RCWA

Residue Class-Wise Affine Groups

(Version 1.0.0)

April 26, 2005

Stefan Kohl

Stefan Kohl — Email: kohl@mathematik.uni-stuttgart.de
— Homepage: http://www.cip.mathematik.uni-stuttgart.de/~kohlsn
— Address: Institut für Geometrie und Topologie
Universität Stuttgart
70550 Stuttgart
Germany

Abstract

The RCWA Package provides methods for computations with the so-called *R*esidue *C*lass-*W*ise *A*ffine mappings (*rcwa* mappings for short) and the groups generated by bijective mappings of this type. The rcwa mappings are a type of mappings of certain euclidian rings R into themselves. A possible choice is $R = \mathbb{Z}$. The bijective rcwa mappings of R form a proper subgroup of $\mathrm{Sym}(R)$. In general, computing with arbitrary mappings from $\mathbb{Z}$ into $\mathbb{Z}$ is an algorithmically difficult task. The rcwa mappings provide a class of mappings which are accessible to computations. The investigation of rcwa mappings and groups generated by them is the central aim of this package.

Copyright

Contents

1 Preface **6**
1.1 Motivation . . . 6
1.2 Purpose of this package . . . 7
1.3 Scope of this package . . . 7
1.4 Acknowledgements . . . 7

2 Residue Class-Wise Affine Mappings **8**
2.1 Basic definitions . . . 8
2.2 Entering rcwa mappings . . . 9
2.2.1 RcwaMapping (R, m, coeffs) . . . 10
2.2.2 ClassShift (r, m) . . . 12
2.2.3 ClassReflection (r, m) . . . 12
2.2.4 ClassTransposition (r1, m1, r2, m2) . . . 13
2.2.5 PrimeSwitch (p) . . . 13
2.2.6 LaTeXObj (f) . . . 14
2.3 Basic functionality for rcwa mappings . . . 14
2.4 Factoring rcwa mappings . . . 16
2.4.1 FactorizationIntoGenerators (g) . . . 16
2.5 Determinant and sign . . . 17
2.5.1 Determinant (sigma) . . . 17
2.5.2 Sign (sigma) . . . 18
2.6 Attributes and properties derived from the coefficients . . . 18
2.6.1 Multiplier (f) . . . 18
2.6.2 Divisor (f) . . . 18
2.6.3 PrimeSet (f) . . . 19
2.6.4 IsIntegral (f) . . . 19
2.6.5 IsClassWiseOrderPreserving (f) . . . 19
2.7 Functionality related to the affine partial mappings . . . 20
2.7.1 LargestSourcesOfAffineMappings (f) . . . 20
2.7.2 Multpk (f, p, k) . . . 20
2.7.3 SetOnWhichMappingIsClassWiseOrderPreserving (f) . . . 21
2.8 Transition graphs and transition matrices . . . 21
2.8.1 TransitionGraph (f, m) . . . 21
2.8.2 OrbitsModulo (f, m) . . . 21
2.8.3 FactorizationOnConnectedComponents (f, m) . . . 22
2.8.4 TransitionMatrix (f, m) . . . 22

2.9 Trajectories . . . 23
2.9.1 Trajectory (f, n, val, cond) . . . 23
2.9.2 TrajectoryModulo (f, n, m, lng) . . . 24
2.9.3 CoefficientsOnTrajectory (f, n, val, cond, all) . . . 24
2.9.4 IncreasingOn (f) . . . 25
2.10 Special functions for non-bijective mappings and miscellanea . . . 25
2.10.1 LikelyContractionCentre (f, maxn, bound) . . . 25
2.10.2 GuessedDivergence (f) . . . 26
2.10.3 ImageDensity (f) . . . 26
2.11 The categories and families of rcwa mappings . . . 26
2.11.1 IsRcwaMapping (f) . . . 26
2.11.2 RcwaMappingsFamily (R) . . . 26

