Semigroup visualization

(Version 0.998)

Manuel Delgado
José João Morais

Manuel Delgado — Email: mdelgado@fc.up.pt
— Homepage: http://www.fc.up.pt/cmup/mdelgado

José João Morais — Email: josejoao@fc.up.pt

Copyright

Acknowledgements

The first author aknowledges 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 helpfull discussions and comments.

Colophon

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

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 `http://www.graphviz.org/`. It is used not only to draw right Cayley graphs of finite semigroups and Sch{\"u}zenberger graphs of finite inverse semigroups but also to visualize in the usual way the egg-box picture of a D-classe of a finite semigroup.

IMPORTANT NOTE: The version of `graphviz` to install should 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.

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 semigroup on the generators [<identity ... >, a, b ]>
```

Example
```
gap> f := FreeSemigroup("a","b");
<free semigroup on the generators [ a, b ]>
gap> a := GeneratorsOfSemigroup( f )[ 1 ];;
gap> b :=   GeneratorsOfSemigroup( 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> b2:= f/r;
<fp semigroup 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, ..., 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);
<semigroup with 4 generators>
gap> el := Elements(css3)[8];;
gap> D := GreensDClassOfElement(css3, el);;
gap> IsRegularDClass(D);
true
gap> DisplayEggBoxOfDClass(D);
[ [  1,  0,  1,  0 ],
  [  0,  1,  0,  1 ],
  [  0,  1,  0,  1 ],
  [  1,  0,  1,  0 ] ]
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 Schutzenberger 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 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, we may specify a string name the file that will be used to write the drawing of the class (in PostScript format) and 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);
```

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 a string specifying the name of the file in which the drawing will be written and the last, 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.

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,"DClasses");
gap> DrawDClasses(poi3, [Transformation( [ 2, 3, 4, 4 ] ),
  Transformation( [ 2, 4, 3, 4 ] )],
  [Transformation( [ 2, 4, 3, 4 ] )],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 colored in ”light blue” and all other idempotents are colored in ”light coral”):

3.4 Schutzenberger graphs

3.4.1 DrawSchutzenbergerGraphs

◊ `DrawSchutzenbergerGraphs(`*S*`)` (function)

Draws the Schutzenberger graphs of the inverse semigroup *S*.

For example, `DrawSchutzenbergerGraphs(poi3);` would produce the following:

3.5 Drawings output formats

By default, when a drawing is requested, it is outputted in PostScript format. Since `graphviz` allows other output formats (see http://www.graphviz.org/doc/info/output.html), it is possible to also request a drawing in a format other than PostScript.

3.5.1 DrawingsListOfExtraFormats

◊ `DrawingsListOfExtraFormats` (global variable)

This is a global variable which holds the valid output formats for a drawing. It currently has the value: `["dia", "fig", "gd", "gd2", "gif", "hpgl", "jpg", "mif", "mp", "pcl", "pic", "plain", "plain-ext", "png", "ps", "ps2", "svg", "svgz", "vrml", "vtx", "wbmp", "none"]` (see http://www.graphviz.org/doc/info/output.html for their description).

3.5.2 DrawingsExtraFormat

◊ `DrawingsExtraFormat` (global variable)

This is a global variable which holds the alternative output format for a drawing. By default its value is `"none"` which indicates that just PostScript will be used as the output format.

If its value becomes one of those in `DrawingsListOfExtraFormats` (3.5.1), then besides the PostScript file, it will also be created a file in the alternative format.

To change this variable's value, please use `SetDrawingsExtraFormat` (3.5.3).

3.5.3 SetDrawingsExtraFormat

◊ `SetDrawingsExtraFormat(`*f*`)` (function)

This function is used to set the value of `DrawingsExtraFormat` (3.5.2) to the format *f* which is one of `DrawingsListOfExtraFormats` (3.5.1).

Example

```
gap> DrawingsExtraFormat;
"none"
gap> SetDrawingsExtraFormat("jpg");
gap> DrawingsExtraFormat;
"jpg"

gap> DrawRightCayleyGraph(poi3);
Displaying file: /tmp/tmp.tpJqvI/cayleygraph.dot.ps
The extra output format file: /tmp/tmp.tpJqvI/cayleygraph.dot.jpg
has also been created.
```

3.6 Drawings extra graph attributes

The functions described in this subsection are intended to give the user a finer control over the final drawing. They allow to define the graph attributes described in http://graphviz.org/doc/info/attrs.html. (Note that only graph attributes are allowed to be defined, not edge nor node attributes are supported yet.)

3.6.1 DrawingsExtraGraphAttributes

◊ DrawingsExtraGraphAttributes (global variable)

This is a global variable which holds a list of strings, each of which defines a *dot* graph attribute. This variable is *not* intended to be modified by the user directly, but can be used to check which extra attributes are currently defined. To set the attributes, please use `SetDrawingsExtraGraphAttributes` (3.6.2). If `DrawingsExtraGraphAttributes` holds the value `"none"` then the default *dot* settings will be used. Use `ClearDrawingsExtraGraphAttributes` (3.6.3) to set it to `"none"`.

3.6.2 SetDrawingsExtraGraphAttributes

◊ SetDrawingsExtraGraphAttributes(*L*) (function)

This is the function to define the drawing's graph attributes (see http://graphviz.org/doc/info/attrs.html for a list and explanation of them). the argument *L* is a list of strings, each of which defines a *dot* graph attribute. For example, if we wanted to define the graph size to be 7x9 (in inches), we would call `SetDrawingsExtraGraphAttributes(["size=7,9"]);`.

If we also wanted to define the graph to be displayed in landscape mode we would call `SetDrawingsExtraGraphAttributes(["size=7,9", "rotate=90"]);`. If, in addition we wanted to define the background color to be, for example, pink, we would call `SetDrawingsExtraGraphAttributes(["size=7,9", "rotate=90", "bgcolor=pink"]);`.

After defining the attributes, any command that creates a drawing will use the last defined attributes. To set them back to the defaults, please use `ClearDrawingsExtraGraphAttributes` (3.6.3).

Example

```
gap> DrawingsExtraGraphAttributes;
"none"


gap> SetDrawingsExtraGraphAttributes(["size=7,9"]);

gap> DrawingsExtraGraphAttributes;
[ "size=7,9" ]


gap> SetDrawingsExtraGraphAttributes(["size=7,9", "rotate=90"]);

gap> DrawingsExtraGraphAttributes;
[ "size=7,9", "rotate=90" ]


gap> SetDrawingsExtraGraphAttributes(["size=7,9", "rotate=90", "bgcolor=pink"]);

gap> DrawingsExtraGraphAttributes;
```

```
[ "size=7,9", "rotate=90", "bgcolor=pink" ]


gap> ClearDrawingsExtraGraphAttributes();

gap> DrawingsExtraGraphAttributes;
"none"
```

3.6.3 ClearDrawingsExtraGraphAttributes

◊ `ClearDrawingsExtraGraphAttributes()` (function)

This function sets the graph drawing attributes back to *dot*'s defaults.

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:

Due to problems with scaling and displaying images, they will be available only in HTML format.

`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. A non deterministic automaton may be given as follows:

By pressing the button OK the GAP shell aquires 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.

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 ε. In the images it is represented by .

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:

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 syntactic 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: 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.)

`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:

4.2.3 Semigroups given by partial transformations

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

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

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. http://www.graphviz.org/. 5

Index