# SgpViz

## A GAP package for semigroup visualisation

Version 0.999.6

**Manuel Delgado**
**José João Morais**

**Manuel Delgado** Email: mdelgado@fc.up.pt
Homepage: https://www.fc.up.pt/cmup/mdelgado

## Copyright

## Acknowledgements

The first author acknowledges financial support of FCT, through the *Centro de Matemática da Universidade do Porto*.

The second author acknowledges financial support of FCT and the POCTI program through a scholarship given by *Centro de Matemática da Universidade do Porto*.

Both authors acknowledge Jorge Almeida, Vítor H. Fernandes and Pedro Silva for many helpful discussions and comments.

CONCERNING MAINTENANCE:

The maintainer wants to acknowledge partial support by:

FCT - Fundação para a Ciência e a Tecnologia under the project PTDC/MAT/65481/2006

*Centro de Matemática da Universidade do Porto* (CMUP), funded by the European Regional Development Fund through the programme COMPETE and by the Portuguese Government through the FCT project PEst-C/MAT/UI0144/2011.

CMUP (UID/MAT/00144/2013 and UID/MAT/00144/2019), which is funded by FCT (Portugal) with national (MEC) and European structural funds through the programs FEDER, under the partnership agreement PT2020.

Furthermore, the maintainer wants to thank the organisers of `GAPDays` in their several editions, as well as several people (for advises, giving feedback, etc). Among them I would like to refer: Max Horn, James Mitchel, Jan Philipp Wächter, João Araújo, Alfredo Costa and Teresa Melo.

## Colophon

This manual describes the GAP package SgpViz, Version 0.999.6, for visualising finite semigroups.

Since Version 0.998 (released in 2008), the package is maintained by the first author.

The present package is superseded by the GAP package *semigroups*, by James Mitchel, in what concerns some aspects of semigroup visualisation. We strongly recommend the usage of that package, unless you find useful specific tools available in SgpViz but not in *semigroups*.

Bug reports, suggestions and comments are, of course, welcome. Please use the email address `mdelgado@fc.up.pt` to this effect.

If you have benefited from the use of the SgpViz GAP package in your research, please cite it in addition to GAP itself, following the scheme proposed in `https://www.gap-system.org/Contacts/cite.html`.

# Contents

# Chapter 1

# Introduction

The aim of this package is to turn GAP more user-friendly, at least for semigroup theorists. It requires the usage of external programs as is the case of `graphviz` [DEG+02], a software for drawing graphs developed at AT & T Labs, that can be obtained at `https://www.graphviz.org/`. It is used not only to draw right Cayley graphs of finite semigroups and Schüzenberger graphs of finite inverse semigroups but also to visualise in the usual way the egg-box picture of a D-classe of a finite semigroup.

IMPORTANT NOTE: The version of `graphviz` to install must be greater or equal to 1.16.

Tcl/Tk should also be available in order to run the graphical interfaces
    XAutomaton and XSemigroup
used to specify automata and semigroups.

WARNING: the use of XAutomaton and XSemigroup is intended only for simple examples. After its use one may have to start another GAP session.

# Chapter 2

# Basics

We give some examples of semigroups to be used later. We also describe some basic functions that are not directly available from GAP, but are useful for the purposes of this package.

## 2.1 Examples

These are some examples of semigroups that will be used through this manual

```
Example
gap> f := FreeMonoid("a","b");
<free monoid on the generators [ a, b ]>
gap> a := GeneratorsOfMonoid( f )[ 1 ];;
gap> b := GeneratorsOfMonoid( f )[ 2 ];;
gap> r:=[[a^3,a^2],
> [a^2*b,a^2],
> [b*a^2,a^2],
> [b^2,a^2],
> [a*b*a,a],
> [b*a*b,b] ];
[ [ a^3, a^2 ], [ a^2*b, a^2 ], [ b*a^2, a^2 ], [ b^2, a^2 ], [ a*b*a, a ],
  [ b*a*b, b ] ]
gap> b21:= f/r;
<fp monoid on the generators [ a, b ]>
```

```
Example
gap> g0:=Transformation([4,1,2,4]);;
gap> g1:=Transformation([1,3,4,4]);;
gap> g2:=Transformation([2,4,3,4]);;
gap> poi3:= Monoid(g0,g1,g2);
<monoid with 3 generators>
```

## 2.2 Some attributes

These functions are semigroup attributes that get stored once computed.

### 2.2.1 HasCommutingIdempotents

▷ `HasCommutingIdempotents(M)` (attribute)

Tests whether the idempotents of the semigroup `M` commute.

### 2.2.2 IsInverseSemigroup

▷ `IsInverseSemigroup(S)` (attribute)

Tests whether a finite semigroup `S` is inverse. It is well-known that it suffices to test whether the idempotents of `S` commute and `S` is regular. The function `IsRegularSemigroup` is part of GAP.

## 2.3 Some basic functions

### 2.3.1 PartialTransformation

▷ `PartialTransformation(L)` (function)

A partial transformation is a partial function of a set of integers of the form $\{1,\ldots,n\}$. It is given by means of the list of images `L`. When an element has no image, we write 0. Returns a full transformation on a set with one more element that acts like a zero.

Example
```
gap> PartialTransformation([2,0,4,0]);
Transformation( [ 2, 5, 4, 5, 5 ] )
```

### 2.3.2 ReduceNumberOfGenerators

▷ `ReduceNumberOfGenerators(L)` (function)

Given a subset `L` of the generators of a semigroup, returns a list of generators of the same semigroup but possibly with less elements than `L`.

### 2.3.3 SemigroupFactorization

▷ `SemigroupFactorization(S, L)` (function)

`L` is an element (or list of elements) of the semigroup `S`. Returns a minimal factorization on the generators of `S` of the element(s) of `L`. Works only for transformation semigroups.

Example
```
gap> el1 := Transformation( [ 2, 3, 4, 4 ] );;
gap> el2 := Transformation( [ 2, 4, 3, 4 ] );;
gap> f1 := SemigroupFactorization(poi3,el1);
[ [ Transformation( [ 1, 3, 4, 4 ] ), Transformation( [ 2, 4, 3, 4 ] ) ] ]
gap> f1[1][1] * f1[1][2] = el1;
true
gap> SemigroupFactorization(poi3,[el1,el2]);
```

```
[ [ Transformation( [ 1, 3, 4, 4 ] ), Transformation( [ 2, 4, 3, 4 ] ) ],
  [ Transformation( [ 2, 4, 3, 4 ] ) ] ]
```

### 2.3.4 GrahamBlocks

▷ `GrahamBlocks(`*`mat`*`)` (function)

*`mat`* is a matrix as displayed by `DisplayEggBoxOfDClass(D);` of a regular D-class `D`. This function outputs a list `[gmat, phi]` where `gmat` is *`mat`* in Graham's blocks form and `phi` maps H-classes of `gmat` to the corresponding ones of *`mat`*, i.e., `phi[i][j] = [i',j']` where `mat[i'][j'] = gmat[i][j]`. If the argument to this function is not a matrix corresponding to a regular D-class, the function may abort in error.