3 Residue Class-Wise Affine Groups **27**
3.1 Constructing residue class-wise affine groups . . . 27
3.1.1 RCWA (R) . . . 27
3.1.2 Random (RCWA(Integers)) . . . 28
3.1.3 IsomorphismRcwaGroup (G) . . . 30
3.2 Attributes and properties of rcwa groups . . . 30
3.2.1 Modulus (G) . . . 30
3.2.2 IsTame (G) . . . 31
3.2.3 PrimeSet (G) . . . 31
3.3 Membership testing, order computation, permutation- / matrix representations . . . 32
3.3.1 \in (g, G) . . . 32
3.3.2 Size (G) . . . 32
3.3.3 IsomorphismPermGroup (G) . . . 32
3.3.4 IsomorphismMatrixGroup (G) . . . 33
3.4 Factoring elements into generators . . . 34
3.4.1 PreImagesRepresentative (phi, g) . . . 34
3.4.2 PreImagesRepresentatives (phi, g) . . . 35
3.5 The action of an rcwa group on the underlying ring R . . . 36
3.5.1 IsTransitive (G, Integers) . . . 36
3.5.2 RepresentativeAction (G, src, dest, act) . . . 37
3.5.3 RepresentativeActionPreImage (G, src, dest, act, F) . . . 38
3.5.4 RepresentativeAction (RCWA(Integers), P1, P2) . . . 38
3.5.5 ShortOrbits (G, S, maxlng) . . . 39
3.5.6 OrbitsModulo (G, m) . . . 40
3.6 Conjugacy in RCWA(R) . . . 40
3.6.1 IsConjugate (RCWA(Integers), f, g) . . . 40
3.6.2 RepresentativeAction (RCWA(Integers), f, g) . . . 41
3.6.3 ShortCycles (f, maxlng) . . . 41
3.6.4 NrConjugacyClassesOfRCWAZOfOrder (ord) . . . 42
3.7 Restriction monomorphisms . . . 42
3.7.1 Restriction (g, f) . . . 42
3.7.2 Restriction (G, f) . . . 43
3.7.3 DirectProduct (G1, G2, ...) . . . 43
3.8 Special attributes for tame rcwa groups . . . 44

3.8.1 RespectedPartition (G) . . . 44
3.8.2 ActionOnRespectedPartition (G) . . . 44
3.8.3 IntegralConjugate (G) . . . 45
3.8.4 IntegralizingConjugator (G) . . . 45
3.9 The categories and families of rcwa groups . . . 46
3.9.1 IsRcwaGroup (G) . . . 46
3.9.2 IntegralRcwaGroupsFamily . . . 46

4 Examples **47**
4.1 Factoring Collatz' permutation of the integers . . . 47
4.2 An rcwa mapping which seems to be contracting, but very slow . . . 54
4.3 Checking a result by P. Andaloro . . . 56
4.4 Two examples by Matthews and Leigh . . . 57
4.5 Exploring the structure of a wild rcwa group . . . 59
4.6 A wild rcwa mapping which has only finite cycles . . . 61
4.7 An abelian rcwa group over a polynomial ring . . . 65
4.8 An rcwa representation of a small group . . . 66
4.9 An rcwa representation of the symmetric group on 10 points . . . 67
4.10 Checking for solvability . . . 70
4.11 Some examples over (semi)localizations of the integers . . . 71
4.12 Twisting 257-cycles into an rcwa mapping with modulus 32 . . . 74
4.13 The behaviour of the moduli of powers . . . 75
4.14 Images and preimages under the Collatz mapping . . . 76

5 The Algorithms Implemented in RCWA **79**

6 Installation and auxiliary functions **85**
6.1 Installation . . . 85
6.2 The Info class of the package . . . 85
6.2.1 InfoRCWA . . . 85
6.3 Building the manual . . . 85
6.3.1 RCWABuildManual . . . 85
6.4 The testing routine . . . 86
6.4.1 RCWATest . . . 86

Chapter 1

Preface

1.1 Motivation

The development of this package has been motivated by the famous $3n+1$ - Conjecture, which states that iterated application of the Collatz mapping

$$T: \mathbb{Z} \longrightarrow \mathbb{Z}, \quad n \longmapsto \begin{cases} \frac{n}{2} & \text{if } n \text{ even,} \\ \frac{3n+1}{2} & \text{if } n \text{ odd} \end{cases}$$

to any given positive integer eventually yields 1.

This has been conjectured by Lothar Collatz in the 1930s, and is still an unsolved problem today. Replacing the Collatz mapping T by its conjugate under some permutation σ of $\mathbb{Z}$ fixing 1 and setwisely fixing the positive integers turns the $3n+1$ - Conjecture into the following equivalent conjecture: $\forall n \in \mathbb{N}\ \exists k \in \mathbb{N}_0 :\ n^{(T^\sigma)^k} = 1$. The $3n+1$ - Conjecture is true if and only if there is such a permutation σ such that T^σ maps all integers $n > 1$ to smaller ones. Hence the problem is to find a suitable normal form for the Collatz mapping T.

