The smallsemi Package – Development Manual

Version 0.3

A. Distler
J. D. Mitchell

A. Distler — Email: a.distler@tu-bs.de
— Address: Mathematical Institute
North Haugh
St Andrews, Fife
KY16 9SS
Scotland, UK

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

Colophon

Contents

Chapter 1

small.g*

1.1 small.g*

1.1.1 SmallSemigroupCreator

◊ SmallSemigroupCreator(*table*) (function)

returns a small semigroup *s* with multiplication table *table*. That is, an element in the category IsSmallSemigroup (**??**) with AsSSortedList (AsSSortedList???), GeneratorsOfSemigroup (GeneratorsOfSemigroup???), Size (Size???), and MultiplicationTable (MultiplicationTable???) with the property IsAssociative (IsAssociative???) set to true.

Although this function can be used to create semigroups in the category IsSmallSemigroup (**??**) where *table* is not a table in the library this may cause problems and there is no reason to do it!

If you want to create semigroups from multiplication table, then use either SemigroupByMultiplicationTableNC (**??**) if you know the table is associative, or MagmaByMultiplicationTable (MagmaByMultiplicationTable???) if you do not know.

Example

```
gap> RecoverMultiplicationTable(5, 1000);
[ [ 1, 1, 1, 1, 1 ], [ 1, 1, 1, 1, 1 ], [ 1, 2, 3, 4, 5 ], [ 1, 2, 4, 5, 3 ],
[ 1, 2, 5, 3, 4 ] ]
gap> SmallSemigroupCreator(last);
<small semigroup of size 5>
```

1.1.2 SmallSemigroupEltFamily

◊ SmallSemigroupEltFamily (global variable)

SmallSemigroupEltFamily is a global variable containing the family of elements satisfying *IsSmallSemigroupElt*.

The value of *SmallSemigroupEltFamily* is installed when smallsemi is loaded. This is done to avoid the cost of repeatedly creating a new family when, say, running through the semigroups of order 8.

Example

```
gap> SmallSemigroupEltFamily;
NewFamily( "SmallSemigroupEltFamily", [ 2201 ], [ 35, 36, 38, 114, 117, 120, 2201 ] )
```

1.1.3 SmallSemigroupEltType

◊ SmallSemigroupEltType (global variable)

SmallSemigroupEltType is a global variable containing the type of *IsSmallSemigroupElt*.

The value of *SmallSemigroupEltType* is installed when smallsemi is loaded. This is done to avoid the cost of repeatedly creating a new family when, say, running through the semigroups of order 8.

Example

```
gap> SmallSemigroupEltType;
NewType( NewFamily( "SmallSemigroupEltFamily", [ 2201 ],
[ 35, 36, 38, 114, 117, 120, 2201 ] ), [ 35, 36, 38, 114, 117, 120, 2201 ] )
```

1.1.4 SmallSemigroupType

◊ SmallSemigroupType (global variable)

SmallSemigroupType is a global variable containing the type of the collections family of *IsSmallSemigroupEltFamily*.

The value of *SmallSemigroupType* is installed when smallsemi is loaded. This is done to avoid the cost of repeatedly creating a new family when, say, running through the semigroups of order 8.

Example

```
gap> SmallSemigroupType;
NewType( NewFamily( "CollectionsFamily(...)", [ 52 ],
[ 51, 52, 114, 115, 117, 118, 121, 133 ] ),
[ 36, 38, 51, 52, 90, 91, 114, 115, 117, 118, 121, 133, 196, 420, 431, 432,
  2200 ] )
```

1.1.5 DATA2TO7

◊ DATA2TO7 (global variable)

is a list containing the raw data of the multiplication tables of semigroups of sizes *2* to *7*. The *i*-th entry is a list of strings from which the multiplication tables of semigroups of size $i+1$ can be recovered.

The *i*-th entry is bound after the first call of RecoverMultiplicationTable (**smallsemi: RecoverMultiplicationTable**) with argument *i+1, j* for some valid *j*. The function UnloadSmallsemiData (**smallsemi: UnloadSmallsemiData**) will unbind all entries.

Example

```
gap> IsBound(DATA2TO7[1]);
false
gap> RecoverMultiplicationTable(2,1);;
gap> DATA2TO7[1];
[ "0100", "0101", "0011" ]
gap> UnloadSmallsemiData(true);
gap> DATA2TO7;
[  ]
```

1.1.6 DATA8

◊ DATA8 (global variable)

is a list containing the raw data of the multiplication tables of semigroups of size *8*. The *i*-th entry is a list of strings from which it is possible to recover the multiplication tables of semigroups with diagonal equal to the *i*-th entry in the component diags of the record MOREDATA2TO8 (1.1.12).

