to have the GAP 4 packages Citrus and sgpviz installed as well. Citrus provides a wide range of functionality for working with semigroups which is not available in the GAP core system while sgpviz is recommended for its ability to graphically represent small semigroups.

1.1.1 Operating System

The current version of Smallsemi was created for use under Unix. It will also work under Windows but only if all files in the directory `smallsemi/data` and all of its subdirectories are uncompressed. See Subsection 1.1.3 for additional comments on working with Smallsemi under Windows.

1.1.2 RAM

Working with Smallsemi can be memory expensive. We recommend to have at least 1 GB of RAM available. With less than 512 MB not all the semigroups of size 8 can be accessed.

You should be able to use the semigroups of orders 1 through 7 having 128 MB of RAM only. If you have a system with little memory or want to use as little memory as possible for the GAP process try using `UnloadSmallsemiData` (4.1.9) to free memory after every access to the library. This is likely to slow down computations though.

For further infomation on how GAP uses memory see 1.2.4 or (**Reference: Command Line Options**).

1.1.3 Disk Space

As the data in the library is compressed, 30 MB of disk space will be sufficient to install Smallsemi under Unix. To use the library under Windows the data has to be uncompressed and will then occupy approx. 1.6 GB.

All data files are compressed using `gzip`. Under Unix GAP can access the original contents of a gzipped file without uncompressing it as a whole by using a pipe. On 32-bit systems this might fail in extreme circumstances. In that case GAP has to be restarted. This functionality is *not* currently available under Windows (or for any other compression type).

It should be possible to use Smallsemi under Windows after unzipping all data files. (These are located in the directory `data` and its subdirectories and have the file extension `.gz`.)

1.2 Installation and Setup

1.2.1 Download and Extract Smallsemi

The installation follows standard GAP rules as outlined in the following two steps; see (**Reference: Installing a GAP Package**) for further details.

1. Download one of the archives `smallsemi-0.6.8.tar.gz` or `smallsemi-0.6.8.tar.bz2` from
   `http://www-history.mcs.st-andrews.ac.uk/~jamesm/smallsemi/index.html`

2. Move the archive inside a pkg directory. This can be either the main pkg directory in your GAP installation or your personal pkg directory.

3. Complete the installation by unpacking the archive, e.g. under Linux type `tar -xzf smallsemi-0.6.8.tar.gz` at the prompt for the gzipped tar-archive. A subdirectory `smallsemi` will be created inside the pkg directory.

1.2.2 Contents

In the subdirectory `smallsemi` you should find the following files and folders:

`CHANGELOG`	documents changes to previous versions
`data`	contains the data files for semigroups
`doc`	contains the documentation
`gap`	contains the GAP code
`GPLv3`	version 3 of the GNU General Public License
`init.g`	implementation file of Smallsemi
`PackageInfo.g`	meta information about Smallsemi
`read.g`	declaration file of Smallsemi
`README.txt`	the README file of Smallsemi
`tst`	contains test files

1.2.3 Loading

To use the package, start a GAP session and type `LoadPackage("smallsemi");` at the GAP prompt. You should see the following:

Example

```
gap> LoadPackage("smallsemi");
--------------------------------------------------------------------------------
Smallsemi - A library of small semigroups
by Andreas Distler & James Mitchell
For contents, type: ?Smallsemi:
Loading Smallsemi 0.6.8 ...
--------------------------------------------------------------------------------
true
gap>
```

You might want to start GAP with a specified amount of memory; see Subsection 1.2.4.

1.2.4 Memory Issues

As mentioned in Subsection 1.1.2, working with smallsemi can be memory expensive. It is therefore necessary to either:

1. start GAP with 1 GB of memory (if possible), for example, by typing `gap -m 1g`; or

2. extend the amount memory used by typing `return;` in the break-loop whenever GAP runs out of memory. For example,

Example

```
gap> s:=SmallSemigroup(8, 183244314);
#I  Loading 'smallsemi' data. Please be patient.
#I  Loading 'smallsemi' data.
Error, exceeded the permitted memory (‘-o’ command line option)
SplitString( StringFile( file ), "\n" ) called from
READ_3NIL_DATA( diag ); called from
RecoverMultiplicationTable( size, nr ) called from
<function>( <arguments> ) called from read-eval-loop
you can 'return;'
brk>
```

1.2.5 Testing

You should verify the success of the installation by running the test file. This is done by the following command and should return a similar output (the number of GAP4stones might differ depending on the speed of your machine):

Example

```
gap> ReadPackage( "smallsemi", "tst/testall.g" );
Smallsemi package: small.tst
GAP4stones: 41
Smallsemi package: properties.tst
GAP4stones: 0
Smallsemi package: enums.tst
GAP4stones: 1
```

1.2.6 Customizing

If you are using Smallsemi regularly you might want to put the command LoadPackage("smallsemi"); into your .gaprc file; see (**Reference: The gap.ini and gaprc files**). Another option is to save a workspace after loading Smallsemi and executing the following commands

Example

```
gap> SmallSemigroup(7,1);; MOREDATA2TO8;;
#I  smallsemi: loading data for semigroups of size 7.
#I  smallsemi: loading data for semigroup properties. Please be patient.
gap> SaveWorkspace( "<filename for the workspace>" );
```

Doing this will mean that it is not necessary to load the data from the library every time you start a new GAP session; see (**Reference: Saving and Loading a Workspace**).

The size of the file containing the saved workspace will be around 76 MB. Loading this workspace is much quicker than starting a new GAP session and all the data for semigroups of orders 1 through 7 is immediately available. (If you are working under Unix you can make use of the functionality mentioned in Subsection 1.1.3 and compress the workspace with gzip to roughly 10 MB.)

Chapter 2

The Data in the Library

In this chapter we outline how the semigroups in the library were found, exactly what semigroups are available, how they are stored, and how further information regarding the properties of these semigroups is handled.

2.1 Creation of the Semigroups

This section describes which semigroups are contained in the library and how they were determined.

The purpose of the library is to provide one semigroup of every 'structural type'. The semigroups are represented by their multiplication table. Usually, say, for groups, 'stuctural type' means 'up to isomorphism' which corresponds to relabelling the elements. Roughly speaking, transposing the multiplicationable of a semigroup does not alter its important structure features either. More precisely, the usual description of the structure of a semigroup using Green's relations is invariant under these operations. So, we consider two semigroups to be of the same structure if they are isomorphic or anti-isomorphic. We will refer to semigroups that are isomorphic or anti-isomorphic as *equivalent*. The vast number of non-equivalent semigroups with small numbers of elements (see Table 1) limits us to providing the semigroups with at most 8 elements.

The problem of constructing semigroups up to isomorphism and anti-isomorphism has been considered by many authors. For very small orders, that is 1 to 5, all the semigroups up to isomorphism and anti-isomorphism were computed by hand [THA+55] and [Tam54]. The first instance of the use of computers to find all semigroups up to isomorphism and anti-isomorphism is described in Forsythe [For55]. Subsequently, the number of semigroups with 6 elements was found by Plemmons [Ple67], with 7 elements by Jürgensen and Wick [JW77], with 8 by Satoh, Yama and Tokizawa [SYT94], and with 9 by Distler, Kelsey, and Mitchell in 2008. Even if the authors could store their results they had no means to make them publically available. Plemmons, for example, explicitly states that he had all multiplication tables for semigroups of size 6 on magnetic tape. Jürgensen and Wick back in 1977 did not store the semigroups of size 7 because of their large number. The tables for semigroups with 8 elements use up several gigabytes of disk space (while the compressed library files in Smallsemi need only 22 MB).

Trying to recreate the results from the existing literature, it quickly becomes obvious that even some 15 years later, with considerably more computing power available, the task of obtaining all semigroups with 8 elements is still by no means trivial. Our technique was to find all associative multiplication tables up to isomorphism and anti-isomorphism using a combination of GAP and the Contraint Satisfaction Problem (CSP) solver Minion [GJM06]. More specific details on the search

will be available in a later version of Smallsemi.

2.2 Storing the Semigroups

As discussed in the previous section, we store data relating to the multiplication table of one representative of every class of equivalent semigroups with 1 to 8 elements.

The tables for semigroups with 2 to 7 elements are stored in the files `data2.gl.gz` to `data7.gl.gz` in the directory `data/data2to7`.

For semigroups of size 8 the data is contained in the directories `data/data8-3nil` and `data/data8`. The former contains the data relating to 3-nilpotent semigroups (see `NilpotencyDegree` (4.2.34)) and the latter the data for all the remaining semigroups of size 8.

The tables of non-3-nilpotent semigroups are partitioned into files `8diag<entries in the diagonal>.gl.gz` with respect to their diagonals. For example, `8diag12345678.gl.gz` contains tables for all the bands of order 8.

Any 3-nilpotent semigroup has a unique minimal generating set containing those elements that do not appear in the table. We only require the subtable with entries corresponding to the product of two generators, as all other products are zero. Thus if m is the number of generators, we retain information regarding the entries of an $m \times m$ table. However, we do not store all the tables in this case. The $m \times m$ tables can be sorted into ranges and then the first table and the number of tables in the range are stored. For every diagonal there is a file `diag<entries in the diagonal>.gl.gz` containing the first tables of each range and a separate file named `diag<entries in the diagonal>pos.gl.gz` containing the lengths of these ranges.

class	file names	data size	- gzipped	compression factor
sizes 2-7	`data<size>.gl`	39 MB	680 KB	58
size 8, not 3-nilpotent	`8diag<diagonal>.gl`	613 MB	10 MB	61
size 8, 3-nilpotent	`diag<diagonal>.gl`	974 MB	11 MB	89

All together the GAP library files take just under 22 MB of disk space after compression while allowing fast recovering of the data. The compression rates demonstrated in the table above were achieved using `gzip` with the highest possible compression (-9 switch) as well as careful analysis and intensive testing of how best to structure the data in the files.

The semigroups in the library satisfying certain standard properties have been identified and this information is stored in the files `info1.g.gz` to `info8.g.gz`. To find out what properties have been considered see `PrecomputedSmallSemisInfo` (4.5.19).

Chapter 3

Extended Examples

The main features of the library can be summarized in three points: it provides a complete set of semigroups up to isomorphism and anti-isomorphism of sizes up to 8; it carries a vast amount of precomputed information about these semigroups; and there is an identification function which takes a semigroup with at most 8 elements and returns a map to the equivalent one from the library.

These features lead to different ways of using the library. It is impossible to describe - or even to anticipate - all possible types of usage. Most problems will admit multiple solutions. We find it difficult to predict which will be most effective. The examples in this chapter should give an idea of the differences in the various functions and help you to find an alternative if a computation uses more time or memory than you have available.

Let us go step by step through some ways to use the library showing which tools are provided.

3.1 Lists, Enumerators and Iterators of Semigroups

At first one could want to search through the stored semigroups for one or all semigroups with a certain property. Going through all the semigroups can take a long time. Just to create all the 1.8 billion semigroups as objects in GAP takes around a day on a modern PC. Doing a simple test on all the semigroups in the library might take another day. Performing complicated tests easily takes weeks. To avoid this, many properties of the semigroups were precomputed. Semigroups with or without a precomputed property can be accessed as quickly as simply creating the same number of semigroups. (Note that timings of two calls to the same command may vary and, of course, heavily depend on your machine.)

Example

```
gap> # obtain a list of all semigroups with 6 elements
gap> AllSmallSemigroups( 6 );;
gap> time;
2636
gap> # obtain a list of all commutative semigroups with 7 elements
gap> AllSmallSemigroups( 7, IsCommutative, true );;
gap> time;
2957
gap> # compare the numbers of semigroups in the two lists
gap> NrSmallSemigroups( 6 ); NrSmallSemigroups( 7, IsCommutative, true );
15973
17291
```

(In all the examples in this section the info messages which are given by default when data is loaded are turned off via `SetInfoLevel(InfoSmallSemi,0)`.)

We provide three commands that can be used if one is interested in all semigroups with some properties. These are `AllSmallSemigroups` (4.5.1), `EnumeratorOfSmallSemigroups` (4.5.2), and `IteratorOfSmallSemigroups` (4.5.11). Which one is best to use depends a lot on the situation. Here we attempt to provide some insight about the essential differences.