Investigating arbitrary permutations of finite sets is both theoretically and computationally a hard problem. It is straightforward to restrict the considerations to permutations which are "similar" to the Collatz mapping itself. Probably the most natural class of such mappings is the class of residue class-wise affine permutations.

Jeffrey C. Lagarias has written and maintains a commented bibliography [Lag04], which currently lists 193 references to publications related to the $3n+1$ - Conjecture. None of the articles which are referenced there tries to attack the problem by means of group theory, or investigates the structure of groups generated by bijective residue class-wise affine mappings. In fact, the subject *residue class-wise affine groups* apparently has not been treated anywhere in the literature before.

The author feels that this is a gap which is worth to be filled, and in particular that the group of all residue class-wise affine bijections of the integers is an object of natural interest in its own.

1.2 Purpose of this package

The purpose of this package is to serve as a tool for computational investigations of residue class-wise affine mappings and -groups mainly over the ring of the integers.

Perhaps the only sensible reason for using this package is being fascinated by the mathematical beauty of the objects it helps to investigate.

This manual is pure software documentation, and as such it does not contain any theorems or proofs. In a few places, where this is absolutely necessary for understanding what some function is good for, corresponding mathematical assertions are made. Proofs for all of them as well as a detailed development of the theoretical background for the subject will be published in the author's forthcoming PhD thesis [Koh05].

1.3 Scope of this package

There are relatively elaborate methods for dealing with residue class-wise affine groups whose elements have a bounded number of different affine partial mappings. These groups have a relatively easy structure. Hence they are called *tame*. Understanding tame residue class-wise affine groups is a necessary prerequisite for being able to investigate those which are not tame.

The groups which are not tame are much more difficult to examine. Hence these groups are called *wild*. Nevertheless, some computations can be done also in them – there is e.g. an algorithm for factoring elements into generators, which usually works reasonably well if the resulting word is not "too long". Often it is also possible to get useful information about a wild group by considering its action on finite orbits – of course provided that such orbits exist.

It cannot be said in a few sentences what can be found out with this package about which mappings or groups. The best way to get an idea about this is to experiment yourself with the examples discussed in this manual and included in the file `pkg/rcwa/examples/examples.g`. Another useful source of examples is the `Random` (3.1.2) - function. For many – if not most – problems the package does not provide an out-of-the-box solution. At the beginning you will probably be bored by extremely long runtimes for seemingly trivial things. But with some experience you will learn to estimate in advance how long something will take and to see why raising some harmlessly-looking mapping to the 20th power would take terabytes of memory, while one can easily find out non-trivial things about some group which looks much more complicate. Quite often it is possible to find an answer for a given question by using an interactive trial-and-error approach.

The author for example has found with substancial help of this package a non-trivial normal subgroup of the group of all residue class-wise affine bijections of the integers.

1.4 Acknowledgements

I would like to thank Bettina Eick for her kind help in trying to make this package and in particular its documentation more useful and more interesting for a larger number of people. Furthermore I would like to thank the two anonymous referees for their constructive criticism and helpful suggestions.

If you use RCWA in some work leading to a publication, I ask you to cite it just as you would cite a journal article. I would be grateful for any bug reports, comments or suggestions and of course for reports of results found with the help of this package.

Stuttgart, April 26, 2005 Stefan Kohl

Chapter 2

Residue Class-Wise Affine Mappings

This chapter describes the functionality provided by this package for computing with residue class-wise affine mappings.

2.1 Basic definitions

Let R be an infinite euclidean domain which is not a field and all of whose proper residue class rings are finite. A mapping $f : R \rightarrow R$ is called *residue class-wise affine*, or in short an *rcwa* mapping, if there is an $m \in R \setminus \{0\}$ such that the restrictions of f to the residue classes $r(m) \in R/mR$ are affine. This means that for any residue class $r(m)$ there are coefficients $a_{r(m)}, b_{r(m)}, c_{r(m)} \in R$, such that the restriction of the mapping f to the set $r(m) = \{r + km | k \in R\}$ is given by

$$f|_{r(m)} : \ r(m) \rightarrow R, \quad n \ \mapsto \ \frac{a_{r(m)} \cdot n + b_{r(m)}}{c_{r(m)}} \ .$$

The value m is called the *modulus* of f. It is understood that all fractions are reduced, i.e. that $\gcd(a_{r(m)}, b_{r(m)}, c_{r(m)}) = 1$, and that m is chosen multiplicatively minimal.