Example
```
gap> p1 := PartialTransformation([6,2,0,0,2,6,0,0,10,10,0,0]);;
gap> p2 := PartialTransformation([0,0,1,5,0,0,5,9,0,0,9,1]);;
gap> p3 := PartialTransformation([0,0,3,3,0,0,7,7,0,0,11,11]);;
gap> p4 := PartialTransformation([4,4,0,0,8,8,0,0,12,12,0,0]);;
gap> css3:=Semigroup(p1,p2,p3,p4);
<transformation semigroup of degree 13 with 4 generators>
gap> el := Elements(css3)[8];;
gap> D := GreensDClassOfElement(css3, el);;
gap> IsRegularDClass(D);
true
gap> DisplayEggBoxOfDClass(D);
[ [  1,  1,  0,  0 ],
  [  1,  1,  0,  0 ],
  [  0,  0,  1,  1 ],
  [  0,  0,  1,  1 ] ]
gap> mat := [ [  1,  0,  1,  0 ],
>    [  0,  1,  0,  1 ],
>    [  0,  1,  0,  1 ],
>    [  1,  0,  1,  0 ] ];;
gap> res := GrahamBlocks(mat);;
gap> PrintArray(res[1]);
[ [  1,  1,  0,  0 ],
  [  1,  1,  0,  0 ],
  [  0,  0,  1,  1 ],
  [  0,  0,  1,  1 ] ]
gap> PrintArray(res[2]);
[ [  [ 1, 1 ],  [ 1, 3 ],  [ 1, 2 ],  [ 1, 4 ] ],
  [  [ 4, 1 ],  [ 4, 3 ],  [ 4, 2 ],  [ 4, 4 ] ],
  [  [ 2, 1 ],  [ 2, 3 ],  [ 2, 2 ],  [ 2, 4 ] ],
  [  [ 3, 1 ],  [ 3, 3 ],  [ 3, 2 ],  [ 3, 4 ] ] ]
```

## 2.4 Cayley graphs

### 2.4.1 RightCayleyGraphAsAutomaton

▷ `RightCayleyGraphAsAutomaton(`*`S`*`)` (function)

Computes the right Cayley graph of a finite monoid or semigroup `S`. It uses the GAP buit-in function `CayleyGraphSemigroup` to compute the Cayley Graph and returns it as an automaton without initial nor final states. (In this automaton state `i` represents the element `Elements(S)[i]`.) The Automata package is used to this effect.

Example
```
gap> rcg := RightCayleyGraphAsAutomaton(b21);
< deterministic automaton on 2 letters with 6 states >
gap> Display(rcg);
   |  1  2  3  4  5  6
-----------------------
 a |  2  4  6  4  2  4
 b |  3  5  4  4  4  3
Initial state:   [ ]
Accepting state: [ ]
```

### 2.4.2 RightCayleyGraphMonoidAsAutomaton

▷ `RightCayleyGraphMonoidAsAutomaton(S)` (function)

This function is a synonym of `RightCayleyGraphAsAutomaton` (2.4.1).

# Chapter 3

# Drawings of semigroups

There are some pictures that may give a lot of information about a semigroup. This is the case of the egg-box picture of the D-classes, the right Cayley graph of a finite monoid and the Schützenberger graphs of a finite inverse monoid.

## 3.1 Drawing the D-class of an element of a semigroup

### 3.1.1 DrawDClassOfElement

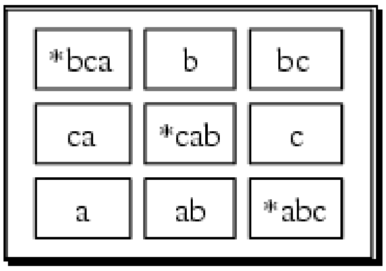
▷ `DrawDClassOfElement(arg)` (function)

This function uses `DotForDrawingDClassOfElement` (3.1.2) to compute the dot code to produce the image that is then displayed. It takes as arguments a semigroup followed by a transformation which is the element whose D-class will be drawn. Optionally we can then specify n lists of elements and the elements of each list will be drawn in different colours. Finally, if the last argument is the integer `1` then the elements will appear as transformations, otherwise they will appear as words. The idempotents will be marked with a * before them.

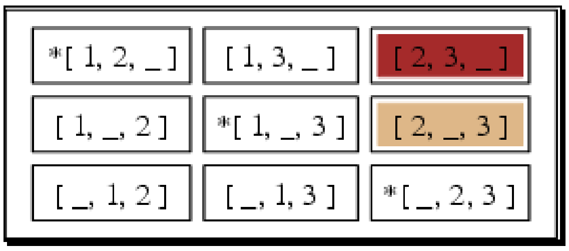
This last optional argument may also be the integer 2 and in this case the elements will appear as integers, where `i` represents the element `Elements(S)[i]`.

Example
```
gap> DrawDClassOfElement(poi3, Transformation([1,4,3,4]));
gap> DrawDClassOfElement(poi3, Transformation([1,4,3,4]),1);
gap> DrawDClassOfElement(poi3, Transformation([1,4,3,4]),
 [Transformation( [ 2, 3, 4, 4 ] )],1);
gap> DrawDClassOfElement(poi3, Transformation([1,4,3,4]),
 [Transformation( [ 2, 3, 4, 4 ] ), Transformation( [ 2, 4, 3, 4 ] )],
 [Transformation( [ 2, 4, 3, 4 ] )],1);
gap> DrawDClassOfElement(poi3, Transformation([1,4,3,4]),
 [Transformation( [ 2, 4, 3, 4 ] )],"Dclass",1);
```

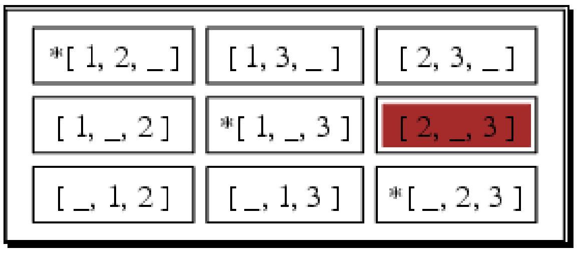
This is the image produced by the first command in the previous example:



This is the image produced by the fourth command in the previous example:



This is the image produced by the last command in the previous example:



### 3.1.2 DotForDrawingDClassOfElement

▷ `DotForDrawingDClassOfElement(`*arg*`)` (function)

This function computes the dot code that can be used to produce a drawing for the D-class of an element of a semigroup. This can be done by using the function `DrawDClassOfElement` (3.1.1) (if the system is properly configured) or by the user in some independent way. The arguments and options are the same than those of `DrawDClassOfElement` (3.1.1).

