New features in support of semigroups

Andrew Solomon

April 1999

Credits for the material presented in this talk...

Implementation:

- Those responsible for writing and maintaining the basic machinery of GAP 4 – 90%
- Isabel Araújo, Robert Arthur, Götz Pfeiffer, Andrew Solomon – 10%

Bugs: Andrew Solomon – 100%

Groovy LaTeX macros and fonts: Alexander Hulpke – 100%

Things we can now do with semigroups

- Work with semigroups of various kinds:
  - generic semigroups and monoids;
  - semigroups of transformations;
  - finitely presented semigroups.
- Standard constructions on semigroups:
  - equivalence relations;
  - congruences and quotients;
  - left, right and two-sided ideals of semigroups;
  - Green's relations;
- Semigroups from groups:
  - 0-groups;
  - [0-]Simple semigroups and the Rees-Suškevič theorem;
  - Rees Matrix semigroups.

Making Semigroups

Given any list of elements in the category `IsAssociativeElement`, we can create the semigroup generated by them:

```
# The semigroup of natural numbers generated by 2 and 3

gap> s := Semigroup([2,3]);
gap> i := Iterator(s);

# Print the first 10 elements of the semigroup s

gap> for j  in [1 ..10] do
> Print(NextIterator(i)," ");
> od;

2 3 4 6 9 8 12 18 27 16
```

Transformations

A transformation is simply an endofunction of the set $\{1, \ldots, n\}$.

For example, below we create the monoid of all order preserving transformations of the chain of length 3:

```
gap> a := Transformation([1,1,3]);;
gap> b := Transformation([2,2,3]);;
gap> c := Transformation([1,2,2]);;
gap> d := Transformation([1,3,3]);;

gap> m := Monoid([a,b,c,d]);;
gap> Size(m);
10
```

Cayley's Theorem states that

Every semigroup is isomorphic to a semigroup of transformations.

Consequently, transformations are to computational semigroup theory what permutations are to computational group theory. In GAP 4 we are able to create transformations and consequently semigroups of transformations.

Cool Feature . . . The algorithms of Linton et. al. for fast computation of the local stucture of transformation semigroups are being incorporated into GAP 4. Our preliminary trials suggest speedups of $1-4$ orders of magnitude over the naive methods.

Finitely Presented Semigroups

We can create finitely presented semigroups, perform Knuth Bendix and Todd Coxeter.

First, create the semigroup with presentation

$$\langle x, y \mid x^5 = y, y^6 = x \rangle.$$

```
gap> fs := FreeSemigroup(["x","y"]);
<free semigroup on the generators [ x, y ]>
gap> x := GeneratorsOfSemigroup(fs)[1];;
gap> y := GeneratorsOfSemigroup(fs)[2];;
gap> g := fs/[[x^5,y],[y^6,x],[x*y,y*x]];
<fp semigroup on the generators [ x, y ]>
```

Cool Feature . . . Fast algorithms for commutative finitely presented semigroups with confluent rewriting system (AS and Araújo). Preliminary trials suggest a speedup of $1 - 3$ orders of magnitude.

Create a rewriting system, make it confluent and calculate the size.

```
gap> k := KnuthBendixRewritingSystem(g);
Knuth Bendix Rewriting System for Semigroup( [ x, y ] ) with rules
[ [ x^5, y ], [ y^6, x ], [ y*x, x*y ] ]
gap> IsConfluent(k);
false
gap> MakeConfluent(k);
gap> IsCommutative(g);
true
gap> Size(g);
29
```

Real Soon Now . . . New *fast* representation for elements of free commutative semigroups.

Todd Coxeter for Fp Semigroups

Let g be the finitely presented semigroup above. The Todd-Coxeter algorithm gives a representation of the semigroup as a semigroup of transformations.

```
gap> i := IsomorphismTransformationSemigroup(g);;
gap> tg := Range(i);
<semigroup with 2 generators>
gap> GeneratorsOfSemigroup(tg);
[ Transformation( [ 2, 3, 4, 5, 6, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
      21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 7, 8, 9, 10, 2 ] ),
  Transformation( [ 6, 11, 16, 21, 26, 7, 8, 9, 10, 2, 12, 13, 14, 15, 3, 17,
      18, 19, 20, 4, 22, 23, 24, 25, 5, 27, 28, 29, 30, 6 ] ) ]
gap> Size(tg);
29
```

Standard Constructions 1: Equivalence Relations

We can create equivalence relations in GAP 4 - they are a subcategory of `IsEndoGeneralMapping`.

```
gap> r := EquivalenceRelationByPartition([1,2,3],[[1,2],[3]]);
<general mapping: [ 1, 2, 3 ] -> [ 1, 2, 3 ] >
```

. . . and equivalence classes.

```
gap> c := EquivalenceClassOfElement(r, 2);
<Equivalence class of <general mapping: [ 1, 2, 3 ] ->
[ 1, 2, 3 ] > containing element 2>
gap> Elements(c);
[ 1, 2 ]
```

Standard Constructions 2: Congruences and Quotients

Let ρ be an equivalence relation on a semigroup S such that for all $(a, b) \in \rho$ and $x, y \in S$, $(xay, xby) \in \rho$. Then we say that ρ is a *congruence* on S and we can form a new semigroup S/ρ, the *quotient* of S by ρ, whose elements are ρ-equivalence classes of elements of S.