At most one entry of the list is bound at a time. The initial value is the empty list. The variable is flushable.

1.1.7 BLUEPRINT_MATS

◊ BLUEPRINT_MATS (global variable)

see GENERATE_BLUEPRINT_MATS (1.1.9).

Example

```
gap> Display( BLUEPRINT_MATS[3] );
[ [  1,  1,  1,  1,  1,  1,  1,  1 ],
  [  1,  1,  1,  1,  1,  1,  1,  1 ],
  [  1,  1,  1,  1,  1,  1,  1,  1 ],
  [  1,  1,  1 ],
  [  1,  1,  1 ],
  [  1,  1,  1 ],
  [  1,  1,  1 ],
  [  1,  1,  1 ] ]
```

1.1.8 3NIL_DATA

◊ 3NIL_DATA (global variable)

is a record carrying data to restore 3-nilpotent semigroups of size 8.

At the time smallsemi is loaded only the component diag is bound and has the value fail. This changes when a 3-nilpotent semigroup of size 8 is called. Then diag becomes a list element from the component 3nildiags of the global variable MOREDATA2TO8 (1.1.12) corresponding to the diagonal of the last called 3-nilpotent semigroup of size 8.

The other components are strlist, a list of strings carrying the information about the entries of the stored multiplication tables; positions, a list of integers, the positions of the stored solutions relative to the first one with the same diagonal and next, an integer storing which position was last called for.

Example

```
gap> LoadPackage("smallsemi");
true
gap> 3NIL_DATA;
rec( diag := fail )
gap> SmallSemigroup( 8, NrSmallSemigroups(8)-2 );;
gap> 3NIL_DATA;
rec( diag := [ 2, 3 ], strlist := [ "0013", "0313" ],
  positions := [ 1, 3, 4, 7 ], next := 4 )
```

1.1.9 GENERATE_BLUEPRINT_MATS

◊ GENERATE_BLUEPRINT_MATS() (function)

generates a list of matrices bound for k in $\{2,...,7\}$ such that the k-th entry has k 'zero' rows and columns. To be stored in the variable BLUEPRINT_MATS.

1.1.10 READ_3NIL_DATA

◊ READ_3NIL_DATA(*diag*) (function)

reads data to recover multiplication tables of 3-nilpotent semigroups of size 8 into the global variable 3NIL_DATA (1.1.8). The data to be read is determined by *diag*. (All information for multiplication tables of which *diag* is the part of the diagonal belonging to the non zero rows and columns.) Is is assumed that *diag* is an element in the component 3nildiags of the record MOREDATA2TO8 (1.1.12)

1.1.11 READ_MOREDATA2TO8

◊ READ_MOREDATA2TO8(*S*) (function)

reads the precomputed information stored in the files infon.g for n in $\{1,...,8\}$ into the variable MOREDATA2TO8.

1.1.12 MOREDATA2TO8

◊ MOREDATA2TO8 (global variable)

contains the precomputed information stored in the files infon.g for n in $\{1,...,8\}$ in a list where the nth entry is a record with components named after the function values they store. For example, to retrieve the stored value of the function MinimalGeneratorsOfSemigroup for a semigroup S of size 5 do

Example

```
MOREDATA2TO8[5].MinimalGeneratorsOfSemigroup[IdSmallSemigroup(S)[2]];
```

1.1.13 InfoSmallsemiEnums

◊ InfoSmallsemiEnums (info class)

is an info class (see (Info Functions???)) for debugging the smallsemi file enums.gi.

Chapter 2

properties.g*

2.1 properties.g*

2.1.1 SMALLSEMI_ALWAYS_FALSE

◊ SMALLSEMI_ALWAYS_FALSE (global variable)

is a global variable whose value is a list of strings *str* of names of properties or attributes that are always `false` for a small semigroup.

For example, `IsFullTransformationSemigroup` (IsFullTransformationSemigroup???) is always `false` for small semigroups but `IsFullTransformationSemigroupCopy` (**smallsemi: IsFullTransformationSemigroupCopy**) can be `true`.

Example

```
gap> SMALLSEMI_ALWAYS_FALSE;
[ "IsFullTransformationSemigroup", "IsSingularSemigroup" ]
```

2.1.2 SMALLSEMI_EQUIV

◊ SMALLSEMI_EQUIV (global variable)

is a global variable whose value is a list of pairs *P* of functions and values where *P[1]* is a property and value which is equivalent to the properties and values in *P[2]* for small semigroups.