Example

```
gap> DotForDrawingDClassOfElement(poi3,Transformation([1,4,3,4]));
"digraph  DClassOfElement {\ngraph [center=yes,ordering=out];\nnode [shape=pla\
intext];\nedge [color=cornflowerblue,arrowhead=none];\n1 [label=<\n<TABLE BORD\
ER=\"0\" CELLBORDER=\"0\" CELLPADDING=\"0\" CELLSPACING=\"0\" PORT=\"1\">\n<TR\
><TD BORDER=\"0\"><TABLE CELLSPACING=\"0\"><TR><TD BGCOLOR=\"white\" BORDER=\"\
0\">*abc</TD></TR>\n</TABLE></TD><TD BORDER=\"0\"><TABLE CELLSPACING=\"0\"><TR\
><TD BGCOLOR=\"white\" BORDER=\"0\">a</TD></TR>\n</TABLE></TD><TD BORDER=\"0\"\
><TABLE CELLSPACING=\"0\"><TR><TD BGCOLOR=\"white\" BORDER=\"0\">ab</TD></TR>\
\n</TABLE></TD></TR>\n<TR><TD BORDER=\"0\"><TABLE CELLSPACING=\"0\"><TR><TD BG\
COLOR=\"white\" BORDER=\"0\">bc</TD></TR>\n</TABLE></TD><TD BORDER=\"0\"><TABL\
E CELLSPACING=\"0\"><TR><TD BGCOLOR=\"white\" BORDER=\"0\">*bca</TD></TR>\n</T\
ABLE></TD><TD BORDER=\"0\"><TABLE CELLSPACING=\"0\"><TR><TD BGCOLOR=\"white\" \
BORDER=\"0\">b</TD></TR>\n</TABLE></TD></TR>\n<TR><TD BORDER=\"0\"><TABLE CELL\
SPACING=\"0\"><TR><TD BGCOLOR=\"white\" BORDER=\"0\">c</TD></TR>\n</TABLE></TD\
><TD BORDER=\"0\"><TABLE CELLSPACING=\"0\"><TR><TD BGCOLOR=\"white\" BORDER=\"\
0\">ca</TD></TR>\n</TABLE></TD><TD BORDER=\"0\"><TABLE CELLSPACING=\"0\"><TR><\
TD BGCOLOR=\"white\" BORDER=\"0\">*cab</TD></TR>\n</TABLE></TD></TR>\n</TABLE>\
>];\n}\n"
```

By using Print (or PrinTo, if one wants to print to a file) the string is displayed as follows:

Example
```
gap> Print(last);
digraph  DClassOfElement {
graph [center=yes,ordering=out];
node [shape=plaintext];
edge [color=cornflowerblue,arrowhead=none];
1 [label=<
<TABLE BORDER="0" CELLBORDER="0" CELLPADDING="0" CELLSPACING="0" PORT="1">
<TR><TD BORDER="0"><TABLE CELLSPACING="0"><TR><TD BGCOLOR="white" BORDER="0">*\
abc</TD></TR>
</TABLE></TD><TD BORDER="0"><TABLE CELLSPACING="0"><TR><TD BGCOLOR="white" BOR\
DER="0">a</TD></TR>
</TABLE></TD><TD BORDER="0"><TABLE CELLSPACING="0"><TR><TD BGCOLOR="white" BOR\
DER="0">ab</TD></TR>
</TABLE></TD></TR>
<TR><TD BORDER="0"><TABLE CELLSPACING="0"><TR><TD BGCOLOR="white" BORDER="0">b\
c</TD></TR>
</TABLE></TD><TD BORDER="0"><TABLE CELLSPACING="0"><TR><TD BGCOLOR="white" BOR\
DER="0">*bca</TD></TR>
</TABLE></TD><TD BORDER="0"><TABLE CELLSPACING="0"><TR><TD BGCOLOR="white" BOR\
DER="0">b</TD></TR>
</TABLE></TD></TR>
<TR><TD BORDER="0"><TABLE CELLSPACING="0"><TR><TD BGCOLOR="white" BORDER="0">c\
</TD></TR>
</TABLE></TD><TD BORDER="0"><TABLE CELLSPACING="0"><TR><TD BGCOLOR="white" BOR\
DER="0">ca</TD></TR>
</TABLE></TD><TD BORDER="0"><TABLE CELLSPACING="0"><TR><TD BGCOLOR="white" BOR\
DER="0">*cab</TD></TR>
</TABLE></TD></TR>
</TABLE>>];
}
```

## 3.2 Drawing the D-classes of a semigroup

### 3.2.1 DrawDClasses

▷ `DrawDClasses(arg)` (function)

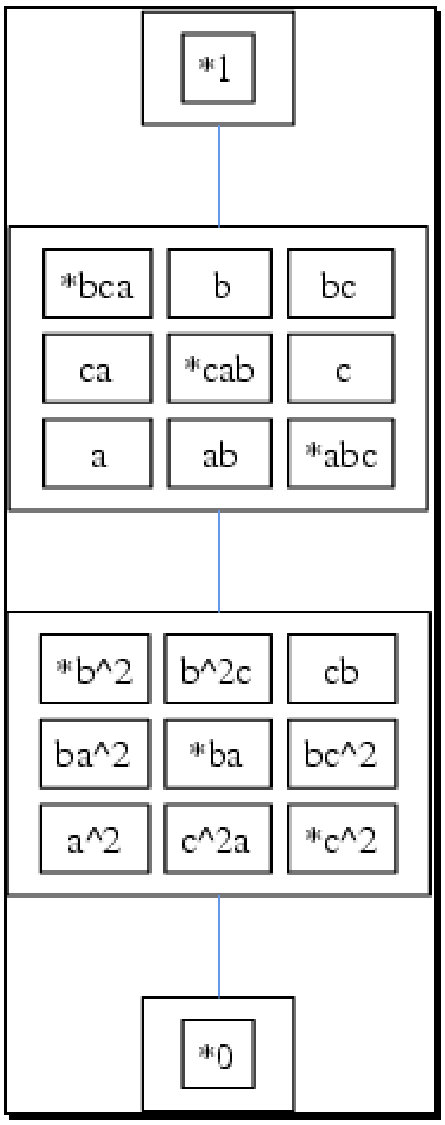
This function is similar to the previous one, except that this one draws all the D-classes of the semigroup given as the first argument. It then takes optionally n lists of elements and the elements of each list will be drawn in different colours. It also accepts, as an optional argument, the integer `1`, to specify whether the elements will appear as words or as transformations as in the previous function. The idempotents will be marked with a * before them.

The dot code is computed by `DotForDrawingDClasses` (3.2.2).