3.1.1 Precomputed properties

We start with examples using only precomputed information. In this case there is essentially no advantage of calling an iterator instead of an enumerator. Thus only `AllSmallSemigroups` (4.5.1) and `EnumeratorOfSmallSemigroups` (4.5.2) will be considered.

We first compare the memory usage and the setup time. Assume we are interested in the commutative semigroups with at most 7 elements.

Example

```
gap> list := AllSmallSemigroups([1..7],IsCommutativeSemigroup,true);;
gap> time; # the time needed will always depend on your machine
3180
gap> enum := EnumeratorOfSmallSemigroups([1..7],IsCommutativeSemigroup,true);
<enumerator of semigroups of sizes [ 1 .. 7 ]>
gap> time;
8
```

The enumerator stores the information, which semigroups it contains, but only creates the semigroups when asked for them explicitly.

Example

```
gap> # now the semigroups have to be created ...
gap> for sg in enum do
# do nothing, the semigroup will be created anyway
od;
gap> time;
3428
gap> # ... and again if you want to look through them another time ...
gap> for sg in enum do
od;
gap> time;
3437
gap> # ... not so for the list of semigroups though
gap> for sg in list do
od;
gap> time;
4
```

There are several reasons why one would nevertheless prefer an enumerator, one is the smaller need for memory. While the number of semigroups in this example is rather moderate (compared with all the semigroups in the library) the difference is remarkable:

Example

```
gap> nr := Length(enum);
17291
gap> MemoryUsage(enum);
70507
```

```
gap> MemoryUsage(list); # this will take a while ...
19089280
gap> # ... but you can get a close approximation much faster
gap> sg := OneSmallSemigroup(7,IsCommutativeSemigroup,true);
<small semigroup of size 7>
gap> nr*MemoryUsage(sg);
19020100
```

As said before the advantage of the enumerator comes from the fact that the members of it are created anew every time they are called. This means on the other hand that information that is computed is not stored.

Example

```
gap> IsZeroSemigroup(list[3]); # a semigroup from the list ...
false
gap> KnownPropertiesOfObject(list[3]); # ... can store new information
[ "IsEmpty", "IsTrivial", "IsNonTrivial", "IsFinite", "IsDuplicateFree",
  "IsAssociative", "IsCommutativeSemigroup", "IsZeroSemigroup" ]
gap> IsZeroSemigroup(enum[3]); # semigroups in the enumerator ...
false
gap> KnownPropertiesOfObject(enum[3]); # ... are created anew in every call
[ "IsEmpty", "IsTrivial", "IsNonTrivial", "IsFinite", "IsDuplicateFree",
  "IsAssociative", "IsCommutativeSemigroup" ]
gap> # but if it turns out this is the semigroup you want to analyse, just do
gap> sg := enum[3];
```

Observe that in the last example the semigroup from the enumerator knew about the property that was used to create the enumerator. The enumerator stores this knowledge and passes it on whenever a member is called.

Another reason to prefer an enumerator is that one might only be interested in some of the elements it contains. This could become clear after analysing some of the elements and then there is no time wasted in creating all semigroups in the enumerator. Or possibly creating the enumerator involving precomputed properties was just the first step. As described in Section 4.5 enumerators themselves can be given as argument to get to a more restricted class of semigroups. This leads us to the next part of this section.

3.1.2 User functions

We now come to examples dealing with properties that are not precomputed - including user defined functions. This makes `IteratorOfSmallSemigroups` (4.5.11) interesting again. Assume you want to work with bands (`IsBand` (4.2.5)) of order 8 having 1 Green's $\mathcal{D}$-class (see (**Reference: Green's Relations**)). You might feel tempted to implement a function testing a semigroup for this combination of properties.

Example

```
gap> isFascinatingSemigroup := function(sgrp)
local dclasses;
dclasses := GreensDClasses(sgrp);
return IsBand(sgrp) and Length(dclasses) = 1;
end;
```

But then the precomputed property `IsBand` (4.2.5) is hidden inside your function and a call like `AllSmallSemigroups(8,isFascinatingSemigroup,true)` would take days to complete.

The following finds the same semigroups more efficiently:

Example

```
gap> list:=AllSmallSemigroups(8,IsBand,true,x->Size(GreensDClasses(x)),1);
[ <small semigroup of size 8>, <small semigroup of size 8> ]
gap> time;
49211
gap> enum:=EnumeratorOfSmallSemigroups(8,IsBand,true,x->Size(GreensDClasses(x)),1);
<enumerator of semigroups of size 8>
gap> time;
48723
```

Observe that the enumerator lost its advantage of returning the answer faster because not all properties are precomputed. Thus all bands have to be constructed to test their number of D-classes. As the number of such semigroups is small, `AllSmallSemigroups` (4.5.1) is the better choice in this example - remember that the semigroups from the enumerator have to be recreated in every call. Often one does not have this kind of knowledge beforehand. Even for a large number of semigroups the enumerator still has the advantage of using far less memory as it stores only the IDs of the semigroups. Before explaining more about this let us for a moment go back to the semigroups from the previous example. It turns out they are the 2 non-equivalent rectangular bands (`IsRectangularBand` (4.2.22)) with 8 elements.

Example

```
gap> ForAll(list,IsRectangularBand);
true
```

As a last example in this subsection we look at semigroups from the library that are not nilpotent. As there are quite some of these we will first try an enumerator. The obvious call seems to be

Example

```
gap> enum1 := EnumeratorOfSmallSemigroups([1..7],IsNilpotentSemigroup,false);
<enumerator of semigroups of sizes [ 2, 3, 4, 5, 6, 7 ]>
gap> time;
103403
```

However, we would like to include the semigroups of order 8 as well. As `IsNilpotentSemigroup` (4.2.20) is not a precomputed property in the current version of Smallsemi this would take a long time. Here, additional knowledge, about the way the semigroups are stored in the library, is helpful. The description of `NilpotencyDegree` (4.2.34) contains information on the IDs of all 3-nilpotent semigroups of order 8. We can create an enumerator without those semigroups doing the following:

Example

```
gap> # all 8 element semigroups that are not 3-nilpotent
gap> enum2 := EnumeratorOfSmallSemigroupsByIds([8],[[1..11433106]]);
<enumerator of semigroups of size 8>
```

Out of this enumerator the subclass of not nilpotent semigroups can be extracted.

Example

```
gap> enum3 := EnumeratorOfSmallSemigroups(enum2,IsNilpotentSemigroup,false);
gap> # This still takes quite a while though
gap> time;
1931140
```

You can avoid the waiting time at setup by using an iterator instead of an enumerator. An iterator does not know how many elements it contains, one can always just access the next element - if such exists - and one cannot go back. (Making copies of an iterator can help to circumvent this problem.) On the other hand one could in the above example start investigating the first couple of elements right away.

Example

```
gap> iter := IteratorOfSmallSemigroups(enum2,IsNilpotentSemigroup,false);
<iterator of semigroups of size 8>
gap> for i in [1..100000] do
NextIterator(iter);
od;
gap> time;
30785
```

But even if you know you want to inspect all the semigroups having a property which is not precomputed, an iterator has the advantage that it does not create the semigroups before you can actually work with them. To create an enumerator all semigroups in question will be created and - as said before - every element is created anew when it is accessed. An iterator on the other hand creates the semigroups in question one-by-one and returns the next one having the property. This makes a big difference if the number of semigroups one is interested in is big like in the example of not nilpotent semigroups of size 8. In the former example with the rectangular bands it would not play a role and the disadvantages of an iterator would prevail.

As you can see the number of semigroups you are interested in is even more important in the case of user defined functions than it was in the previous section about precomputed properties. Sometimes you might have a rough idea about the numbers - or even a very good one - to base your choice on. Otherwise the best approach seems to consist of two steps. First, create an enumerator involving all precomputed properties (try to find as many implied properties as possible). Then work with an iterator, call the semigroups one-by-one and store them in a separate list if you think you might want to look at them again at a later stage.

3.1.3 Semigroups of order 8

When using enumerators and iterators of semigroups of order 8 there are some limitations. In a 32-bit system the number of semigroups of order 8 exceeds the maximal length of a list in GAP. The following will work in a 64-bit system, but not on a 32-bit system.

Example

```
gap> EnumeratorOfSmallSemigroups(8);
```

In all other cases there is currently no difference between 32-bit and 64-bit systems. Hence the following will fail in any case.

Example

```
gap> EnumeratorOfSmallSemigroups(8,IsCommutativeSemigroup,false);
```

Note though that an enumerator of semigroups of order 8 can be created if one of the required properties is precomputed and takes `true` as value. This fact was used in the previous subsection, when creating the enumerator of all bands of order 8 having 1 Green's $\mathcal{D}$-class.

One could try to circumvent the described problem by using a iterator. The command

Example

```
gap> iter := IteratorOfSmallSemigroups(8,IsCommutativeSemigroup,false);
<iterator of semigroups of size 8>
```

will succeed. But running through the elements in the iterator can take a long time since the precomputed information is not utilized. A better idea in the current version of Smallsemi is to divide the enumerator into smaller pieces by restricting the range of IDs considered at once to at most $2^{28} - 1$ (the maximal length of a list in a 32-bit GAP) or possibly by a smaller value, depending on the amount of memory you have available. For example start with

Example

```
gap> enum1 := EnumeratorOfSmallSemigroupsByIds([8],[[1..2^24-1]]);
<enumerator of semigroups of size 8>
gap> enum2 := EnumeratorOfSmallSemigroups(enum1, IsCommutativeSemigroup, false);
<enumerator of semigroups of size 8>
```

Thanks go to Michal Stolorz for the idea of circumventing the current performance issue for enumerators of small semigroups of order 8 by splitting it in the described way.

3.2 Identifying Semigroups

The data in Smallsemi is as a big catalogue of all structural types of semigroups with at most 8 elements making it possible to refer to the types by their catalogue number, that is by their ID. With `IdSmallSemigroup` (4.1.6) one can find the ID of the structural type of a particular semigroup with at most 8 elements.

Example

```
gap> t1 := RandomTransformation(3);
Transformation( [ 1, 3, 1 ] )
gap> t2 := RandomTransformation(3);
Transformation( [ 1, 2, 3 ] )
gap> sgrp := SemigroupByGenerators([t1,t2]);
<semigroup with 2 generators>
gap> Size(sgrp);
3
gap> IdSmallSemigroup(sgrp);
[ 3, 8 ]
```

Moreover, one can draw conclusions about a semigroup of size at most 8 using the precomputed information about the equivalent semigroup from the library. The precomputed properties are all invariant under isomorphism and anti-isomorphism. This is most useful in the case where there is no method in GAP to decide the property in the original representation of the semigroup.

Example

```
gap> # use the semigroup from the previous example
gap> IsCommutative(sgrp); # no need to use the library for this
true
gap> # for the following there exists no method for a trans-
gap> # formation semigroup; access the precomputed information instead
gap> IsMultSemigroupOfNearRing(SmallSemigroup([3,8]));
false
```

`EquivalenceSmallSemigroup` (4.1.7) even provides an isomorphism or anti-isomorphism to a semigroup from the library. This means one can map elements between the semigroups. Remember that an isomorphism is returned whenever one exists. This allows to distinguish between structure types up to isomorphism. Note though, that no information about subsets - like the set of idempotents or a generating set - is precomputed for semigroups in the library. If an operation has a method for the semigroup in the original representation, it is usually more sensible to simply call this.