Apart from the restrictions imposed by the condition that the image of any residue class $r(m)$ under f must be a subset of R and that one cannot divide by 0, the coefficients $a_{r(m)}$, $b_{r(m)}$ and $c_{r(m)}$ can be any ring elements.

When talking about the *product* of some rcwa mappings, it is always meant their composition as mappings, and by the inverse of a bijective rcwa mapping it is meant its inverse mapping. Products are evaluated from the left to the right.

The set $\mathrm{RCWA}(R) \ := \ \{ \ g \in \mathrm{Sym}(R) \ | \ g \text{ is residue class-wise affine} \ \}$ is closed under multiplication and taking inverses (this can be verified easily), hence forms a subgroup of $\mathrm{Sym}(R)$. A subgroup of $\mathrm{RCWA}(R)$ is called a *residue class-wise affine* group, or in short an *rcwa* group.

2.2 Entering rcwa mappings

In order to define an rcwa mapping, it is necessary to specify the underlying ring R, the modulus m and the coefficients $a_{r(m)}$, $b_{r(m)}$ and $c_{r(m)}$ for $r(m)$ running over the residue classes (mod m).

A coefficient list for an rcwa mapping with modulus m consists of $|R/mR|$ coefficient triples `[ a_rm, b_rm, c_rm ]`. Their ordering is determined by the ordering of the representatives of the residue classes (mod m) in the sorted list returned by `AllResidues(R, m)`. For $R = \mathbb{Z}$, this means that the coefficient triple for the residue class $0(m)$ comes first, then comes the one for $1(m)$, the one for $2(m)$ and so on.

The easiest way to enter a given rcwa mapping is by `RcwaMapping( [ R, m, ] coeffs )`. If the arguments `R` and `m` are omitted, they default to `Integers` resp. the length of the coefficient list `coeffs`. If the given coefficients would enforce a division by zero or images of elements of `R` outside `R`, an error message is printed and a break loop is entered. For example, the coefficient triple `[1,1,3]` at the first position is not allowed if `R` is the ring of integers. The reason for this is that not all integers congruent to 0 + 1 = 1 mod `m` are divisible by 3.

Example

```
gap> T := RcwaMapping([[1,0,2],[3,1,2]]); # The Collatz mapping.
<rcwa mapping of Z with modulus 2>
gap> SetName(T,"T"); Display(T);

Rcwa mapping of Z with modulus 2

                n mod 2                |                n^T
---------------------------------------+---------------------------------------
  0                                    | n/2
  1                                    | (3n + 1)/2
```

In the sequel, a description of the general-purpose constructor for rcwa mappings is given. This might look a bit technical on a first glance. For getting started, the reader may find it easier to look first at the functions for constructing some kinds of particularly simple-structured bijective rcwa mappings of $\mathbb{Z}$ described afterwards.

2.2.1 RcwaMapping (R, m, coeffs)

◊ `RcwaMapping( R, m, coeffs )` (method)
◊ `RcwaMapping( R, coeffs )` (method)
◊ `RcwaMapping( coeffs )` (method)
◊ `RcwaMapping( perm, range )` (method)
◊ `RcwaMapping( m, values )` (method)
◊ `RcwaMapping( pi, coeffs )` (method)
◊ `RcwaMapping( q, m, coeffs )` (method)
◊ `RcwaMapping( P1, P2 )` (method)
◊ `RcwaMapping( cycles )` (method)

Returns: An rcwa mapping.

Construction of an rcwa mapping.

In all cases the argument `R` is the underlying ring, `m` is the modulus and `coeffs` is the coefficient list as described above. In case one or several of these arguments are omitted or replaced by other arguments, they are either derived from the latter or default values are taken. The meaning of the other arguments is defined in the detailed description of the particular methods given in the sequel. The above methods return the rcwa mapping

(a) of `R` with modulus `modulus` and coefficients `coeffs`, resp.

(b) of `R` = $\mathbb{Z}$ or `R` = $\mathbb{Z}_{(\pi)}$ with modulus `Length(coeffs)` and coefficients `coeffs`, resp.

(c) of `R` = $\mathbb{Z}$ with modulus `Length(coeffs)` and coefficients `coeffs`, resp.

(d) of `R` = $\mathbb{Z}$, acting on any set `range + k*Length(range)` like the permutation `perm` on the range `range`, resp.