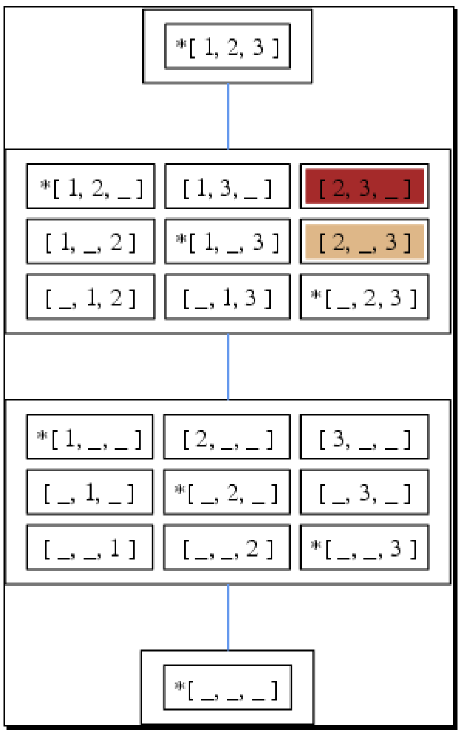
This last optional argument may also be the integer `2` and in this case the elements will appear as integers, where `i` represents the element `Elements(S)[i]`.

Example
```
gap> DrawDClasses(poi3);
gap> DrawDClasses(poi3, [Transformation( [ 2, 3, 4, 4 ] ),
  Transformation( [ 2, 4, 3, 4 ] )],
  [Transformation( [ 2, 4, 3, 4 ] )],1);
```

This is the image produced by the first command in the previous example:



This is the image produced by the second command in the previous example:



### 3.2.2 DotForDrawingDClasses

▷ `DotForDrawingDClasses(arg)` (function)

This function computes the dot code that can be used to produce a drawing for the D-class of an element of a semigroup. This can be done by using the function `DrawDClasses` (3.2.1) (if the system is properly configured) or by the user in some independent way. The arguments and options are the same than those of `DrawDClasses` (3.2.1).

Example
```
gap> Print(DotForDrawingDClasses(poi3));
digraph  DClasses {
graph [center=yes,ordering=out];
node [shape=plaintext];
edge [color=cornflowerblue,arrowhead=none];
## ... many more lines ...
</TABLE></TD></TR>
</TABLE>>];
4:4 -> 3:3;
3:3 -> 2:2;
2:2 -> 1:1;
}
```

## 3.3 Cayley graphs

### 3.3.1 DrawRightCayleyGraph

▷ `DrawRightCayleyGraph(S)` (function)

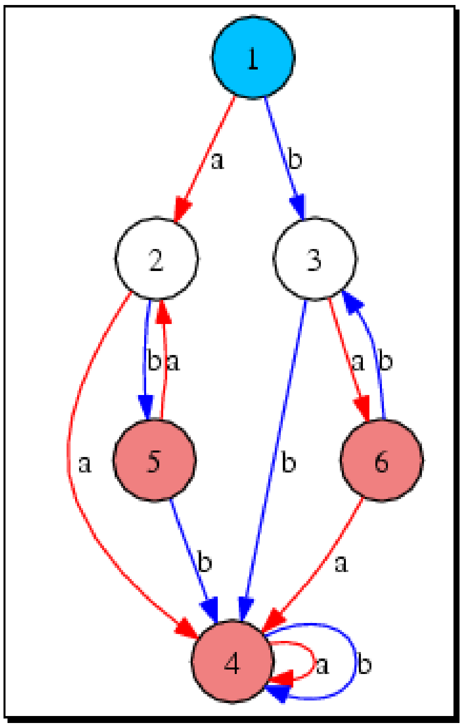
Draws the right Cayley graph of a finite monoid or semigroup `S`.

### 3.3.2 DrawCayleyGraph

▷ `DrawCayleyGraph(S)` (function)

This function is a synonym of `DrawRightCayleyGraph` (3.3.1).

For example, the command `DrawCayleyGraph(b21);` would produce the following image (where state `i` represents the element `Elements(S)[i]`, the neutral element is coloured in "light blue" and all other idempotents are coloured in "light coral"):



### 3.3.3 DotForDrawingRightCayleyGraph

▷ `DotForDrawingRightCayleyGraph(S)` (function)

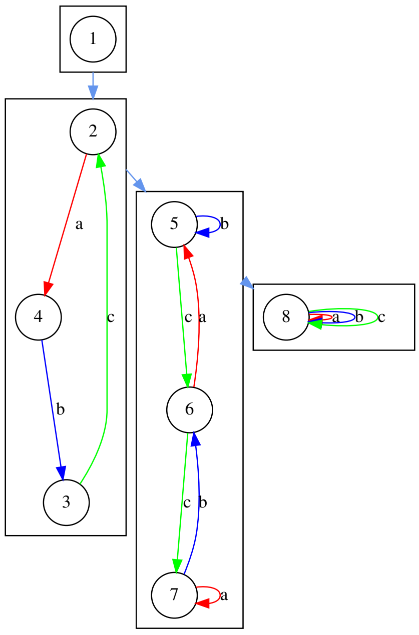
This function computes the dot code that is used by the previous function and can also be used by the reader in some independent way.

## 3.4 Schützenberger graphs

### 3.4.1 DrawSchutzenbergerGraphs

▷ `DrawSchutzenbergerGraphs(S)` (function)

Draws the Schützenberger graphs of the inverse semigroup *S*.
For example, `DrawSchutzenbergerGraphs(poi3);` would produce the following:



### 3.4.2 DotForDrawingSchutzenbergerGraphs

▷ `DotForDrawingSchutzenbergerGraphs(arg)` (function)

This function computes the dot code that can be used to produce a drawing for the Schutzenberger Graphs of an inverse semigroup. This can be done by using the function `DrawSchutzenbergerGraphs` (3.4.1) (if the system is properly configured) or by the user in some independent way. The argument is an inverse semigroup.

Example
```
gap> DotForDrawingSchutzenbergerGraphs(poi3);
"digraph  SchutzenbergerGraphs{\ncompound=true;\nsubgraph cluster4{\n1 [shape=\
circle];\n}\nsubgraph cluster3{\n2 -> 4 [label=\"a\",color=red];\n3 -> 2 [labe\
l=\"c\",color=green];\n4 -> 3 [label=\"b\",color=blue];\n2 [shape=circle];\n3 \
[shape=circle];\n4 [shape=circle];\n}\nsubgraph cluster2{\n5 -> 5 [label=\"b\"\
,color=blue];\n5 -> 6 [label=\"c\",color=green];\n6 -> 5 [label=\"a\",color=re\
d];\n6 -> 7 [label=\"c\",color=green];\n7 -> 7 [label=\"a\",color=red];\n7 -> \
6 [label=\"b\",color=blue];\n5 [shape=circle];\n6 [shape=circle];\n7 [shape=ci\
rcle];\n}\nsubgraph cluster1{\n8 -> 8 [label=\"a\",color=red];\n8 -> 8 [label=\
\"b\",color=blue];\n8 -> 8 [label=\"c\",color=green];\n8 [shape=circle];\n}\n1\
 -> 2 [lhead=cluster3,ltail=cluster4,color=cornflowerblue];\n2 -> 5 [lhead=clu\
ster2,ltail=cluster3,color=cornflowerblue];\n5 -> 8 [lhead=cluster1,ltail=clus\
ter2,color=cornflowerblue];\n}\n"
```

By using Print (or PrinTo, if one wants to print to a file) the string is displayed as follows:

Example

```
gap> Print(last);
digraph  SchutzenbergerGraphs{
compound=true;
subgraph cluster4{
1 [shape=circle];
}
subgraph cluster3{
2 -> 4 [label="a",color=red];
3 -> 2 [label="c",color=green];
4 -> 3 [label="b",color=blue];
2 [shape=circle];
3 [shape=circle];
4 [shape=circle];
}
subgraph cluster2{
5 -> 5 [label="b",color=blue];
5 -> 6 [label="c",color=green];
6 -> 5 [label="a",color=red];
6 -> 7 [label="c",color=green];
7 -> 7 [label="a",color=red];
7 -> 6 [label="b",color=blue];
5 [shape=circle];
6 [shape=circle];
7 [shape=circle];
}
subgraph cluster1{
8 -> 8 [label="a",color=red];
8 -> 8 [label="b",color=blue];
8 -> 8 [label="c",color=green];
8 [shape=circle];
}
1 -> 2 [lhead=cluster3,ltail=cluster4,color=cornflowerblue];
2 -> 5 [lhead=cluster2,ltail=cluster3,color=cornflowerblue];
5 -> 8 [lhead=cluster1,ltail=cluster2,color=cornflowerblue];
}
```

# Chapter 4

# User friendly ways to give semigroups and automata

This chapter describes two Tcl/Tk graphical interfaces that can be used to define and edit semigroups and automata.

## 4.1 Finite automata

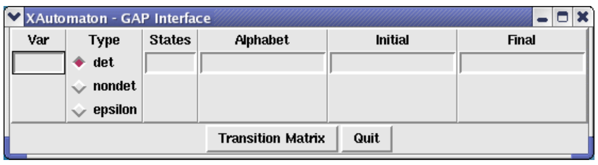
### 4.1.1 XAutomaton

▷ `XAutomaton( [A])` (function)

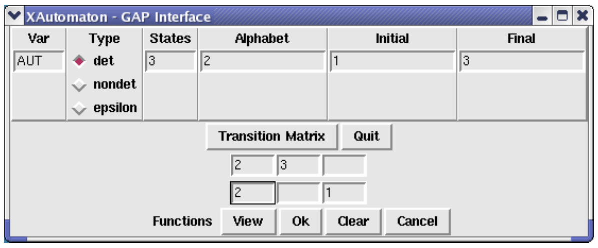
The function `Xautomaton` without arguments opens a new window where an automaton may be specified. A finite automaton (which may then be edited) may be given as argument.

```
Example
gap> XAutomaton();
```

It opens a window like the following:



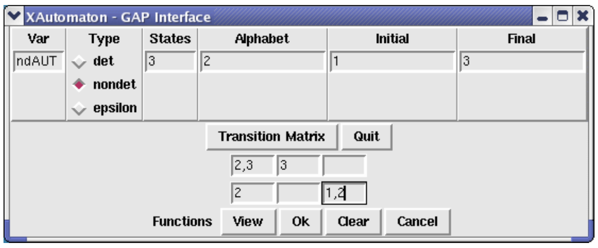
`Var` is the GAP name of the automaton, `States` is the number of states, `Alphabet` represents the alphabet and may be given through a positive integer (in this case the alphabet is understood to be `a,b,c,...` ) or through a string whose symbols, in order, being the letters of the alphabet. The numbers corresponding to the initial and accepting states are placed in the respective boxes. The automaton may be specified to be deterministic, non deterministic or with epsilon transitions. After pressing the TRANSITION MATRIX button the window gets larger and the transition matrix of the automaton may be given. The *i*th row of the matrix describes the action of the *i*th letter on the states.



| Var | Type | States | Alphabet | Initial | Final |
|---|---|---|---|---|---|
| AUT | [✓ det] nondet epsilon | 3 | 2 | 1 | 3 |

| | | |
|---|---|---|
| 2 | 3 | |
| 2 | | 1 |

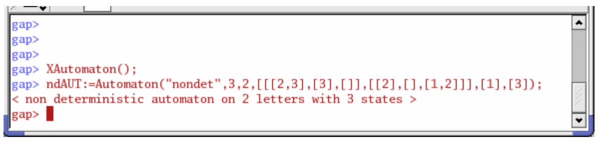
A non deterministic automaton may be given as follows:



| Var | Type | States | Alphabet | Initial | Final |
|---|---|---|---|---|---|
| ndAUT | det [✓ nondet] epsilon | 3 | 2 | 1 | 3 |

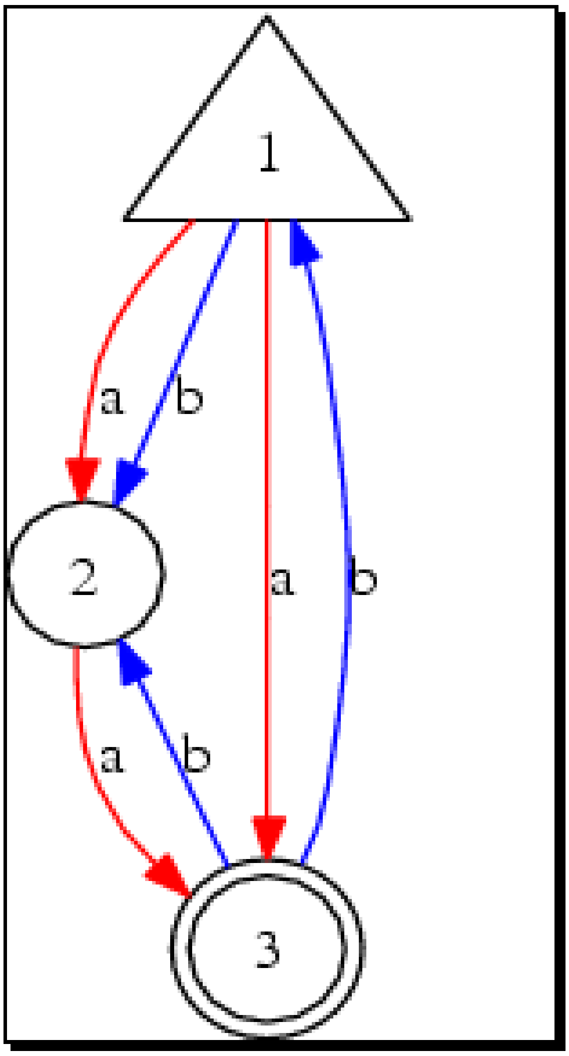
| | | |
|---|---|---|
| 2,3 | 3 | |
| 2 | | 1,2 |

By pressing the button OK the GAP shell acquires the aspect shown in the following picture and the automaton `ndAUT` may be used to do computations. Some computations such as getting a rational expression representing the language of the automaton, the (complete) minimal automaton representing the same language or the transition semigroup of the automaton, may be done directly after pressing the FUNCTIONS button.