Example

```
gap> t1 := RandomTransformation(3);
Transformation( [ 2, 2, 1 ] )
gap> t2 := RandomTransformation(3);
Transformation( [ 2, 1, 1 ] )
gap> sgrp := SemigroupByGenerators([t1,t2]);
<semigroup with 2 generators>
gap> Size(sgrp);
6
gap> map := EquivalenceSmallSemigroup(sgrp);
MappingByFunction( <semigroup with 2 generators>, <small semigroup of size
6>, function( x ) ... end )
gap> RespectsMultiplication(map); # verify that this is an anti-isomorphism
false
gap> MinimalGeneratingSet(Range(map));
[ s2, s4 ]
gap> PreImage(map,last); # get a minimal generating set of <sgrp>
[ Transformation( [ 1, 1, 2 ] ), Transformation( [ 2, 1, 1 ] ) ]
gap> Idempotents(Range(map));
[ s1, s3, s5 ]
gap> PreImage(map,last); # in the same way you can get the idempotents ...
[ Transformation( [ 1, 1, 1 ] ), Transformation( [ 1, 2, 2 ] ),
  Transformation( [ 2, 2, 2 ] ) ]
gap> Idempotents(sgrp); # ... but this can be done directly instead
[ Transformation( [ 1, 1, 1 ] ), Transformation( [ 1, 2, 2 ] ),
  Transformation( [ 2, 2, 2 ] ) ]
```

If for a certain application you are interested in the semigroups up to isomorphism you can still use the IDs from Smallsemi. Simply mark the ID with $*$, or however else you denote the dual of a semigroup, to refer to the semigroup being anti-isomorphic to the one in the library having the same ID. For all semigroups `IsSelfDualSemigroup` (4.2.25) is precomputed. This will help to decide whether a semigroup and its dual are actually non-isomorphic.

Chapter 4

Functionality

4.1 Individual Semigroups

The semigroups of sizes 1 to 8 are available up to isomorphism and anti-isomorphism in Smallsemi. Every semigroup in the library is identified by its size m and a number n lying between 1 and the number of semigroups of size m (see Table 1). We call the pair (m,n) the *ID* of the semigroup.

In this section we give details about the functions relating to individual semigroups in Smallsemi. This includes how to access semigroups in the library and how to identify the semigroup in the library equivalent to an arbitrary semigroup (of size 1 to 8).

If you are interested in the properties of a semigroup in the library or would like to find all the semigroups satisfying a given set of properties please see Section 4.2 or Section 4.5 respectively.

4.1.1 SmallSemigroup

▷ `SmallSemigroup(m, n)` (function)
▷ `SmallSemigroupNC(m, n)` (function)

returns the semigroup with ID (m,n) from the library, that is the nth semigroup with m elements.

In `SmallSemigroupNC` no check is performed to verify that m and n are valid arguments.

In `SmallSemigroup` an error is signalled if the semigroups of size m are not classified or if n is greater than the number of semigroups with m elements.

Example

```
gap> SmallSemigroup(8,1353452);
<small semigroup of size 8>
gap> SmallSemigroupNC(5,1);
<small semigroup of size 5>
gap> SmallSemigroupNC(5,1)=SmallSemigroup(5,1);
true
```

4.1.2 IsSmallSemigroup

▷ `IsSmallSemigroup(sgrp)` (filter)

returns `true` if *sgrp* is a semigroup from the library, that is if it was created using `SmallSemigroup` (4.1.1). Otherwise `false` is returned.

Example
```
gap> sgrp:=RandomSmallSemigroup(5);
<small semigroup of size 5>
gap> IsSmallSemigroup(sgrp);
true
gap> sgrp:=Semigroup(Transformation([1]));;
gap> IsSmallSemigroup(sgrp);
false
```

4.1.3 IsSmallSemigroupElt

▷ IsSmallSemigroupElt(x) (filter)

returns true if x is an element of a semigroup from the library, and false otherwise.

IsSmallSemigroupElt is a representation satisfying *IsPositionalObjectRep* and IsMultiplicativeElement (**Reference: IsMultiplicativeElement**) and *IsAttributeStoringRep*.

Example
```
gap> IsSmallSemigroupElt(Transformation([1]));
false
gap> sgrp:=RandomSmallSemigroup(5);;
gap> IsSmallSemigroupElt(Random(sgrp));
true
```

4.1.4 RecoverMultiplicationTable

▷ RecoverMultiplicationTable(m, n) (function)
▷ RecoverMultiplicationTableNC(m, n) (function)

return the multiplication table of the n-th semigroup with m elements from the library.

If m is greater than 8 or n greater than the number of semigroups of size m, then fail is returned. The NC version does not perform any tests on the input and will most likely run into an error in such a case.

Example
```
gap> RecoverMultiplicationTable(10,2);
fail
gap> RecoverMultiplicationTable(1,2);
fail
gap> RecoverMultiplicationTable(2,1);
[ [ 1, 1 ], [ 1, 1 ] ]
gap> RecoverMultiplicationTable(8,11111111);
[ [ 1, 1, 1, 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1, 1, 1, 3 ],
  [ 3, 3, 3, 3, 3, 3, 3, 3 ], [ 1, 1, 1, 4, 4, 4, 4, 1 ],
  [ 1, 2, 3, 4, 5, 6, 7, 1 ], [ 1, 2, 3, 4, 5, 6, 7, 1 ],
  [ 1, 2, 3, 4, 5, 6, 7, 1 ], [ 8, 8, 8, 8, 8, 8, 8, 8 ] ]
gap> RecoverMultiplicationTable(2,11111111);
fail
```

Note that no semigroup is created calling this function but just the table is created. This makes it useful if one wants to perform very simple (i.e. quick in GAP) tests on a large number of semigroups which can be performed on the multiplication table.

To create a semigroup with the multiplication table obtained by `RecoverMultiplicationTable(m,n)` use the function `SmallSemigroup` (4.1.1) with arguments `m,n`.

4.1.5 SemigroupByMultiplicationTableNC

▷ `SemigroupByMultiplicationTableNC(table)` (function)

returns an object with `IsSemigroup` (**Reference: IsSemigroup**) and multiplication table `table` without checking if the multiplication defined by the table is associative.

If `table` is not associative, this can lead to errors and wrong results or might even crash GAP.

Example

```
gap> s:=SemigroupByMultiplicationTableNC([[1,2],[2,1]]);
<semigroup with 2 generators>
gap> IsSmallSemigroup(s);
false
```

Note that this function is *not* used to create semigroups when `SmallSemigroup` (4.1.1) is called. It can be useful in combination with `RecoverMultiplicationTable` (4.1.4) if one wants to avoid that a semigroup knows it comes from the library.

4.1.6 IdSmallSemigroup

▷ `IdSmallSemigroup(sgrp)` (attribute)

returns a pair `[m, n]` such that (m,n) is the ID of a semigroup in Smallsemi equivalent to `sgrp`. The argument `sgrp` has to be a semigroup of size 8 or less, otherwise an error is signalled.

Example

```
gap> sgrp:=Semigroup(Transformation( [ 1, 2, 2 ] ), Transformation( [ 1, 2, 3 ] ));;
gap> IdSmallSemigroup(sgrp);
[ 2, 3 ]
```

4.1.7 EquivalenceSmallSemigroup

▷ `EquivalenceSmallSemigroup(sgrp)` (attribute)

returns a mapping `map` from `sgrp` to the semigroup in Smallsemi equivalent to `sgrp`. The mapping is an isomorphism if such exists and an anti-isomorphism otherwise. The argument `sgrp` has to be a semigroup of size 8 or less, otherwise an error is signalled.

Example

```
gap> sgrp:=Semigroup(Transformation( [ 1, 2, 2 ] ),
> Transformation( [ 1, 2, 3 ] ));;
gap> EquivalenceSmallSemigroup(sgrp);
SemigroupHomomorphismByImages ( Monoid(
[ Transformation( [ 1, 2, 2 ] ) ] )->small semigroup of size 2>)
```

4.1.8 InfoSmallsemi

▷ `InfoSmallsemi` (info class)

is the info class (see (**Reference: Info Functions**)) of Smallsemi. The info level is initially set to 1 which triggers a message whenever data is loaded into GAP.

4.1.9 UnloadSmallsemiData

▷ `UnloadSmallsemiData(use_later)` (function)

deletes most or all of the data from the GAP workspace that was loaded by Smallsemi.

If the boolean `use_later` is `false` all data loaded by Smallsemi is deleted from the workspace, in which case Smallsemi is not guaranteed to work properly without restarting your GAP session.

If the boolean `use_later` is `true` only the recoverable data is deleted. This leaves roughly 10 MB of data in the workspace.

4.2 Properties of Semigroups

In this section we detail the GAP functions that can be used to determine whether a small semigroup satisfies a certain property. Let `S` be a semigroup. Then

- `S` is a *left zero semigroup* if `xy=x` for all `x,y` in `S`.
- `S` is a *right zero semigroup* if `xy=y` for all `x,y` in `S`.
- `S` is *commutative* if `xy=yx` for all `x,y` in `S`.
- `S` is *simple* if it has no proper two-sided ideals.
- `S` is *zero simple* if the only 2-sided ideals are `{0}` and `S`.
- `S` is *regular* if for all `x` in `S` there exists `y` in `S` such that `xyx=x`.
- `S` is *completely regular* if every element of `S` lies in a subgroup.
- `S` is an *inverse semigroup* if every element `x` in `S` has a unique semigroup inverse, that is, a unique element `y` such that `xyx=x` and `yxy=y`.
- `S` is a *Clifford semigroup* if it is a regular semigroup whose idempotents are central, that is, for all `e,x` in `S` where `e^2=e` we have that `ex=xe`.
- `S` is a *band* if every element is an idempotent, that is, `x^2=x` for all `x` in `S`.
- `S` is a *Brandt semigroup* if it is inverse and zero simple.
- `S` is a *rectangular band* if for all `x,y,z` in `S` we have that `x^2=x` and `xyz=xz`.
- `S` is a *semiband* if it is generated by its idempotent elements, that is, elements satisfying `x^2=x`.
- `S` is an *orthodox semigroup* if it is regular and its idempotents (elements satisfying `x^2=x`) form a subsemigroup.

- *S* is a *zero semigroup* if there exists an element *0* in *S* such that *xy=0* for all *x,y* in *S*.
- *S* is a *zero group* if there exists an element *0* in *S* such that *S* without *0* is a group and for all *x* in *S* we have that *x0=0x=0*.

The MONOID package was used to determined which semigroups in the library satisfy the properties above. All of the resulting information is stored in the library.

4.2.1 Annihilators

▷ `Annihilators(`*sgrp*`)` (attribute)

returns the set of annihilators of *sgrp* if *sgrp* contains a zero element and `fail` otherwise.

An element x in a semigroup with zero z is an *annihilator* if $xy = yx = z$ for every element y in the semigroup.

Example

```
gap> s := SmallSemigroup(5,6);
<small semigroup of size 5>
gap> Annihilators(s);
[ s1, s2 ]
```

4.2.2 DiagonalOfMultiplicationTable

▷ `DiagonalOfMultiplicationTable(`*sgrp*`)` (attribute)

returns the diagonal of the multiplication table of the semigroup *sgrp*.

Example

```
gap> s:=SmallSemigroup(8,10101);;
gap> DiagonalOfMultiplicationTable(s);
[ 1, 1, 1, 1, 1, 1, 1, 1 ]
gap> s:=SmallSemigroup(7,10101);;
gap> DiagonalOfMultiplicationTable(s);
[ 1, 1, 1, 1, 1, 1, 1 ]
```

4.2.3 DisplaySmallSemigroup

▷ `DisplaySmallSemigroup(`*sgrp*`)` (function)

displays all the information about the small semigroup *sgrp* that is stored in the library and its Green's classes and idempotents.

Example

```
gap> s:=SmallSemigroup(6, 3838);;
gap> DisplaySmallSemigroup(s);
IsBand:                          false
IsBrandtSemigroup:               false
IsCommutative:                   false
IsCompletelyRegularSemigroup:    false
IsFullTransformationSemigroupCopy:  false
IsGroupAsSemigroup:              false
IsIdempotentGenerated:           false
```

```
IsInverseSemigroup:                 false
IsMonogenicSemigroup:               false
IsMonoidAsSemigroup:                false
IsMultSemigroupOfNearRing:          false
IsOrthodoxSemigroup:                false
IsRectangularBand:                  false
IsRegularSemigroup:                 false
IsSelfDualSemigroup:                false
IsSemigroupWithClosedIdempotents:   true
IsSimpleSemigroup:                  false
IsSingularSemigroupCopy:            false
IsZeroSemigroup:                    false
IsZeroSimpleSemigroup:              false
MinimalGeneratingSet:               [ s3, s4, s5, s6 ]
Idempotents:                        [ s1, s5, s6 ]
GreensRClasses:                     [ {s1}, {s2}, {s3}, {s4}, {s5}, {\
s6} ]
GreensLClasses:                     [ {s1}, {s2}, {s3}, {s4}, {s6} ]
GreensHClasses:                     [ {s1}, {s2}, {s3}, {s4}, {s5}, {\
s6} ]
GreensDClasses:                     [ {s1}, {s2}, {s3}, {s4}, {s6} ]
```

4.2.4 IndexPeriod

▷ `IndexPeriod(x)` (attribute)

returns the minimum numbers *m, r* such that *x^{m+r}=x^m*; known as the index and period of the small semigroup element *x*.