(e) of `R` = $\mathbb{Z}$ with modulus `modulus` and values prescribed by the list `val`, which consists of 2·`modulus` pairs giving preimage and image for 2 points per residue class (mod `modulus`), resp.

(f) of $\mathbb{Z}_{(\pi)}$ with modulus `Length(coeffs)` and coefficients `coeffs` (the set of primes π denoting the underlying ring is given as argument `pi`), resp.

(g) of GF(`q`)[`x`] with modulus `modulus` and coefficients `coeffs`, resp.

(h) an arbitrary rcwa mapping which induces a bijection between the partitions `P1` and `P2` of `R` into disjoint single residue classes and which is affine on the elements of `P1`, resp.

(i) an arbitrary rcwa mapping with “residue class cycles” as given by `cycles`. The latter is a list of lists of disjoint residue classes which the mapping should permute cyclically, each.

The methods for the operation `RcwaMapping` perform a number of argument checks, which can be skipped by using `RcwaMappingNC` instead.

Example

```
gap> f := RcwaMapping([[1,1,1],[1,-1,1],[1,1,1],[1,-1,1]]);
<rcwa mapping of Z with modulus 2>
gap> f = RcwaMapping((2,3),[2..3]);
true
gap> g := RcwaMapping((1,2,3)(8,9),[4..20]);
<rcwa mapping of Z with modulus 17>
gap> Action(Group(g),[4..20]);
Group([ (5,6) ])
gap> T = RcwaMapping(2,[[1,2],[2,1],[3,5],[4,2]]);
true
gap> t := RcwaMapping(1,[[-1,1],[1,-1]]); # The involution n -> -n.
Rcwa mapping of Z: n -> -n
gap> d := RcwaMapping([2],[[1/3,0,1]]);
Rcwa mapping of Z_( 2 ): n -> 1/3 n
gap> RcwaMapping([2,3],ShallowCopy(Coefficients(T)));
<rcwa mapping of Z_( 2, 3 ) with modulus 2>
gap> u := RcwaMapping([[3,0,5],[9,1,5],[3,-1,5],[9,-2,5],[9,4,5]]);
<rcwa mapping of Z with modulus 5>
gap> x := Indeterminate(GF(2),1);; SetName(x,"x");
gap> R := PolynomialRing(GF(2),1); z := Zero(R);; e := One(R);;
GF(2)[x]
gap> r := RcwaMapping( R, x^2 + e,
>                       [ [ x^2 + x + e, z      , x^2 + e ],
>                         [ x^2 + x + e, x      , x^2 + e ],
>                         [ x^2 + x + e, x^2    , x^2 + e ],
>                         [ x^2 + x + e, x^2 + x, x^2 + e ] ] );
<rcwa mapping of GF(2)[x] with modulus x^2+Z(2)^0>
gap> rc := function(r,m) return ResidueClass(DefaultRing(m),m,r); end;;
gap> f1 := RcwaMapping([[rc(1,6),rc(0, 8)],[rc(5,6),rc(4, 8)]]);;
gap> f2 := RcwaMapping([[rc(1,6),rc(0, 4)],[rc(5,6),rc(2, 4)]]);;
gap> f3 := RcwaMapping([[rc(2,6),rc(1,12)],[rc(4,6),rc(7,12)]]);;
gap> List([f1,f2,f3],Order);
[ 2, 2, 2 ]
gap> f := f1*f2*f3;
<bijective rcwa mapping of Z with modulus 12>
gap> Order(f);
infinity
gap> a := RcwaMapping([rc(0,2),rc(1,4),rc(3,4)],[rc(0,3),rc(1,3),rc(2,3)]);
<rcwa mapping of Z with modulus 4>
gap> [rc(0,2),rc(1,4),rc(3,4)]^a;
[ 0(3), 1(3), 2(3) ]
gap> Cycle(a,44);
[ 44, 66, 99, 74, 111, 83, 62, 93, 70, 105, 79, 59 ]
```

Some kinds of very simple-structured rcwa mappings are of particular interest:

2.2.2 ClassShift (r, m)

◊ `ClassShift( r, m )` (function)

Returns: The *class shift* $\nu_{r(m)}$.

The *class shift* $\nu_{r(m)}$ is the rcwa mapping of $\mathbb{Z}$ which maps $n \in r(m)$ to $n+m$ and fixes $\mathbb{Z} \setminus r(m)$ pointwise. The residue class `ResidueClass(r,m)` itself can be given in place of the arguments `r` and `m`. Enclosing the argument list in list brackets is permitted.