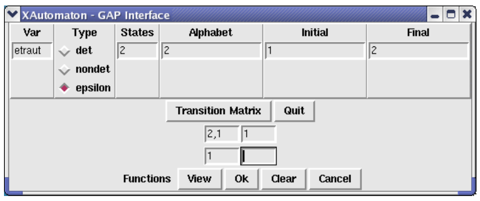


```
gap>
gap>
gap>
gap> XAutomaton();
gap> ndAUT:=Automaton("nondet",3,2,[[[2,3],[3],[]],[[2],[],[1,2]]],[1],[3]);
< non deterministic automaton on 2 letters with 3 states >
gap>
```

By pressing the button VIEW an image representing the automaton is displayed in a new window.



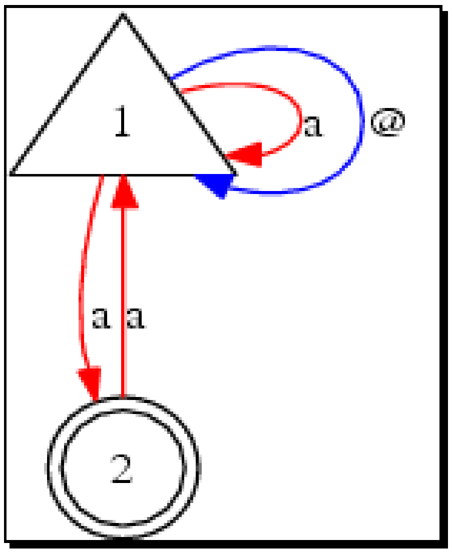
An automaton with epsilon transitions may be given as follows shown in the following picture. The last letter of the alphabet is always considered to be the $\varepsilon$. In the images it is represented by @.



| Var | Type | States | Alphabet | Initial | Final |
|---|---|---|---|---|---|
| etraut | det | 2 | 2 | 1 | 2 |
| | nondet | | | | |
| | [✓ epsilon] | | | | |

| | |
|---|---|
| 2,1 | 1 |
| 1 | |

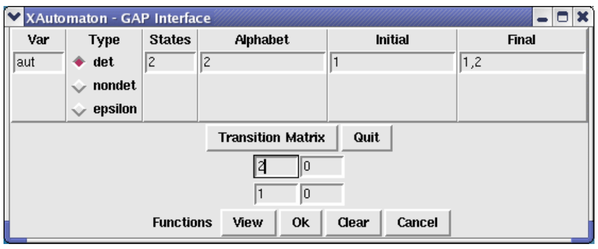
A new window with an image representing the automaton may be obtained by pressing the button VIEW .



In the next example it is given an argument to the function `XAutomaton`.

Example
```
gap> A := RandomAutomaton("det",2,2);
< deterministic automaton on 2 letters with 2 states >
gap> XAutomaton(A);
```

It opens a window like the following:



| Var | Type | States | Alphabet | Initial | Final |
|---|---|---|---|---|---|
| aut | [✓ det] nondet epsilon | 2 | 2 | 1 | 1,2 |

Transition Matrix

| | |
|---|---|
| 2 | 0 |
| 1 | 0 |

## 4.2 Finite semigroups

The most common ways to give a semigroup to are through generators and relations, a set of (partial) transformations as generating set and as syntatic semigroups of automata or rational languages.

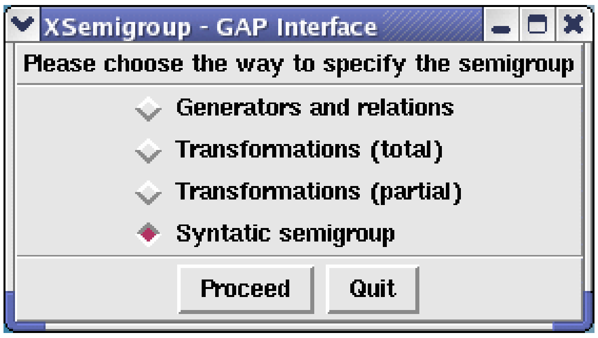
### 4.2.1 XSemigroup

▷ `XSemigroup(` *[S]* `)` (function)

The function `XSemigroup` without arguments opens a new window where a semigroup (or monoid) may be specified. A finite semigroup (which may then be edited) may be given as argument.

Example
```
gap> XSemigroup();
```

It opens a window like the following:


XSemigroup - GAP Interface

Please choose the way to specify the semigroup

Generators and relations

Transformations (total)

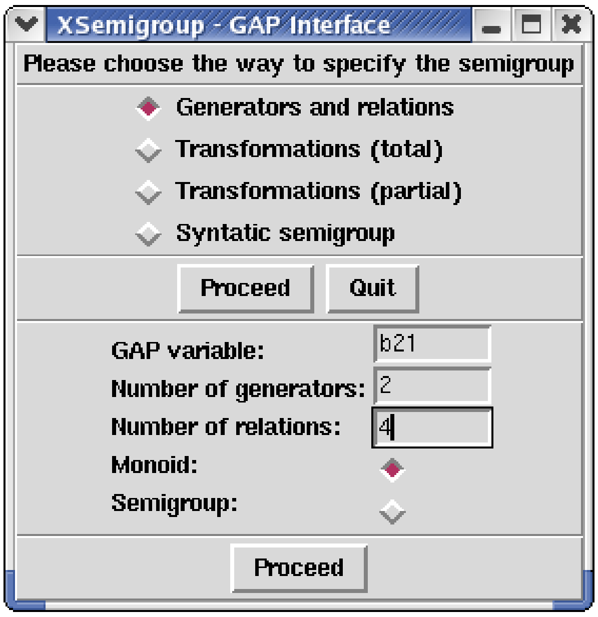
Transformations (partial)

[✓ Syntatic semigroup]

where one may choose how to give the semigroup.

### 4.2.2 Semigroups given through generators and relations

In the window opened by `XSemigroup`, by pressing the button PROCEED the window should enlarge and have the following aspect. (If the window does not enlarge automatically, use the mouse to do it.)


XSemigroup - GAP Interface

Please choose the way to specify the semigroup

[✓ Generators and relations]

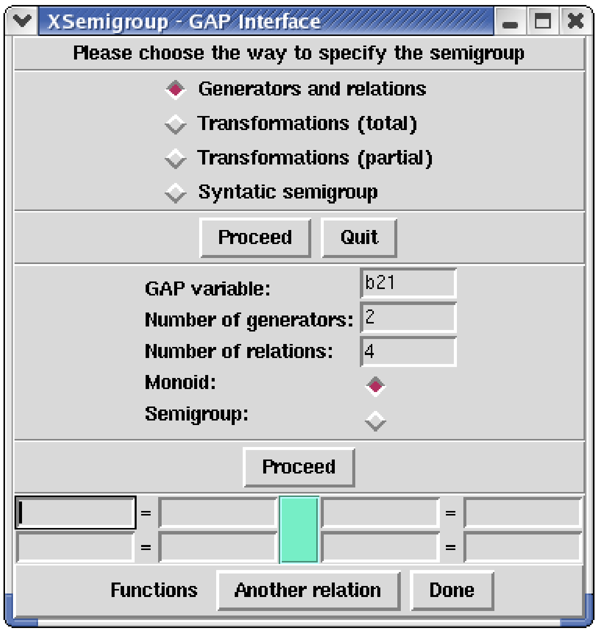
Transformations (total)