Example

```
gap> s:=SmallSemigroup(5,116);
<small semigroup of size 5>
gap> x:=Elements(s)[3];
s3
gap> IndexPeriod(x);
[ 2, 1 ]
gap> x^3=x^2;
true
gap> x^2=x^1;
false
gap> x^3=x^1;
false
```

4.2.5 IsBand

▷ `IsBand(sgrp)` (property)

returns `true` if the small semigroup *sgrp* is a band and `false` otherwise.

A semigroup *sgrp* is a *band* if every element is an idempotent, that is, *x^2=x* for all *x* in *sgrp*.

Example

```
gap> s:=SmallSemigroup(5,519);;
gap> IsBand(s);
```

```
false
gap> s:=OneSmallSemigroup(5, IsBand, true);
<small semigroup of size 5>
gap> IsBand(s);
true
gap> IdSmallSemigroup(s);
[ 5, 1010 ]
```

4.2.6 IsBrandtSemigroup

▷ IsBrandtSemigroup(*sgrp*) (property)

returns true if the small semigroup *sgrp* is a Brandt semigroup and false otherwise.

A finite semigroup *sgrp* is a *Brandt semigroup* if it is inverse and zero simple.

Example

```
gap> s:=SmallSemigroup(5,519);;
gap> IsBrandtSemigroup(s);
false
gap> s:=OneSmallSemigroup(5, IsBrandtSemigroup, true);
<small semigroup of size 5>
gap> IsBrandtSemigroup(s);
true
gap> IdSmallSemigroup(s);
[ 5, 149 ]
```

4.2.7 IsCliffordSemigroup

▷ IsCliffordSemigroup(*sgrp*) (property)

returns true if the small semigroup *sgrp* is a Clifford semigroup and false otherwise.

A semigroup *sgrp* is a *Clifford semigroup* if it is a regular semigroup whose idempotents are central, that is, for all *e,x* in *sgrp* where *e^2=e* we have that *ex=xe*.

Example

```
gap> s:=SmallSemigroup(5,519);;
gap> IsBand(s);
false
gap> s:=OneSmallSemigroup(5, IsBand, true);
<small semigroup of size 5>
gap> IsBand(s);
true
gap> IdSmallSemigroup(s);
[ 5, 1010 ]
gap> s:=SmallSemigroup(5,519);;
gap> IsCliffordSemigroup(s);
false
gap> s:=OneSmallSemigroup(5, IsCliffordSemigroup, true);
<small semigroup of size 5>
gap> IsCliffordSemigroup(s);
true
gap> IdSmallSemigroup(s);
[ 5, 148 ]
```

4.2.8 IsCommutativeSemigroup

▷ `IsCommutativeSemigroup(`*sgrp*`)` (property)
▷ `IsCommutative(`*sgrp*`)` (property)

return `true` if the small semigroup *sgrp* is commutative and `false` otherwise.
A semigroup *sgrp* is *commutative* if *xy=yx* for all *x,y* in *sgrp*.

Example

```
gap> s:=SmallSemigroup(6,871);;
gap> IsCommutativeSemigroup(s);
false
gap> s:=OneSmallSemigroup(5, IsCommutative, true);
<small semigroup of size 5>
gap> IsCommutativeSemigroup(s);
true
gap> IsCommutative(s);
true
gap> IdSmallSemigroup(s);
[ 5, 1 ]
gap> s:=OneSmallSemigroup(5, IsCommutativeSemigroup, true);
<small semigroup of size 5>
gap> IsCommutativeSemigroup(s);
true
gap> IsCommutative(s);
true
```

4.2.9 IsCompletelyRegularSemigroup

▷ `IsCompletelyRegularSemigroup(`*sgrp*`)` (property)

returns `true` if the semigroup *sgrp* is completely regular and `false` otherwise.
A semigroup is *completely regular* if every element is contained in a subgroup.

Example

```
gap> s:=SmallSemigroup(6,13131);
<small semigroup of size 6>
gap> IsCompletelyRegularSemigroup(s);
false
gap> s:=OneSmallSemigroup(6, IsCompletelyRegularSemigroup, true);
<small semigroup of size 6>
gap> IsCompletelyRegularSemigroup(s);
true
gap> IdSmallSemigroup(s);
[ 6, 3164 ]
```

4.2.10 IsFullTransformationSemigroupCopy

▷ `IsFullTransformationSemigroupCopy(`*sgrp*`)` (property)

returns `true` if the semigroup *sgrp* is isomorphic to a full transformation semigroup and `false` otherwise.

The size of the full transformation semigroup on an n element set is n^n and so there are only two semigroup in the library that have this property.

Example

```
gap> s:=SmallSemigroup(1,1);
<small semigroup of size 1>
gap> IsFullTransformationSemigroupCopy(s);
true
gap> s:=OneSmallSemigroup(4, IsFullTransformationSemigroupCopy, true);
<small semigroup of size 4>
gap> IsFullTransformationSemigroupCopy(s);
true
gap> IdSmallSemigroup(s);
[ 4, 96 ]
gap> s:=OneSmallSemigroup(6, IsFullTransformationSemigroupCopy, true);
fail
```

4.2.11 IsGroupAsSemigroup

▷ `IsGroupAsSemigroup(`*sgrp*`)` (property)

returns `true` if the semigroup *sgrp* is a group and `false` otherwise.

Example

```
gap> s:=SmallSemigroup(7,7);
<small semigroup of size 7>
gap> IsGroupAsSemigroup(s);
false
gap> s:=SmallSemigroup(4,37);;
gap> IsGroupAsSemigroup(s);
true
```

4.2.12 IsIdempotentGenerated

▷ `IsIdempotentGenerated(`*sgrp*`)` (property)
▷ `IsSemiband(`*sgrp*`)` (property)

returns `true` if the semigroup *sgrp* is a semiband and `false` otherwise.

A semigroup *sgrp* is *idempotent generated* or equivalently a *semiband* if it is generated by its idempotent elements, i.e elements satisfying `x^2=x`.

Example

```
gap> s:=SmallSemigroup(3, 13);
<small semigroup of size 3>
gap> IsIdempotentGenerated(s);
true
gap> s:=OneSmallSemigroup(3, IsIdempotentGenerated, false);
<small semigroup of size 3>
gap> IsIdempotentGenerated(s);
false
gap> IdSmallSemigroup(s);
[ 3, 1 ]
gap> s:=OneSmallSemigroup(4, IsIdempotentGenerated, true,
```

```
> IsSingularSemigroupCopy, true);
fail
gap> s:=OneSmallSemigroup(2, IsIdempotentGenerated, true,
> IsSingularSemigroupCopy, true);
<small semigroup of size 2>
```

4.2.13 IsInverseSemigroup

▷ IsInverseSemigroup(*sgrp*) (property)

returns true if the semigroup *sgrp* is an inverse semigroup and false otherwise.

A semigroup *sgrp* is an *inverse semigroup* if every element *x* in *sgrp* has a unique semigroup inverse, that is, a unique element *y* such that *xyx=x* and *yxy=y*.

Example

```
gap> s:=OneSmallSemigroup(7, IsInverseSemigroup, true);
<small semigroup of size 7>
gap> IsInverseSemigroup(s);
true
gap> s:=SmallSemigroup(7, 101324);
<small semigroup of size 7>
gap> IsInverseSemigroup(s);
false
```

4.2.14 IsLeftZeroSemigroup

▷ IsLeftZeroSemigroup(*sgrp*) (property)

returns true if the semigroup *sgrp* is a left zero semigroup and false otherwise.

A semigroup *sgrp* is a *left zero semigroup* if *xy=x* for all *x,y* in *sgrp*.

Example

```
gap> s:=SmallSemigroup(5, 438);
<small semigroup of size 5>
gap> IsLeftZeroSemigroup(s);
false
gap> s:=SmallSemigroup(5, 1141);
<small semigroup of size 5>
gap> IsLeftZeroSemigroup(s);
true
```

4.2.15 IsMonogenicSemigroup

▷ IsMonogenicSemigroup(*sgrp*) (property)

returns true if the small semigroup *sgrp* is generated by a single element and false otherwise.

Example

```
gap> s:=RandomSmallSemigroup(7);
<small semigroup of size 7>
gap> IsMonogenicSemigroup(s);
false
```

```
gap> s:=OneSmallSemigroup(7, IsMonogenicSemigroup, true);
<small semigroup of size 7>
gap> IsMonogenicSemigroup(s);
true
gap> MinimalGeneratingSet(s);
[ s7 ]
gap> s:=SmallSemigroup( 7, 406945);
<small semigroup of size 7>
gap> IsMonogenicSemigroup(s);
false
```

4.2.16 IsMonoidAsSemigroup

▷ IsMonoidAsSemigroup(*sgrp*) (property)

returns true if the semigroup *sgrp* is a monoid (i.e. has an identity element) and false otherwise.

Example

```
gap> s:=SmallSemigroup(4, 126);
<small semigroup of size 4>
gap> IsMonoidAsSemigroup(s);
false
gap> s:=OneSmallSemigroup(4, IsMonoidAsSemigroup, true);
<small semigroup of size 4>
gap> IsMonoidAsSemigroup(s);
true
gap> One(s);
s1
gap> IdSmallSemigroup(s);
[ 4, 7 ]
```

4.2.17 IsMultSemigroupOfNearRing

▷ IsMultSemigroupOfNearRing(*sgrp*) (property)

returns true if *sgrp* is isomorphic (or anti-isomorphic?) to the multiplicative semigroup of a near-ring and false otherwise.

Those semigroups in the library that have this property were identified using the Sonata package.

Example

```
gap> s:=OneSmallSemigroup(7, IsMultSemigroupOfNearRing, true);
<small semigroup of size 7>
gap> IdSmallSemigroup(s);
[ 7, 1 ]
gap> IsMultSemigroupOfNearRing(s);
true
gap> s:=SmallSemigroup(2,2);
<small semigroup of size 2>
gap> IsMultSemigroupOfNearRing(s);
false
```

4.2.18 IsNGeneratedSemigroup

▷ IsNGeneratedSemigroup(*sgrp*, *n*) (operation)

returns true if the least size of a generating set for the small semigroup *sgrp* is *n* and false otherwise.

Example

```
gap> s:=SmallSemigroup(7, 760041);
<small semigroup of size 7>
gap> IsNGeneratedSemigroup(s, 4);
false
gap> IsNGeneratedSemigroup(s, 3);
true
gap> MinimalGeneratingSet(s);
[ s3, s5, s7 ]
gap> s:=OneSmallSemigroup(4, x-> Length(MinimalGeneratingSet(x)), 4);
<small semigroup of size 4>
gap> IsNGeneratedSemigroup(s, 4);
true
```

4.2.19 IsNIdempotentSemigroup

▷ IsNIdempotentSemigroup(*sgrp*, *n*) (operation)

returns true if the small semigroup *sgrp* has *n* idempotents and false otherwise.

Example

```
gap> s:=SmallSemigroup(4, 75);;
gap> IsNIdempotentSemigroup(s, 1);
false
gap> IsNIdempotentSemigroup(s, 2);
false
gap> IsNIdempotentSemigroup(s, 3);
true
```

4.2.20 IsNilpotentSemigroup

▷ IsNilpotentSemigroup(*sgrp*) (property)
▷ IsNilpotent(*sgrp*) (property)

returns true if the small semigroup *sgrp* is nilpotent and false otherwise.