For example, `IsMonogenicSemigroup` (**smallsemi: IsMonogenicSemigroup**) implies *Is1GeneratedSemigroup* for all semigroups and is hence a synonym and the pair *[[IsMonogenicSemigroup, true], [Is1GeneratedSemigroup, true]]* does not need to be installed in *SMALLSEMI_EQUIV*. On the other hand, `IsCompletelySimpleSemigroup` (**smallsemi: IsCompletelySimpleSemigroup**) only holds for `IsSimpleSemigroup` (IsSimpleSemigroup???) and `IsFinite` (IsFinite???) and hence is not a synonym and the pairs *[[IsCompletelySimpleSemigroup, true], [IsSimpleSemigroup, true]]* and *[[IsCompletelySimpleSemigroup, false], [IsSimpleSemigroup, false]]* must be entered in *SMALLSEMI_EQUIV*. Note that *[[IsOrthodoxSemigroup, true], [IsRegularSemi* can be entered in *SMALLSEMI_EQUIV* but currently it is not possible to have any pair in *SMALLSEMI_EQUIV* with first entry *[IsOrthodoxSemigroup, false]*.

Also note that if *P* is an entry in *SMALLSEMI_EQUIV*, then *P[1]* must have length 2.

The reason for doing all of this is so that when `EnumeratorOfSmallSemigroups` (**smallsemi: EnumeratorOfSmallSemigroups**) is called with argument *IsCompletelySimpleSemigroup* there is no component in the record in the info file with the name *"IsCompletelySimpleSemigroup"* and so this name is provided by *SMALLSEMI_EQUIV*. If two properties are synonymous, then *NAME_FUNC* has the same value for both and so it is only necessary to store a component with that one name and hence not necessary to put a pair in *SMALLSEMI_EQUIV*.

The name of the component in the info file should be in the second component of a pair in *SMALLSEMI_EQUIV*

Example

```
gap> SMALLSEMI_EQUIV;
[ [ "IsCompletelySimpleSemigroup", "IsSimpleSemigroup" ],
[ "IsCommutativeSemigroup", "IsCommutative" ],
[ "IsNilpotent", "IsNilpotentSemigroup" ] ]
```

2.1.3 STORED_INFO

◊ `STORED_INFO(n, str)` (function)

returns the value of the component of the record `MOREDATA2TO8[n]` with name the string `str`. This is equivalent to doing the following.

Example

```
MOREDATA2TO8[n].(name);
```

Chapter 3

enum.g*

3.1 enum.g*

3.1.1 EmptyEnumeratorOfSmallSemigroups

◊ `EmptyEnumeratorOfSmallSemigroups()` (function)

the argument should be empty and the returned enumerator is too.

Example

```
gap> EmptyEnumeratorOfSmallSemigroups();
<empty enumerator of semigroups>
```

3.1.2 EmptyIteratorOfSmallSemigroups

◊ `EmptyIteratorOfSmallSemigroups()` (function)

the argument should be empty and the returned iterator is too.

Example

```
gap> EmptyIteratorOfSmallSemigroups();
<empty iterator of semigroups>
```

3.1.3 SMALLSEMI_ARG_OK

◊ `SMALLSEMI_ARG_OK(`*`arg`*`)` (function)

checks that the argument *arg* is valid for any of the functions `EnumeratorOfSmallSemigroups` (**smallsemi: EnumeratorOfSmallSemigroups**), `AllSmallSemigroups` (**smallsemi: AllSmallSemigroups**), `EnumeratorSortedOfSmallSemigroups` (**smallsemi: EnumeratorSortedOfSmallSemigroups**), `IdsOfSmallSemigroups` (**smallsemi: IdsOfSmallSemigroups**), `IteratorOfSmallSemigroups` (**smallsemi: IteratorOfSmallSemigroups**), `NrSmallSemigroups` (**smallsemi: NrSmallSemigroups**), `OneSmallSemigroup` (**smallsemi: OneSmallSemigroup**), `PositionsOfSmallSemigroups` (**smallsemi: PositionsOfSmallSemigroups**), `RandomSmallSemigroup` (**smallsemi: RandomSmallSemigroup**).

Currently a valid argument is one with:

- odd length
- *arg[1]* is a positive integer between 1 and 8, a list of positive integers between 1 and 8, an enumerator of small semigroups or an iterator of small semigroups
- *arg[2i]* (the even indexed arguments) should be functions.

3.1.4 SMALLSEMI_CAN_CREATE_ENUM_NC

◊ SMALLSEMI_CAN_CREATE_ENUM_NC(*arg*) (function)

checks that the argument *arg* can be used to produce an enumerator. This function does not check SMALLSEMI_ARG_OK (3.1.3) is true with argument *arg* and it is assumed that *arg* is of this form.