In the following example, a, b, c, d, e and f are the canonical generators of O_4.

```
gap> o4 := Monoid([a,b,c,d,e,f]);
<monoid with 6 generators>
gap> eO4 := Elements(M4);;
```

Now create the minimal congruence collapsing together two elements with image size 3.

```
gap> eO4[6];
Transformation( [ 1, 1, 2, 3 ] )
gap> eO4[9];
Transformation( [ 1, 1, 3, 4 ] )
gap> congO4:= MagmaCongruenceByGeneratingPairs(O4, [[eO4[6],eO4[9]]]);
<MagmaCongruence with 1 generating pairs>
```

Finally, create the quotient semigroup

```
gap> qO4 := o4/congO4;
<monoid with 7 generators>
gap> o4;
<monoid with 6 generators>
gap> Size(qO4);
2
```

The elements of the quotient semigroup are:

```
gap> elqO4 := Elements(qO4);
[ <Equivalence class of MagmaCongruence(
    [ [ Transformation( [ 1, 1, 2, 3 ] ), Transformation( [ 1, 1, 3, 4 ] ) ]
     ] ) containing element Transformation( [ 1, 2, 2, 4 ] )>,
  <Equivalence class of MagmaCongruence(
    [ [ Transformation( [ 1, 1, 2, 3 ] ), Transformation( [ 1, 1, 3, 4 ] ) ]
     ] ) containing element Transformation( [ 1 .. 4 ] )> ]
```

Standard Constructions 3: Left-, Right- and TwoSided Ideals

Let O_4 be as above. Then we can construct the two sided ideal i of all elements with image size 1 as follows:

```
# just pick any element with image size 1 as generator
gap> i := MagmaIdealByGenerators(o4,[eO4[1]]);
<MagmaIdeal with 1 generators>
gap> Elements(i);
[ Transformation( [ 1, 1, 1 ] ), Transformation( [ 2, 2, 2 ] ),
  Transformation( [ 3, 3, 3 ] ) ]
```

Note – the left ideal with the same generator is just a singleton:

```
gap> l := LeftMagmaIdealByGenerators(o4, [eO4[1]]);
<LeftMagmaIdeal with 1 generators>
gap> Elements(l);
[ Transformation( [ 1, 1, 1 ] ) ]
```

Standard Constructions 4: Green's Relations

Green defined five equivalence relations on semigroups which we denote $\mathcal{L}$, $\mathcal{R}$, $\mathcal{D}$, $\mathcal{J}$, and $\mathcal{H}$.

For example, two elements of a semigroup are $\mathcal{L}$-related if they generate the same principal left ideal. We have functions to define and enumerate Green's equivalence classes. In the case of transformation semigroups, these utilize the algorithms of Linton et. al. producing results many orders of magnitude faster than the generic algorithms.

Create Green's $\mathcal{R}$ relation:

```
gap> r := GreensRRelation(o4);
<general mapping: Semigroup( [ Transformation( [ 1, 1, 3 ] ),
  Transformation( [ 2, 2, 3 ] ), Transformation( [ 1, 2, 2 ] ),
  Transformation( [ 1, 3, 3 ] ) ] ) -> Semigroup(
[ Transformation( [ 1, 1, 3 ] ), Transformation( [ 2, 2, 3 ] ),
  Transformation( [ 1, 2, 2 ] ), Transformation( [ 1, 3, 3 ] ) ] ) >
```

Make the $\mathcal{R}$-class of the generator c ...

```
gap> rc := EquivalenceClassOfElement(r,c);
<Equivalence class of <general mapping: Semigroup(
[ Transformation( [ 1, 1, 3 ] ), Transformation( [ 2, 2, 3 ] ),
  Transformation( [ 1, 2, 2 ] ), Transformation( [ 1, 3, 3 ] )
 ] ) -> Semigroup( [ Transformation( [ 1, 1, 3 ] ),
  Transformation( [ 2, 2, 3 ] ), Transformation( [ 1, 2, 2 ] ),
  Transformation( [ 1, 3, 3 ] ) ] ) > containing element Transformation(
[ 1, 2, 2 ] )>
```

... and calculate the elements.

```
gap> Elements(rc);
[ Transformation( [ 1, 2, 2 ] ), Transformation( [ 1, 3, 3 ] ),
  Transformation( [ 2, 3, 3 ] ) ]
```

Semigroups from groups

One of the most important ways of constructing a semigroup while retaining some control over the structure is to build it from a group.

A zero group is a semigroup G^0 which has a distinguished element 0 with the property that for all $x \in G^0$, $x0 = 0x = 0$ and such that the set of nonzero elements forms a group.

We can create a zero group as follows:

```
gap> g := Group((1,2,3));;
gap> i := InjectionZeroMagma(g);;
gap> gz := Range(i);
<monoid with 2 generators>
gap> Elements(gz);
[ 0, (), (1,2,3), (1,3,2) ]
```

[0-]Simple semigroups and the Rees-Suškevič theorem

A simple semigroup is one whose only ideal is the whole semigoup. A 0-simple semigroup is a semigroup with zero whose only ideals are the whole semigoup and the singleton consisting of 0.

The Rees-Suškevič Theorem states that *finite semigroups without nontrivial (two sided) ideals are precisely Rees matrix semigroups over zero groups*

Question 1: What the hell's that?!?

Question 2: How can you make one in GAP 4?

Assume that `egz` is the list of elements of `gz` above:

```
# create the rees matrix (which is just a list)
gap> rmat := [[egz[1],egz[3]],[egz[2],egz[2]]];;

# now the semigroup ...
gap> rms := ReesMatrixSemigroup(gz,rmat);
Rees Matrix Semigroup over Monoid( [ (1,2,3), 0 ], ... )
```

```
# now make a couple of elements
gap> e1 := ReesMatrixSemigroupElement(rms,egz[2],1,1);
(();1,1)
gap> e2 := ReesMatrixSemigroupElement(rms,egz[1],1,1);
0
gap> IsMultiplicativeZero(e2,rms);
true
gap> e2*e1;
0
```

Real Soon Now: Methods for IsSimpleSemigroup and IsomorphismReesMatrixSemigroup.