A semigroup is *nilpotent* if it has a zero element and every element to some power equals this zero.

Example

```
gap> s:=SmallSemigroup(5,116);
<small semigroup of size 5>
gap> IsNilpotentSemigroup(s);
false
gap> s:=SmallSemigroup(7, 673768);;
gap> IsNilpotentSemigroup(s);
true
gap> s:=SmallSemigroup(7, 657867);;
```

```
gap> IsNilpotent(s);
true
```

4.2.21 IsOrthodoxSemigroup

▷ IsOrthodoxSemigroup(*sgrp*) (property)

returns true if the semigroup *sgrp* is orthodox and false otherwise.
A semigroup is *orthodox* if it is regular and its idempotents form a subsemigroup.

Example

```
gap> s:=SmallSemigroup(6, 15858);;
gap> IsSemigroupWithClosedIdempotents(s);
true
gap> IsRegularSemigroup(s);
true
gap> IsOrthodoxSemigroup(s);
true
```

4.2.22 IsRectangularBand

▷ IsRectangularBand(*sgrp*) (property)

returns true if the small semigroup *sgrp* is a rectangular band and false otherwise.
A semigroup *sgrp* is a *rectangular band* if for all *x,y,z* in *sgrp* we have that *x^2=x* and *xyz=xz*.

Example

```
gap> s:=SmallSemigroup(5, 216);;
gap> IsRectangularBand(s);
false
gap> s:=SmallSemigroup(6, 15854);;
gap> IsRectangularBand(s);
true
```

4.2.23 IsRegularSemigroup

▷ IsRegularSemigroup(*sgrp*) (property)

returns true if the small semigroup *sgrp* is a regular semigroup and false otherwise.
A semigroup *sgrp* is *regular* if for all *x* in *sgrp* there exists *y* in *sgrp* such that *xyx=x*.

Example

```
gap> s:=SmallSemigroup(3, 10);;
gap> IsRegularSemigroup(s);
true
gap> s:=SmallSemigroup(3, 1);;
gap> IsRegularSemigroup(s);
false
gap> s:=OneSmallSemigroup(4, IsFullTransformationSemigroupCopy, true);
<small semigroup of size 4>
gap> IsRegularSemigroup(s);
true
```

4.2.24 IsRightZeroSemigroup

▷ IsRightZeroSemigroup(*sgrp*) (property)

returns false for any small semigroup *sgrp* since the library contains only left zero semigroups. A semigroup *sgrp* is a *right zero semigroup* if *xy=y* for all *x,y* in *sgrp*.

Example

```
gap> s:=SmallSemigroup(5, 438);
<small semigroup of size 5>
gap> IsRightZeroSemigroup(s);
false
```

4.2.25 IsSelfDualSemigroup

▷ IsSelfDualSemigroup(*sgrp*) (property)

returns true if the semigroup *sgrp* is self dual and false otherwise.

A semigroup is *self dual* if it is isomorphic to its dual, that is, the semigroup *t* with multiplication * defined by *x*y=yx* where *yx* denotes the product in *sgrp*.

Example

```
gap> s:=SmallSemigroup(5,116);
<small semigroup of size 5>
gap> IsSelfDualSemigroup(s);
false
gap> s:=RandomSmallSemigroup(5, IsSelfDualSemigroup, true);
<small semigroup of size 5>
gap> IsSelfDualSemigroup(s);
true
```

4.2.26 IsSemigroupWithClosedIdempotents

▷ IsSemigroupWithClosedIdempotents(*sgrp*) (property)

returns true if the idempotent elements of the semigroup *sgrp* form a subsemigroup and false otherwise.

Example

```
gap> s:=SmallSemigroup(5, 677);;
gap> IsSemigroupWithClosedIdempotents(s);
true
gap> s:=SmallSemigroup(5, 659);;
gap> IsSemigroupWithClosedIdempotents(s);
true
gap> s:=SmallSemigroup(5, 327);;
gap> IsSemigroupWithClosedIdempotents(s);
false
```

4.2.27 IsSemigroupWithZero

▷ IsSemigroupWithZero(*sgrp*) (property)

returns `true` if the semigroup *sgrp* has a zero element and false otherwise.

An element z is a *zero* if $z*x = x*z = z$ for all x in the semigroup.

Example

```
gap> s:=SmallSemigroup(5,1);
<small semigroup of size 5>
gap> IsSemigroupWithZero(s);
true
gap> s:=SmallSemigroup(4,26);
<small semigroup of size 4>
gap> IsSemigroupWithZero(s);
false
```

4.2.28 IsSimpleSemigroup

▷ `IsSimpleSemigroup(`*sgrp*`)` (property)

▷ `IsCompletelySimpleSemigroup(`*sgrp*`)` (property)

return `true` if the semigroup *sgrp* is simple or completely simple and `false` otherwise.

A semigroup is *simple* if it has no proper 2-sided ideals. A semigroup is *completely simple* if it is simple and possesses minimal left and right ideals.

A finite semigroup is simple if and only if it is completely simple.

Example

```
gap> s:=SmallSemigroup(7, 835080);;
gap> IsSimpleSemigroup(s);
true
gap> IsCompletelySimpleSemigroup(s);
true
gap> s:=SmallSemigroup(7, 208242);;
gap> IsSimpleSemigroup(s);
false
```

4.2.29 IsSingularSemigroupCopy

▷ `IsSingularSemigroupCopy(`*sgrp*`)` (property)

returns `true` if the semigroup *sgrp* is isomorphic to a semigroup of singular (i.e. non-invertible) mappings on a finite set and `false` otherwise.

The size of this semigroup on an n element set is $n^n - n!$ and so there is only one semigroup in the library that has this property.

Example

```
gap> s:=SmallSemigroup(1,1);
<small semigroup of size 1>
gap> IsSingularSemigroupCopy(s);
false
gap> s:=OneSmallSemigroup(2, IsSingularSemigroupCopy, true);
<small semigroup of size 2>
gap> IsSingularSemigroupCopy(s);
true
gap> IdSmallSemigroup(s);
[ 2, 4 ]
```

```
gap> s:=OneSmallSemigroup(4, IsSingularSemigroupCopy, true);
fail
```

4.2.30 IsZeroGroup

▷ IsZeroGroup(*sgrp*) (property)

returns `true` if the semigroup *sgrp* is a zero group and `false` otherwise.

The semigroup *sgrp* is a *zero group* if there exists an element z in *sgrp* such that *sgrp* without z is a group and for all x in *sgrp* we have that $xz = zx = z$.

Example

```
gap> g:=Group((1,2),(3,4));
Group([ (1,2), (3,4) ])
gap> IdSmallSemigroup(g);
[ 4, 7 ]
gap> s := Range(InjectionZeroMagma(g));
<Group([ (1,2), (3,4) ]) with 0 adjoined>
gap> IdSmallSemigroup(s);
[ 5, 149 ]
gap> IsZeroGroup(s);
true
```

4.2.31 IsZeroSemigroup

▷ IsZeroSemigroup(*sgrp*) (property)

returns `true` if the semigroup *sgrp* is a zero semigroup and `false` otherwise.

The semigroup *sgrp* is a *zero semigroup* if there exists an element z in *sgrp* such that $xy = z$ for all x, y in *sgrp*.

Example

```
gap> s:=OneSmallSemigroup(5, IsZeroSemigroup, true);
<small semigroup of size 5>
gap> IsZeroSemigroup(s);
true
gap> IdSmallSemigroup(s);
[ 5, 1 ]
gap> s:=OneSmallSemigroup(5, IsZeroSemigroup, false);
<small semigroup of size 5>
gap> IdSmallSemigroup(s);
[ 5, 2 ]
gap> IsZeroSemigroup(s);
false
```

Note that for each size the unique zero semigroup is always the first semigroup of this size in the library.

Example

```
gap> IsZeroSemigroup(SmallSemigroup(6,1));
true
gap> IsZeroSemigroup(SmallSemigroup(7,1));
true
```

```
gap> IsZeroSemigroup(SmallSemigroup(8,1));
true
```

4.2.32 IsZeroSimpleSemigroup

▷ IsZeroSimpleSemigroup(*sgrp*) (property)

return `true` if the semigroup *sgrp* is zero simple and `false` otherwise.
A semigroup *sgrp* is *zero simple* if the only 2-sided ideals are the zero *{0}* and *sgrp*.

Example

```
gap> s:=SmallSemigroup(7, 519799);
<small semigroup of size 7>
gap> IsZeroSimpleSemigroup(s);
false
gap> s:=RandomSmallSemigroup(7, IsZeroSimpleSemigroup, true);
<small semigroup of size 7>
gap> IsZeroSimpleSemigroup(s);
true
```

4.2.33 MinimalGeneratingSet

▷ MinimalGeneratingSet(*sgrp*) (attribute)

returns a set of generators for *sgrp* with minimal size.

Example

```
gap> s:=SmallSemigroup(8, 1478885610);;
gap> MinimalGeneratingSet(s);
[ s4, s5, s6, s7, s8 ]
gap> s:=SmallSemigroup(7, 673768);;
gap> MinimalGeneratingSet(s);
[ s4, s5, s6, s7 ]
gap> s:=SmallSemigroup(4, 4);;
gap> MinimalGeneratingSet(s);
[ s2, s3, s4 ]
```

4.2.34 NilpotencyDegree

▷ NilpotencyDegree(*sgrp*) (attribute)

returns the least n such that every product of n elements in the nilpotent semigroup *sgrp* equals the zero element and returns `fail` if the semigroup *sgrp* is not nilpotent.

Example

```
gap> s := SmallSemigroup(5, 1121);;
gap> NilpotencyDegree(s);
fail
gap> s := SmallSemigroup(7, 393450);;
gap> NilpotencyDegree(s);
3
```

Note that for size 8 a semigroup in the library with ID $(8,n)$ is nilpotent of rank 3 if and only if n is greater than 11433106.

Example

```
gap> s := SmallSemigroup(8, 11433106+1231);;
gap> NilpotencyDegree(s);
3
gap> s := SmallSemigroup(8,NrSmallSemigroups(8));;
gap> NilpotencyDegree(s);
3
```

4.3 Nilpotent semigroups by coclass

A useful parameter in the classification of nilpotent semigroups is their coclass. For a finite nilpotent semigroup of order n and nilpotency degree d the coclass is defined as $n-d$. In [Dis14] lists up to (anti-)isomorphism are provided for nilpotent semigroups of coclass 0, 1, and 2. The semigroups in the lists are given by finite presentations. In this section we describe a function that allows to access such lists in GAP.

A further invariant of a nilpotent semigroup S is the size of its unique minimal generating set $S\backslash S^2$. The possible sizes for a particular coclass are restricted. Monogenic nilpotent semigroups are precisely those of coclass 0. For coclass $d \geq 1$ the size of the minimal generating set is at least 2 and at most $d+1$.

4.3.1 NilpotentSemigroupsByCoclass

▷ `NilpotentSemigroupsByCoclass(n, d[, r])` (function)

returns for a positive integer `n` and an integer `d` with value 0, 1, or 2 a list of nilpotent semigroups of order `n` and coclass `d` up to (anti-)isomorphism. If the optional third argument `r` is given then only semigroups of rank `r` are returned. The semigroups in the list are given by finite presentations.