Currently a valid argument is one with:

- the maximum size of semigroup satisfying *arg* at most 7; OR
- the maximum size of semigroup satisfying *arg* equal 8 and there exists i such that *arg[2i]* in PrecomputedSmallSemisInfo (**smallsemi: PrecomputedSmallSemisInfo**)[8] and *arg[2i+1]* is true.

The reason for this is that on a 32-bit computer the maximum length of a list is smaller than the number of semigroups with 8 elements. Enumerators use lists of id numbers to specify their elements and so it is not currently possible to create arbitrary enumerators of small semigroups containing semigroups with 8 elements.

3.1.5 SMALLSEMI_CONVERT_ARG_NC

◊ SMALLSEMI_CONVERT_ARG_NC(*arg*) (function)

arg is assumed to satisfy SMALLSEMI_ARG_OK (3.1.3) (arg)=true but this is not checked. SMALLSEMI_CONVERT_ARG_NC replaces every function *arg[2i]* by an equivalent function in PrecomputedSmallSemisInfo (**smallsemi: PrecomputedSmallSemisInfo**) if it exists.

See SMALLSEMI_EQUIV (2.1.2) for more details.

Example

```
gap> SMALLSEMI_CONVERT_ARG_NC(5, IsCommutativeSemigroup, true);
[ 5, <Operation "IsCommutative">, true ]
gap> SMALLSEMI_CONVERT_ARG_NC(7, Is4GeneratedSemigroup, true);
[ 7, <Operation "Is1GeneratedSemigroup">, false,
<Operation "Is2GeneratedSemigroup">, false,
<Operation "Is3GeneratedSemigroup">, false,
<Operation "Is5GeneratedSemigroup">, false,
<Operation "Is6GeneratedSemigroup">, false,
<Operation "Is7GeneratedSemigroup">, false ]
gap> SMALLSEMI_CONVERT_ARG_NC(5, IsCommutative, true);
[ 5, <Operation "IsCommutative">, true ]
```

3.1.6 SMALLSEMI_CREATE_ENUM

◊ SMALLSEMI_CREATE_ENUM(*source, positions, names*) (function)

- *source* should be a positive integer between 1 and 8, a list of positive integers between 1 and 8, an enumerator of small semigroups, or an iterator of small semigroups;
- *positions* should be the list such that *positions[i]* is the list of second components of id numbers ## of small semigroups in the enumerator we are creating
- *names* is the list of functions and values being used to create the enumerator. It is not checked if it is possible to create an enumerator ## using *Concatenation([source],names)* as an argument. See SMALLSEMI_CAN_CREATE_ENUM_NC (3.1.4) for more details.

SMALLSEMI_CREATE_ENUM returns the same value as EnumeratorOfSmallSemigroupsByIds (**smallsemi: EnumeratorOfSmallSemigroupsByIds**) but here the attributes FuncsOfSmallSemisInEnum (**smallsemi: FuncsOfSmallSemisInEnum**) and NamesFuncsSmallSemisInEnum (**smallsemi: NamesFuncsSmallSemisInEnum**) are set according to the argument *names*.

Elements of enumerators creating using SMALLSEMI_CREATE_ENUM should have any operation in *names* set to the value specified when the enumerator was created. That is, it should not be necessary to recompute this information.

3.1.7 SMALLSEMI_ENTAB

◊ SMALLSEMI_ENTAB(*str*) (function)

returns the string *str* with SMALLSEMI_TAB_LEVEL (3.1.12) characters '>' and a space juxtaposed at the beginning.

3.1.8 SMALLSEMI_SORT_ARG_NC

◊ SMALLSEMI_SORT_ARG_NC(*arg*) (function)

arg is assumed to satisfy SMALLSEMI_ARG_OK (3.1.3) (arg)=true but this is not checked. SMALLSEMI_SORT_ARG_NC sorts *arg* so that the functions *arg[2i]* where *arg[2i+1]* is true come at the start, and then the arguments *arg[2i]* are ordered alphabetically.

3.1.9 SMALLSEMI_STRIP_ARG

◊ SMALLSEMI_STRIP_ARG(*arg*) (function)

returns *arg* or *arg[1]* if *arg* is a list containing an argument in its first position. This is required as an arg is not input as a list but occurs as a list in the function where it is used. Hence passing the arg to another function passes a list rather than the correct argument.

3.1.10 SMALLSEMI_RETURN

◊ SMALLSEMI_RETURN(*arg*) (function)

Example

3.1.11 SMALLSEMI_RS

◊ SMALLSEMI_RS (global variable)

Example

3.1.12 SMALLSEMI_TAB_LEVEL

◊ SMALLSEMI_TAB_LEVEL(*arg*) (function)

Example

Index