Transformations (partial)

Syntatic semigroup

| | |
|---|---|
| GAP variable: | b21 |
| Number of generators: | 2 |
| Number of relations: | 4 |
| Monoid: | ✓ |
| Semigroup: | |

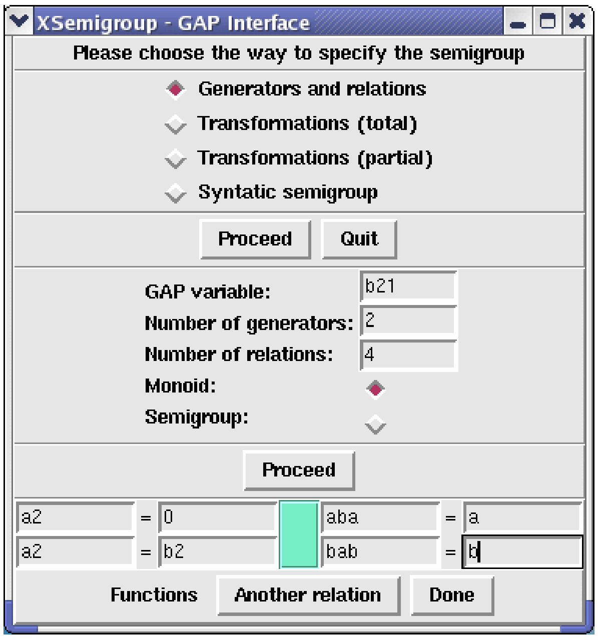
`GAP variable` is the GAP name of the semigroup. One has then to specify the number of generators, the number of relations (which does not to be exact) and whether one wants to produce a monoid or a semigroup. Pressing the PROCEED button one gets:



Please choose the way to specify the semigroup

[✓ Generators and relations]
Transformations (total)
Transformations (partial)
Syntactic semigroup

GAP variable: b21
Number of generators: 2
Number of relations: 4
[✓ Monoid:]
Semigroup:

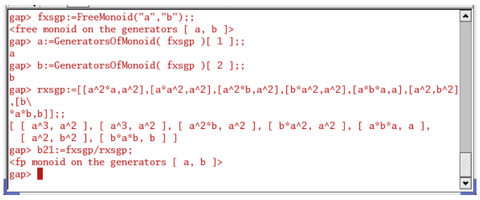


Please choose the way to specify the semigroup

[✓ Generators and relations]
Transformations (total)
Transformations (partial)
Syntactic semigroup

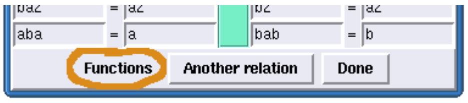
GAP variable: b21
Number of generators: 2
Number of relations: 4
[✓ Monoid:]
Semigroup:

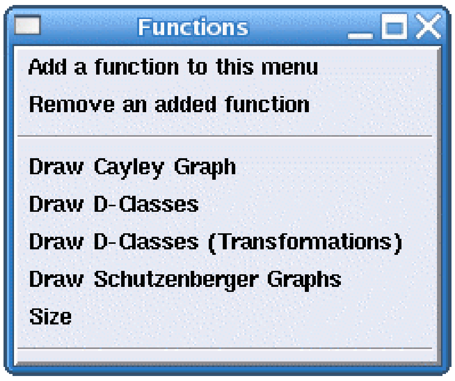
a2 = 0    aba = a
a2 = b2    bab = b



```
gap> fxsgp:=FreeMonoid("a","b");;
<free monoid on the generators [ a, b ]>
gap> a:=GeneratorsOfMonoid( fxsgp )[ 1 ];;
a
gap> b:=GeneratorsOfMonoid( fxsgp )[ 2 ];;
b
gap> rxsgp:=[[a^2*a,a^2],[a*a^2,a^2],[a^2*b,a^2],[b*a^2,a^2],[a*b*a,a],[a^2,b^2]
,[b\
*a*b,b]];;
[ [ a^3, a^2 ], [ a^3, a^2 ], [ a^2*b, a^2 ], [ b*a^2, a^2 ], [ a*b*a, a ],
  [ a^2, b^2 ], [ b*a*b, b ] ]
gap> b21:=fxsgp/rxsgp;
<fp monoid on the generators [ a, b ]>
gap> 
```

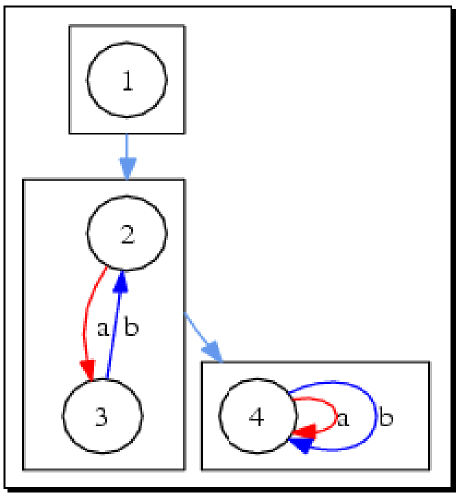
The menu button FUNCTIONS has the following commands:

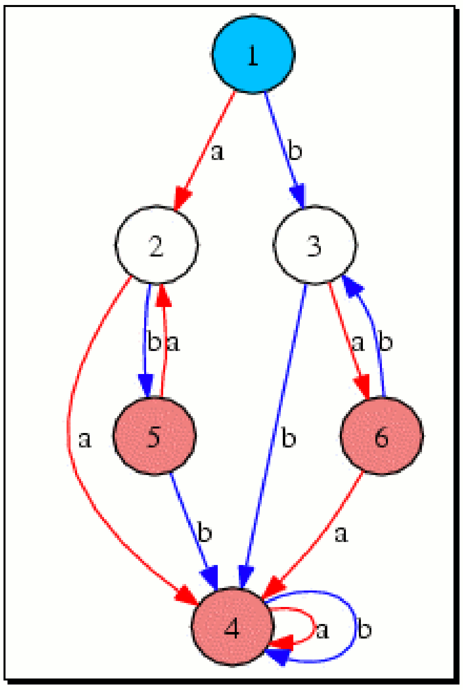


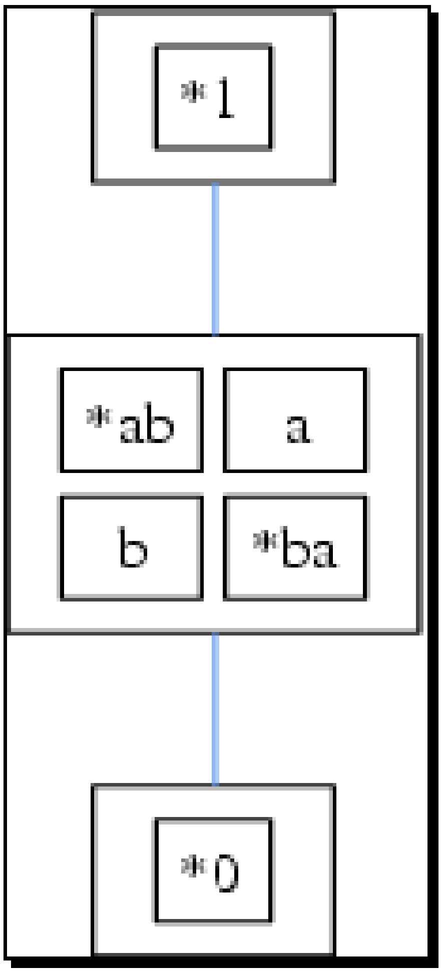

Functions
Add a function to this menu
Remove an added function
Draw Cayley Graph
Draw D-Classes
Draw D-Classes (Transformations)
Draw Schutzenberger Graphs
Size