Example

```
gap> NilpotentSemigroupsByCoclass(5,1);
[ <fp semigroup on the generators [ s1, s2 ]>,
  <fp semigroup on the generators [ s1, s2 ]>,
  <fp semigroup on the generators [ s1, s2 ]>,
  <fp semigroup on the generators [ s1, s2 ]>,
  <fp semigroup on the generators [ s1, s2 ]>,
  <fp semigroup on the generators [ s1, s2 ]>,
  <fp semigroup on the generators [ s1, s2 ]> ]
gap> NilpotentSemigroupsByCoclass(7,0);
[ <fp semigroup on the generators [ s1 ]> ]
gap> NilpotentSemigroupsByCoclass(4,2,3);
[ <fp semigroup on the generators [ s1, s2, s3 ]> ]
```

4.4 Starred Green's relations

In this section functionality around the starred Green's relations is described. The five starred Green's relations are R^*, L^*, J^*, H^*, and D^*; two elements a, b from a semigroup S are R^*-related if for all $x,y \in S^1 : xa = ya$ if and only if $xb = yb$; and a and b are L^*-related if for all $x,y \in S^1 : ax = ay$ if

and only if $bx = by$. In parallel to the classical Green's relations (**Reference: Green's Relations**) $H^* = R^* \wedge L^*$ and $D^* = R^* \vee L^*$ (but $R^* \circ L^* = L^* \circ R^*$ does *not* hold in general). To describe J^* is a bit more technical. For $a, b \in S$ one can show that b lies in $J^*(a)$, the principal *-ideal of a, if and only if there exist $c_0, c_1 \ldots, c_n \in S$ and $x_1, \ldots, x_n, y_1, \ldots, y_n \in S^1$ such that $a = c_0, b = c_n$ and $c_i D^* x_i c_{i-1} y_i$ for $1 \leq i \leq n$. Then aJ^*b if and only if both $a \in J^*(b)$ and $b \in J^*(a)$

Note that even for finite semigroups J^* does not always equal D^* (in contrast to the situation for classical Green's relations). Using Smallsemi it was shown that there exist semigroups of order 8 with $J^* \neq D^*$ [DMU13].

4.4.1 IsStarRelation

▷ `IsStarRelation(`*bin-relation*`)` (property)
▷ `IsRStarRelation(`*equiv-relation*`)` (property)
▷ `IsLStarRelation(`*equiv-relation*`)` (property)
▷ `IsJStarRelation(`*equiv-relation*`)` (property)
▷ `IsHStarRelation(`*equiv-relation*`)` (property)
▷ `IsDStarRelation(`*equiv-relation*`)` (property)

These functions return `true` if the argument is the respective type of relation and `false` otherwise.

4.4.2 RStarRelation

▷ `RStarRelation(`*semigroup*`)` (attribute)
▷ `LStarRelation(`*semigroup*`)` (attribute)
▷ `JStarRelation(`*semigroup*`)` (attribute)
▷ `DStarRelation(`*semigroup*`)` (attribute)
▷ `HStarRelation(`*semigroup*`)` (attribute)

The starred Green's relations (which are equivalence relations) are attributes of the semigroup *semigroup*.

4.4.3 RStarClass (for a semigroup and element)

▷ `RStarClass(`*S*`, a)` (operation)
▷ `LStarClass(`*S*`, a)` (operation)
▷ `DStarClass(`*S*`, a)` (operation)
▷ `JStarClass(`*S*`, a)` (operation)
▷ `HStarClass(`*S*`, a)` (operation)

Creates the $X*$-class of the element `a` in the semigroup *S* where X is one of L, R, D, J, or H.

Example

```
gap> s := SmallSemigroup(7, 280142);
<small semigroup of size 7>
gap> elm := AsList(s)[5];;
gap> jclass := JStarClass(s, elm);
{s5}
```

```
gap> AsList(jclass);
[ s2, s3, s4, s5 ]
```

4.4.4 RStarClass (for a Green's *-class)

▷ RStarClass(*C*) (attribute)
▷ LStarClass(*C*) (attribute)
▷ DStarClass(*C*) (attribute)
▷ JStarClass(*C*) (attribute)

are attributes reflecting the natural ordering over the various starred Green's classes. They return the respective class in which the given class *C* is contained, where *C* must be a class from a strictly finer relation.

Example

```
gap> s := SmallSemigroup(7, 280142);
<small semigroup of size 7>
gap> elm := AsList(s)[5];;
gap> hclass := HStarClass(s, elm);
{s5}
gap> AsList(LStarClass(hclass));
[ s5 ]
gap> AsList(RStarClass(hclass));
[ s2, s5 ]
gap> AsList(DStarClass(hclass));
[ s2, s3, s4, s5 ]
```

4.4.5 IsStarClass

▷ IsStarClass(*equiv-class*) (property)
▷ IsRStarClass(*equiv-class*) (property)
▷ IsLStarClass(*equiv-class*) (property)
▷ IsJStarClass(*equiv-class*) (property)
▷ IsHStarClass(*equiv-class*) (property)
▷ IsDStarClass(*equiv-class*) (property)

return `true` if the equivalence class *equiv-class* is a starred Green's class of any type, or of *R*, *L*, *J*, *H*, *D* type, respectively, or `false` otherwise.

4.4.6 RStarClasses

▷ RStarClasses(*semigroup*) (attribute)
▷ LStarClasses(*semigroup*) (attribute)
▷ JStarClasses(*semigroup*) (attribute)
▷ DStarClasses(*semigroup*) (attribute)
▷ HStarClasses(*semigroup*) (attribute)

return the *R*, *L*, *J*, *H*, or *D* starred Green's classes, respectively for semigroup *semigroup*. `EquivalenceClasses` for a Green's relation lead to one of these functions.

Example

```
gap> s := SmallSemigroup(6, 54);
<small semigroup of size 6>
gap> JStarClasses(s);
[ {s1}, {s2}, {s4}, {s5}, {s6} ]
```

4.5 Families of Semigroups

In this section we describe how to find semigroups in the library satisfying a given set of parameters.

The following functions have the same usage but may return different values: EnumeratorOfSmallSemigroups (4.5.2), AllSmallSemigroups (4.5.1), EnumeratorSortedOfSmallSemigroups (4.5.4), IdsOfSmallSemigroups (4.5.7), IteratorOfSmallSemigroups (4.5.11), NrSmallSemigroups (4.5.15), OneSmallSemigroup (4.5.16), PositionsOfSmallSemigroups (4.5.17), RandomSmallSemigroup (4.5.20).

The number of arguments should be odd:

- the first argument arg[1] should be a positive integer, a list of positive integers, or an enumerator or iterator of small semigroups satisfying IsEnumeratorOfSmallSemigroups (4.5.8) or IsIteratorOfSmallSemigroups (4.5.10)
- the even arguments arg[2i], if present, should be a function
- the odd arguments arg[2i+1] argument should be a possible value that can be returned by the function arg[2i].

In the case that the function is AllSmallSemigroups (4.5.1) and arg[1] is a positive integer, then the returned value is a list of all semigroups S with arg[1] elements such that arg[2i](S)=arg[2i+1].

For example, to obtain all the commutative semigroups with *3* idempotents of sizes *2* to *5* use one of EnumeratorOfSmallSemigroups (4.5.2), AllSmallSemigroups (4.5.1), EnumeratorSortedOfSmallSemigroups (4.5.4), IteratorOfSmallSemigroups (4.5.11) with argument

Example

```
[2..5], IsCommutative, true, Is3IdempotentGenerated, true
```

AllSmallSemigroups (4.5.1) returns a list of all such semigroups, EnumeratorOfSmallSemigroups (4.5.2), EnumeratorSortedOfSmallSemigroups (4.5.4), and IteratorOfSmallSemigroups (4.5.11) return an enumerator and an iterator of all such semigroups, respectively. For more information on enumerators and iterators see Enumerator (**Reference: Enumerator**), EnumeratorSorted (**Reference: EnumeratorSorted**), or Iterator (**Reference: Iterator**). The following are rules of thumb regarding the different situations when these functions should be used in order of slowest to fastest and greatest memory use to least:

- AllSmallSemigroups (4.5.1) should be used if the number of semigroups is not too large and you want to keep the created semigroups in a list.
- EnumeratorOfSmallSemigroups (4.5.2) or EnumeratorSortedOfSmallSemigroups (4.5.4) should be used when the functions in even indexed positions are those stored in the library (see PrecomputedSmallSemisInfo (4.5.19)) or you want repeatedly search the same

set of semigroups and there are too many to store in a list. Note that the enumerator stores the id numbers of its elements but not the semigroups themselves. Hence every time an element of the enumerator is required it must be recreated from the multiplication table data.

- `IteratorOfSmallSemigroups` (4.5.11) should be used if the functions in even indexed positions are not stored in the library (see `PrecomputedSmallSemisInfo` (4.5.19)) or if you just want to run through all the semigroups satisfying the specified parameters once only. Note that each new call of `IteratorOfSmallSemigroups` (4.5.11) requires GAP to recompute its elements which may be slow if the functions are user-defined or not stored in the library.

Further information on the relative virtues of these different commands can be found in Chapter 3.

As a further example, if we want to obtain a single non-simple semigroup with 7 elements and trivial automorphism group, then we would use one of the functions `OneSmallSemigroup` (4.5.16) or `RandomSmallSemigroup` (4.5.20) with argument

Example

```
7, IsSimpleSemigroup, false, x-> IsTrivial(AutomorphismGroup(x)), true
```

`OneSmallSemigroup` (4.5.16) should return an answer more quickly than `RandomSmallSemigroup` (4.5.20). Also note that `OneSmallSemigroup` (4.5.16) will always return the same semigroup, i.e. the first semigroup in the library with the given parameters.

4.5.1 AllSmallSemigroups

▷ `AllSmallSemigroups(`*arg*`)` (function)

the number of argument of this function should be odd. The first argument `arg[1]` should be a positive integer, an enumerator of small semigroups with `IsEnumeratorOfSmallSemigroups` (4.5.8), or an iterator of small semigroup with `IsIteratorOfSmallSemigroups` (4.5.10).

The even arguments `arg[2i]`, if present, should be functions, and the odd arguments `arg[2i+1]` should be a value that the preceeding function can have. For example, a typical input might be `3, IsRegularSemigroup, true`. The functions `arg[2i]` can be user defined or existing GAP functions.

Please see Section 4.5 or Chapter 3 for more details.

If `arg[1]` is a positive integer, then `AllSmallSemigroups` returns a list of all the small semigroups `S` in the library with `Size(S)=arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

If `arg[1]` is a list of positive integers, then `AllSmallSemigroups` returns a list of all the small semigroups `S` in the library with `Size(S) in arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

If `arg[1]` is an enumerator or iterator of small semigroups, then `AllSmallSemigroups` returns a list of all the small semigroups `S` in the library with `S in arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

Example

```
gap> AllSmallSemigroups(2);
[ <small semigroup of size 2>, <small semigroup of size 2>,
  <small semigroup of size 2>, <small semigroup of size 2> ]
gap> AllSmallSemigroups([2,3], IsRegularSemigroup, true,
> x-> Length(GreensRClasses(x)), 1);
[ <small semigroup of size 2>, <small semigroup of size 2>,
  <small semigroup of size 3>, <small semigroup of size 3> ]
```

```
gap> enum:=EnumeratorOfSmallSemigroups(8, IsInverseSemigroup, true,
> IsCommutativeSemigroup, true);;
gap> AllSmallSemigroups(enum, x-> Length(GreensRClasses(x)), 1);
[ <small semigroup of size 8>, <small semigroup of size 8>,
  <small semigroup of size 8> ]
gap> iter:=IteratorOfSmallSemigroups(7, x-> Length(GreensRClasses(x)), 1);;
gap> AllSmallSemigroups(iter, IsCommutative, true,
> IsSimpleSemigroup, true);
[ <small semigroup of size 7> ]
```

4.5.2 EnumeratorOfSmallSemigroups

▷ `EnumeratorOfSmallSemigroups(`*arg*`)` (function)

the number of argument of this function should be odd. The first argument `arg[1]` should be a positive integer, an enumerator of small semigroups with `IsEnumeratorOfSmallSemigroups` (4.5.8), or an iterator of small semigroup with `IsIteratorOfSmallSemigroups` (4.5.10).

The even arguments `arg[2i]`, if present, should be functions, and the odd arguments `arg[2i+1]` should be a value that the preceeding function can have. For example, a typical input might be `3, IsRegularSemigroup, true`. The functions `arg[2i]` can be user defined or existing GAP functions.

Please see Section 4.5 or Chapter 3 for more details.

If `arg[1]` is a positive integer, then `EnumeratorOfSmallSemigroups` returns an enumerator of all the small semigroups `S` in the library with `Size(S)=arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

If `arg[1]` is a list of positive integers, then `EnumeratorOfSmallSemigroups` returns an enumerator of all the small semigroups `S` in the library with `Size(S) in arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

If `arg[1]` is an enumerator or iterator of small semigroups, then `EnumeratorOfSmallSemigroups` returns an enumerator of all the small semigroups `S` in the library with `S in arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

Example

```
gap> enum:=EnumeratorOfSmallSemigroups(7);
<enumerator of semigroups of size 7>
gap> EnumeratorOfSmallSemigroups([2,3], IsRegularSemigroup, true);
<enumerator of semigroups of sizes [ 2, 3 ]>
gap> enum:=EnumeratorOfSmallSemigroups(8, IsInverseSemigroup, true,
> IsCommutativeSemigroup, true);
<enumerator of semigroups of size 8>
gap> EnumeratorOfSmallSemigroups(enum, IsCommutativeSemigroup, true,
> IsSimpleSemigroup, false);
<enumerator of semigroups of size 8>
gap> iter:=IteratorOfSmallSemigroups(8);
<iterator of semigroups of size 8>
gap> EnumeratorOfSmallSemigroups(iter, IsCommutativeSemigroup, true,
> IsSimpleSemigroup, false);
<enumerator of semigroups of size 8>
```

4.5.3 EnumeratorOfSmallSemigroupsByIds

▷ `EnumeratorOfSmallSemigroupsByIds(`*arg*`)` (operation)
▷ `EnumeratorOfSmallSemigroupsByIdsNC(`*arg*`)` (operation)

the argument of this function should be one of the following:

- a positive integer *arg[1]* and a set of positive integers less than NrSmallSemigroups (4.5.15) with argument *arg[1]*. For example, the argument *3, [1..10]* yields the first 10 semigroups with 3 elements.
- a set of positive integers *arg[1]* and a list of sets of positive integers *arg[2]* such that *x* is at most NrSmallSemigroups (4.5.15) with argument *arg[1][i]* for all *x* in *arg[2][i]*. For example, *[2,3], [[1..2],[1..10]]* yields the first 2 semigroups of size 2, and the first 10 semigroups of size 3.
- a list of id numbers, for example, *[[7,1], [6,1], [5,1]]*.

The no check version does not check that the arguments are valid and may return unpredictable results.

Example

```
gap> enum:=EnumeratorOfSmallSemigroupsByIds([[7,1],[6,1],[5,1]]);
<enumerator of semigroups of sizes [ 5, 6, 7 ]>
gap> enum:=EnumeratorOfSmallSemigroupsByIds(7, [1..1000]);
<enumerator of semigroups of size 7>
gap> enum:=EnumeratorOfSmallSemigroupsByIds([2,3], [[1..2],[1..10]]);
<enumerator of semigroups of sizes [ 2, 3 ]>
```

4.5.4 EnumeratorSortedOfSmallSemigroups

▷ `EnumeratorSortedOfSmallSemigroups(`*arg*`)` (function)

accepts the same arguments and returns the same values as EnumeratorOfSmallSemigroups (4.5.2).

4.5.5 FuncsOfSmallSemisInEnum

▷ `FuncsOfSmallSemisInEnum(`*enum*`)` (function)

returns a list of the functions and their values that were used to create the enumerator of small semigroups *enum*. If you only want the names of these functions use NamesFuncsSmallSemisInEnum (4.5.12).

Example

```
gap> enum:=EnumeratorOfSmallSemigroups([2..4], IsSimpleSemigroup, false,
> IsRegularSemigroup, true);;
gap> FuncsOfSmallSemisInEnum(enum);
[ <Property "IsRegularSemigroup">, true,
  <Property "IsSimpleSemigroup">, false ]
```

4.5.6 FuncsOfSmallSemisInIter

▷ `FuncsOfSmallSemisInIter(`*iter*`)` (function)

returns a list of the functions and their values that were used to create the iterator of small semigroups *iter*. If you only want the names of these functions use `NamesFuncsSmallSemisInIter` (4.5.13).

Example

```
gap> enum:=IteratorOfSmallSemigroups([2..4], IsSimpleSemigroup, false,
> IsRegularSemigroup, true);;
gap> FuncsOfSmallSemisInIter(enum);
[ <Property "IsRegularSemigroup">, true,
  <Property "IsSimpleSemigroup">, false ]
```

4.5.7 IdsOfSmallSemigroups

▷ `IdsOfSmallSemigroups(`*arg*`)` (function)

the number of argument of this function should be odd. The first argument `arg[1]` should be a positive integer, an enumerator of small semigroups with `IsEnumeratorOfSmallSemigroups` (4.5.8), or an iterator of small semigroup with `IsIteratorOfSmallSemigroups` (4.5.10).

The even arguments `arg[2i]`, if present, should be functions, and the odd arguments `arg[2i+1]` should be a value that the preceeding function can have. For example, a typical input might be `3, IsRegularSemigroup, true`. The functions `arg[2i]` can be user defined or existing GAP functions.

Please see Section 4.5 or Chapter 3 for more details.

If `arg[1]` is a positive integer, then `IdsOfSmallSemigroups` returns a list of the id numbers of all the small semigroups `S` in the library with `Size(S)=arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

If `arg[1]` is a list of positive integers, then `IdsOfSmallSemigroups` returns a list of the id numbers of all the small semigroups `S` in the library with `Size(S) in arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

If `arg[1]` is an enumerator or iterator of small semigroups, then `IdsOfSmallSemigroups` returns a list of the id numbers of all the small semigroups `S` in the library with `S in arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

Example

```
gap>  enum:=EnumeratorOfSmallSemigroups(5, x-> Length(GreensRClasses(x)), 1);;
gap> IdsOfSmallSemigroups(enum, IsCommutativeSemigroup, true,
> IsSimpleSemigroup, false);
[  ]
gap> IdsOfSmallSemigroups([2,3], IsRegularSemigroup, true);
[ [ 2, 2 ], [ 2, 3 ], [ 2, 4 ], [ 3, 10 ], [ 3, 11 ], [ 3, 12 ],
  [ 3, 13 ], [ 3, 14 ], [ 3, 15 ], [ 3, 16 ], [ 3, 17 ], [ 3, 18 ] ]
```

4.5.8 IsEnumeratorOfSmallSemigroups

▷ `IsEnumeratorOfSmallSemigroups(`*enum*`)` (property)

returns `true` if *enum* is an enumerator of small semigroups created using `EnumeratorOfSmallSemigroups` (4.5.2), `EnumeratorOfSmallSemigroupsByIds` (4.5.3).

Example

```
gap> enum:=EnumeratorOfSmallSemigroupsByIds([[2,1], [3,1], [4,1]]);;
gap> IsEnumeratorOfSmallSemigroups(enum);
true
```

4.5.9 IsIdSmallSemigroup

▷ `IsIdSmallSemigroup(`*arg*`)` (property)

return `true` if the *arg* is the id of a small semigroup or *`[arg[1], arg[2]]`* is the id of a small semigroup.

Example

```
gap> IsIdSmallSemigroup(8,1);
true
gap> IsIdSmallSemigroup([1,2]);
false
gap> IsIdSmallSemigroup([3,18]);
true
```

4.5.10 IsIteratorOfSmallSemigroups

▷ `IsIteratorOfSmallSemigroups(`*iter*`)` (property)

returns `true` if *iter* is an iterator of small semigroups created using `IteratorOfSmallSemigroups` (4.5.11).

Example

```
gap> iter:=IteratorOfSmallSemigroups(8);;
gap> IsIteratorOfSmallSemigroups(iter);
true
```

4.5.11 IteratorOfSmallSemigroups

▷ `IteratorOfSmallSemigroups(`*arg*`)` (function)

the number of argument of this function should be odd. The first argument `arg[1]` should be a positive integer, an enumerator of small semigroups with `IsEnumeratorOfSmallSemigroups` (4.5.8), or an iterator of small semigroup with `IsIteratorOfSmallSemigroups` (4.5.10).

The even arguments `arg[2i]`, if present, should be functions, and the odd arguments `arg[2i+1]` should be a value that the preceeding function can have. For example, a typical input might be `3, IsRegularSemigroup, true`. The functions `arg[2i]` can be user defined or existing GAP functions.

Please see Section 4.5 or Chapter 3 for more details.

If `arg[1]` is a positive integer, then `IteratorOfSmallSemigroups` returns an iterator of all the small semigroups `S` in the library with `Size(S)=arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

If `arg[1]` is a list of positive integers, then `IteratorOfSmallSemigroups` returns an iterator of all the small semigroups `S` in the library with `Size(S) in arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

If arg[1] is an enumerator or iterator of small semigroups, then IteratorOfSmallSemigroups returns an iterator of all the small semigroups S in the library with S in arg[1] and arg[2i](S)=arg[2i+1] for all i.

Example

```
gap> iter:=IteratorOfSmallSemigroups(8);
<iterator of semigroups of size 8>
gap> NextIterator(iter);
<small semigroup of size 8>
gap> IsDoneIterator(iter);
false
gap> iter:=IteratorOfSmallSemigroups([2,3], IsRegularSemigroup, true,
> x-> Length(Idempotents(x))=1, true);
<iterator of semigroups of sizes [ 2, 3 ]>
gap> NextIterator(iter);
<small semigroup of size 2>
gap> NextIterator(iter);
<small semigroup of size 3>
gap> NextIterator(iter);
fail
gap> enum:=EnumeratorOfSmallSemigroups(5, x-> Length(Idempotents(x))=1, true);
<enumerator of semigroups of size 5>
gap> iter:=IteratorOfSmallSemigroups(enum, x-> Length(GreensRClasses(x))=2, true);
<iterator of semigroups of size 5>
```

4.5.12 NamesFuncsSmallSemisInEnum

▷ NamesFuncsSmallSemisInEnum(*enum*) (function)

returns a list of the names of functions and their values that were used to create the enumerator of small semigroups *enum*. If you only want the actual functions themselves then use FuncsOfSmallSemisInEnum (4.5.5).

Example

```
gap> enum:=EnumeratorOfSmallSemigroups([2..4], IsSimpleSemigroup, false,
> IsRegularSemigroup, true);;
gap> NamesFuncsSmallSemisInEnum(enum);
[ "IsRegularSemigroup", true, "IsSimpleSemigroup", false ]
```

4.5.13 NamesFuncsSmallSemisInIter

▷ NamesFuncsSmallSemisInIter(*iter*) (attribute)

returns a list of the names of functions and their values that were used to create the iterator of small semigroups *iter*. If you only want the actual functions themselves then use FuncsOfSmallSemisInIter (4.5.6).

Example

```
gap> iter:=IteratorOfSmallSemigroups([2..4], IsSimpleSemigroup, false,
> IsRegularSemigroup, true);;
gap> NamesFuncsSmallSemisInIter(iter);
[ "IsRegularSemigroup", true, "IsSimpleSemigroup", false ]
```

4.5.14 Nr3NilpotentSemigroups

▷ Nr3NilpotentSemigroups(*n[, type]*) (function)

returns the number of 3-nilpotent semigroups on a set with *n* elements. If the optional argument *type* is given it must be one of "UpToEquivalence", "UpToIsomorphism", "SelfDual", "Commutative", "Labelled", "Labelled-Commutative". The number will be returned for the respective type of semigroup. By default *type* is "UpToEquivalence".

The function implements the formulae calculating the number of 3-nilpotent semigroups developed in [Dis10]

Example

```
gap> Nr3NilpotentSemigroups( 4 );
8
gap> Nr3NilpotentSemigroups( 9, "UpToIsomorphism" );
105931872028455
gap> Nr3NilpotentSemigroups( 9, "Labelled" );
38430603831264883632
gap> Nr3NilpotentSemigroups( 16, "SelfDual" );
49750008379418478147447102904698904559855 30
gap> Nr3NilpotentSemigroups( 19, "Commutative" );
120942706561604039207679356046247489089931699493174547676177 95
```

4.5.15 NrSmallSemigroups

▷ NrSmallSemigroups(*arg*) (function)

the number of argument of this function should be odd. The first argument arg[1] should be a positive integer, an enumerator of small semigroups with IsEnumeratorOfSmallSemigroups (4.5.8), or an iterator of small semigroup with IsIteratorOfSmallSemigroups (4.5.10).