The interface allows to add and remove GAP functions to the menu. When adding a function, the name of the function should be provided. (In its current version, it works only with functions that have as only argument a semigroup.)

By pressing the menu button FUNCTIONS and selecting "Draw Schutzenberger Graphs" would pop up the following window:

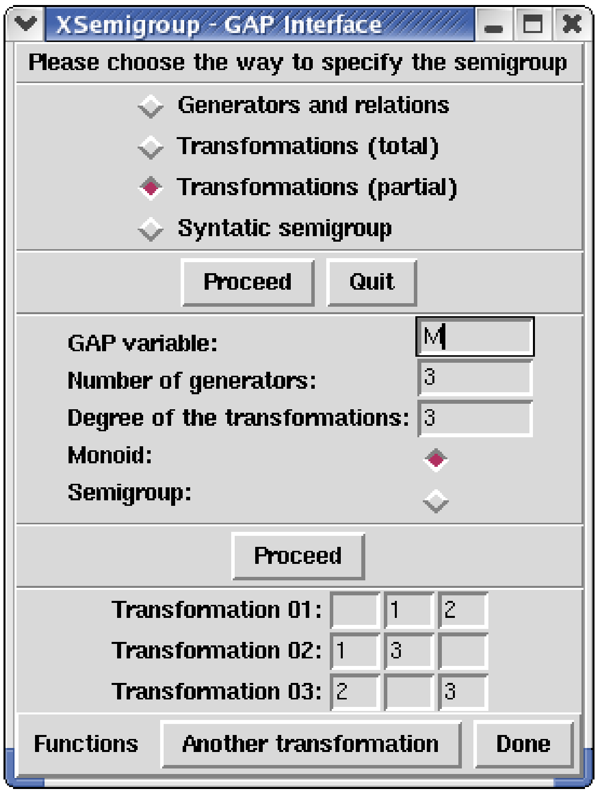



1
2
3
4
5
6
a
b
a
b
a
b
a
b
b
a
a
b



### 4.2.3 Semigroups given by partial transformations

`XSemigroup(poi3);` would pop up the following window, where everything should be clear:


XSemigroup - GAP Interface

Please choose the way to specify the semigroup

- Generators and relations
- Transformations (total)
- [✓ Transformations (partial)]
- Syntatic semigroup

Proceed Quit

| | |
|---|---|
| GAP variable: | M |
| Number of generators: | 3 |
| Degree of the transformations: | 3 |
| Monoid: | ✓ |
| Semigroup: | |

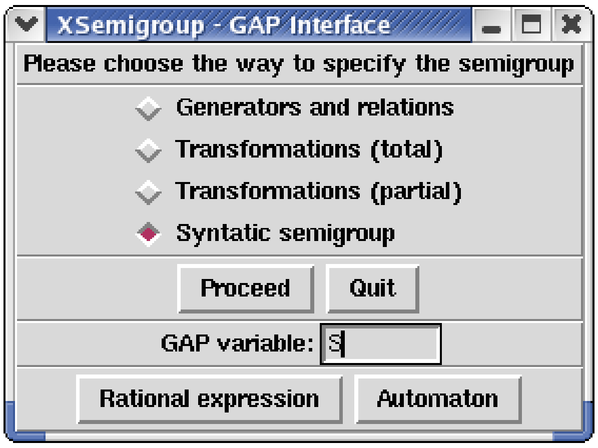
Proceed

| | | | |
|---|---|---|---|
| Transformation 01: | | 1 | 2 |
| Transformation 02: | 1 | 3 | |
| Transformation 03: | 2 | | 3 |

Functions Another transformation Done

### 4.2.4 Syntatic semigroups

`XSemigroup();` would pop up the following window, where we would select "Syntatic semigroup", press the PROCEED button and then choose either to give a "Rational expression" or an "Automaton" by pressing one of those buttons:


XSemigroup - GAP Interface

Please choose the way to specify the semigroup

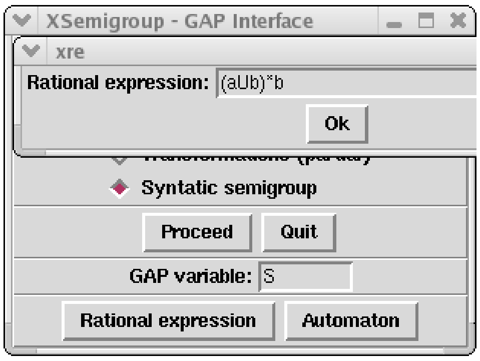
- Generators and relations
- Transformations (total)
- Transformations (partial)
- [✓ Syntatic semigroup]

Proceed Quit

GAP variable: S

Rational expression Automaton

If "Rational expression" is chosen, a new window pops up where the expression can be specified:

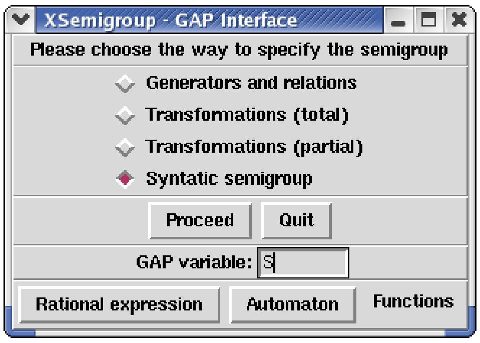


Rational expression: (aUb)*b

[✓ Syntactic semigroup]

GAP variable: S

After pressing the OK button, notice that the menu button FUNCTIONS appears on the main window (lower right corner) meaning that GAP already recognizes the given semigroup:



Please choose the way to specify the semigroup

Generators and relations
Transformations (total)
Transformations (partial)
[✓ Syntactic semigroup]

GAP variable: S

# References

[DEG$^+$02] D. Dobkin, J. Ellson, E. Gansner, E. Koutsofios, S. North, and G. Woodhull. Graphviz - graph drawing programs. Technical report, AT&T Research and Lucent Bell Labs, 2002. https://www.graphviz.org/. 4

# Index