The even arguments arg[2i], if present, should be functions, and the odd arguments arg[2i+1] should be a value that the preceeding function can have. For example, a typical input might be 3, IsRegularSemigroup, true. The functions arg[2i] can be user defined or existing GAP functions.

Please see Section 4.5 or Chapter 3 for more details.

If arg[1] is a positive integer, then NrSmallSemigroups returns the number of small semigroups S in the library with Size(S)=arg[1] and arg[2i](S)=arg[2i+1] for all i.

If arg[1] is a list of positive integers, then NrSmallSemigroups returns the number of small semigroups S in the library with Size(S) in arg[1] and arg[2i](S)=arg[2i+1] for all i.

If arg[1] is an enumerator or iterator of small semigroups, then NrSmallSemigroups returns the number of small semigroups S in the library with S in arg[1] and arg[2i](S)=arg[2i+1] for all i.

Example

```
gap> List([1..8], NrSmallSemigroups);
[ 1, 4, 18, 126, 1160, 15973, 836021, 1843120128 ]
gap> NrSmallSemigroups(8, IsCommutative, true, IsInverseSemigroup, true);
4443
gap> NrSmallSemigroups([1..8], IsCliffordSemigroup, true);
5610
gap> NrSmallSemigroups(8, IsRegularSemigroup, true,
```

```
> IsCompletelyRegularSemigroup, false);
1164
gap> NrSmallSemigroups(5, NilpotencyDegree, 3);
84
```

4.5.16 OneSmallSemigroup

▷ `OneSmallSemigroup(arg)` (function)

the number of argument of this function should be odd. The first argument `arg[1]` should be a positive integer, an enumerator of small semigroups with `IsEnumeratorOfSmallSemigroups` (4.5.8), or an iterator of small semigroup with `IsIteratorOfSmallSemigroups` (4.5.10).

The even arguments `arg[2i]`, if present, should be functions, and the odd arguments `arg[2i+1]` should be a value that the preceeding function can have. For example, a typical input might be `3, IsRegularSemigroup, true`. The functions `arg[2i]` can be user defined or existing GAP functions.

Please see Section 4.5 or Chapter 3 for more details.

If `arg[1]` is a positive integer, then `OneSmallSemigroup` returns the first small semigroup `S` in the library with `Size(S)=arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

If `arg[1]` is a list of positive integers, then `OneSmallSemigroup` returns the first small semigroup `S` in the library with `Size(S) in arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

If `arg[1]` is an enumerator or iterator of small semigroups, then `OneSmallSemigroup` returns the first small semigroup `S` in the library with `S in arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

Example

```
gap> OneSmallSemigroup(8, IsCommutative, true, IsInverseSemigroup, true);
<small semigroup of size 8>
gap> OneSmallSemigroup([1..8], IsCliffordSemigroup, true);
<small semigroup of size 1>
gap> iter:=IteratorOfSmallSemigroups(8, IsCommutative, false);
<iterator of semigroups of size 8>
gap> OneSmallSemigroup(iter);
<small semigroup of size 8>
```

4.5.17 PositionsOfSmallSemigroups

▷ `PositionsOfSmallSemigroups(arg)` (function)

the number of argument of this function should be odd. The first argument `arg[1]` should be a positive integer or an enumerator with `IsEnumeratorOfSmallSemigroups` (4.5.8), the even arguments `arg[2i]`, if present, should be functions, and the odd arguments `arg[2i+1]` should be a value that the preceeding function can have. For example, a typical input might be `3, IsRegularSemigroup, true`. The functions `arg[2i]` can be user defined or existing GAP functions. The argument can be a list `arg` with the same components as given above.

The function returns a list of the second components of the `IdSmallSemigroup` (4.1.6) of all the small semigroups `S` in the library satisfying `Size(S) in arg[1]` or `Size(S) in SizesOfSmallSemisInEnum(arg[1])` and `arg[2i](S)=arg[2i+1]` for all `i` partitioned by size of the semigroups.

Example

```
gap> PositionsOfSmallSemigroups(3);
[ [ 1 .. 18 ] ]
gap> PositionsOfSmallSemigroups(3, IsRegularSemigroup, false);
[ [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ] ]
gap> enum:=EnumeratorOfSmallSemigroups(3, IsRegularSemigroup, false);;
gap> PositionsOfSmallSemigroups(enum);
[ [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ] ]
gap> PositionsOfSmallSemigroups([1..4], IsBand, true);
[ [ 1 ], [ 3, 4 ], [ 12 .. 17 ], [ 98 .. 123 ] ]
gap> PositionsOfSmallSemigroups(enum, Is1IdempotentSemigroup, true,
> Is2GeneratedSemigroup, true, IsCliffordSemigroup, false);
[ [ 1, 2 ], [ 2, 3, 5, 6, 8, 9, 10, 12, 34, 35, 36, 97 ],
  [ 5, 20, 21, 22, 23, 26, 29, 32, 35, 54, 55, 56, 60, 61, 62, 63, 64, 65,
      152, 156, 159, 177, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
      191, 192, 193, 540, 1009, 1157, 1158 ] ]
```

4.5.18 PositionsOfSmallSemisInEnum

▷ `PositionsOfSmallSemisInEnum(`*enum*`)` (function)

returns the second components of the id numbers of the small semigroups in the enumerator of small semigroups *enum* in a list partitioned according the size of the semigroup. The same value is returned by using `PositionsOfSmallSemigroups` (4.5.17).

Example

```
gap> enum := EnumeratorOfSmallSemigroups([2..4],IsSimpleSemigroup,true);;
gap> PositionsOfSmallSemisInEnum
> (enum);
[ [ 2, 4 ], [ 17, 18 ], [ 7, 37, 52, 122, 123 ] ]
```

4.5.19 PrecomputedSmallSemisInfo

▷ `PrecomputedSmallSemisInfo` (global variable)

the global variable `PrecomputedSmallSemisInfo` contains a list of all the names of precomputed properties stored in the library. The `i`th element of the list contains the list of properties that have been precomputed for all semigroups in the library of order `i`.

Example

```
gap> PrecomputedSmallSemisInfo[3];
[ "Is1GeneratedSemigroup", "Is2GeneratedSemigroup", "Is3GeneratedSemigroup",
  "IsBand", "IsCliffordSemigroup", "IsCommutative",
  "IsCompletelyRegularSemigroup", "IsFullTransformationSemigroupCopy",
  "IsGroupAsSemigroup", "IsIdempotentGenerated", "IsInverseSemigroup",
  "IsMonoidAsSemigroup", "IsMultSemigroupOfNearRing", "IsRegularSemigroup",
  "IsSelfDualSemigroup", "IsSemigroupWithoutClosedIdempotents",
  "IsSimpleSemigroup", "IsSingularSemigroupCopy", "IsZeroSemigroup",
  "IsZeroSimpleSemigroup" ]
```

4.5.20 RandomSmallSemigroup

▷ `RandomSmallSemigroup(`*arg*`)` (function)

the number of argument of this function should be odd. The first argument `arg[1]` should be a positive integer, an enumerator of small semigroups with `IsEnumeratorOfSmallSemigroups` (4.5.8), or an iterator of small semigroup with `IsIteratorOfSmallSemigroups` (4.5.10).

The even arguments `arg[2i]`, if present, should be functions, and the odd arguments `arg[2i+1]` should be a value that the preceeding function can have. For example, a typical input might be `3, IsRegularSemigroup, true`. The functions `arg[2i]` can be user defined or existing GAP functions.

Please see Section 4.5 or Chapter 3 for more details.

If `arg[1]` is a positive integer, then `RandomSmallSemigroup` returns a random small semigroup `S` in the library with `Size(S)=arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

If `arg[1]` is a list of positive integers, then `RandomSmallSemigroup` returns the a random small semigroup `S` in the library with `Size(S) in arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

If `arg[1]` is an enumerator or iterator of small semigroups, then `RandomSmallSemigroup` returns the a random small semigroup `S` in the library with `S in arg[1]` and `arg[2i](S)=arg[2i+1]` for all `i`.

Example

```
gap> RandomSmallSemigroup(8, IsCommutative, true,
> IsInverseSemigroup, true);
<small semigroup of size 8>
gap> RandomSmallSemigroup([1..8], IsCliffordSemigroup, true);
<small semigroup of size 8>
gap> iter:=IteratorOfSmallSemigroups([1..7]);
<iterator of semigroups of size [ 1 .. 7 ]>
gap> RandomSmallSemigroup(iter);
<small semigroup of size 7>
```

4.5.21 SizesOfSmallSemisInEnum

▷ `SizesOfSmallSemisInEnum(`*enum*`)` (function)

returns the sizes of the semigroups in the enumerator of small semigroups *enum*.

Example

```
gap> enum:=EnumeratorOfSmallSemigroups([2..4], IsSimpleSemigroup, false);
<enumerator of semigroups of sizes [ 2, 3, 4 ]>
gap> SizesOfSmallSemisInEnum(enum);
[ 2, 3, 4 ]
```

4.5.22 SizesOfSmallSemisInIter

▷ `SizesOfSmallSemisInIter(`*iter*`)` (function)

returns the sizes of the semigroups in the iterator *iter* of small semigroups.

Example

```
gap> iter:=IteratorOfSmallSemigroups(7, IsCommutative, false);
<iterator of semigroups of size 7>
```

```
gap> SizesOfSmallSemisInIter(iter);
[ 7 ]
```

4.5.23 UpToIsomorphism

▷ UpToIsomorphism(*sgrps*) (operation)

takes a list *sgrps* of non-equivalent semigroups from the library as input and returns a list of non-isomorphic semigroups containing an isomorphic semigroup and an anti-isomorphic semigroup for every semigroup in *sgrps*.

Example

```
gap> UpToIsomorphism([SmallSemigroup(5,126),SmallSemigroup(6,2)]);
[ <small semigroup of size 5>, <small semigroup of size 6> ]
gap> UpToIsomorphism([SmallSemigroup(5,126),SmallSemigroup(5,3)]);
[ <small semigroup of size 5>, <small semigroup of size 5>,
  <semigroup with 5 generators> ]
```

References

[BEO02] H. U. Besche, B. Eick, and E. A. O'Brien. *The Small Groups Library - a GAP package*, 2002. 4

[Dis10] A. Distler. *Classification and Enumeration of finite Semigroups*. PhD thesis, St Andrews University, 2010. 44

[Dis14] Andreas Distler. Finite nilpotent semigroups of small coclass. *Communications in Algebra*, 42(3):1136–1150, 2014. 34

[DMU13] Andreas Distler, Victor Maltcev, and Abdullahi Umar. J*=D* need not hold in finite semigroups. *submitted*, 2013. http://arxiv.org/abs/1302.0985. 35

[For55] George E. Forsythe. SWAC computes 126 distinct semigroups of order 4. *Proc. Amer. Math. Soc.*, 6:443–447, 1955. 8

[GJM06] Ian P. Gent, Christopher Jefferson, and Ian Miguel. Minion: A fast scalable constraint solver. In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *The European Conference on Artificial Intelligence 2006 (ECAI 06)*, pages 98–102. IOS Press, 2006. 2, 8

[JW77] H. Jürgensen and P. Wick. Die Halbgruppen der Ordnungen ≤ 7. *Semigroup Forum*, 14(1):69–79, 1977. 8

[Ple67] Robert J. Plemmons. There are 15973 semigroups of order 6. *Math. Algorithms*, 2:2–17, 1967. 8

[SYT94] S. Satoh, K. Yama, and M. Tokizawa. Semigroups of order 8. *Semigroup Forum*, 49(1):7–29, 1994. 8

[Tam54] Takayuki Tamura. Notes on finite semigroups and determination of semigroups of order 4. *J. Gakugei. Tokushima Univ. Math.*, 5:17–27, 1954. 8

[THA+55] Kazutoshi Tetsuya, Takao Hashimoto, Tadao Akazawa, Ryoichi Shibata, Tadashi Inui, and Takayuki Tamura. All semigroups of order at most 5. *J. Gakugei Tokushima Univ. Nat. Sci. Math.*, 6:19–39. Errata on loose, unpaginated sheet, 1955. 8

Index