Example

```
gap> Display(ClassShift(5,12));

Tame bijective rcwa mapping of Z with modulus 12, of order infinity

               n mod 12                |          n^ClassShift(5,12)
---------------------------------------+--------------------------------------
   0  1  2  3  4  6  7  8  9 10 11     | n
   5                                   | n + 12
```

2.2.3 ClassReflection (r, m)

◊ `ClassReflection( r, m )` (function)

Returns: The *class reflection* $\varsigma_{r(m)}$.

The *class reflection* $\varsigma_{r(m)}$ is the rcwa mapping of $\mathbb{Z}$ which maps $n \in r(m)$ to $-n+2r$ and fixes $\mathbb{Z} \setminus r(m)$ pointwise. The residue class `ResidueClass(r,m)` itself can be given in place of the arguments `r` and `m`. Enclosing the argument list in list brackets is permitted.

Example

```
gap> Display(ClassReflection(5,9));

Bijective rcwa mapping of Z with modulus 9, of order 2

                n mod 9                |        n^ClassReflection(5,9)
---------------------------------------+--------------------------------------
  0 1 2 3 4 6 7 8                      | n
  5                                    | -n + 10
```

2.2.4 ClassTransposition (r1, m1, r2, m2)

◊ `ClassTransposition( r1, m1, r2, m2 )` (function)

Returns: The *class transposition* $\tau_{r_1(m_1),r_2(m_2)}$.

The *class transposition* $\tau_{r_1(m_1),r_2(m_2)}$ is an rcwa mapping of $\mathbb{Z}$ of order 2 which interchanges the disjoint residue classes $r_1(m_1)$ and $r_2(m_2)$ of $\mathbb{Z}$ and fixes the complement of their union pointwise. The residue classes `ResidueClass(r1,m1)` and `ResidueClass(r2,m2)` themselves can be given in place of the arguments `r1`, `m1`, `r2` and `m2`. Enclosing the argument list in list brackets is permitted.

Example

```
gap> Display(ClassTransposition(1,2,8,10));

Bijective rcwa mapping of Z with modulus 10, of order 2

               n mod 10               |    n^ClassTransposition(1,2,8,10)
---------------------------------------+--------------------------------------
   0  2  4  6                         | n
   1  3  5  7  9                      | 5n + 3
   8                                  | (n - 3)/5
```

2.2.5 PrimeSwitch (p)

◊ `PrimeSwitch( p )` (function)

◊ `PrimeSwitch( p, k )` (function)

Returns: In the one-argument form the *prime switch* $\sigma_p := \tau_{0(8),1(2p)} \cdot \tau_{4(8),-1(2p)} \cdot \tau_{0(4),1(2p)} \cdot \tau_{2(4),-1(2p)} \cdot \tau_{2(2p),1(4p)} \cdot \tau_{4(2p),2p+1(4p)}$, and in the two-argument form the restriction of σ_p by $n \mapsto kn$ (cp. `Restriction` (3.7.1)).

For an odd prime p, the *prime switch* σ_p is a bijective rcwa mapping of $\mathbb{Z}$ with modulus $4p$, multiplier p (see `Multiplier` (2.6.1)) and divisor 2 (see `Divisor` (2.6.2)). The prime switches σ_p play an important role in factoring non-balanced rcwa mappings into class shifts, class reflections and class transpositions (cp. `FactorizationIntoGenerators` (2.4.1)).

Example

```
gap> Display(PrimeSwitch(3));

Wild bijective rcwa mapping of Z with modulus 12

               n mod 12               |            n^PrimeSwitch(3)
---------------------------------------+--------------------------------------
   0                                  | n/2
   1  7                               | n + 1
   2  6 10                            | (3n + 4)/2
   3  9                               | n
   4                                  | n - 3
   5  8 11                            | n - 1
```

In most cases an rcwa mapping is not determined uniquely by the output of the `ViewObj` method – in these cases the output is enclosed in brackets. There are methods installed for `Display`, `Print` and `String`. The `Print`ed representation of an rcwa mapping is GAP - readable if and only if the `Print`ed representation of the elements of the underlying ring is so. There is also a method for `LaTeXObj`:

2.2.6 LaTeXObj (f)

◊ `LaTeXObj( f )` (method)

Returns: A LaTeX representation of the integral rcwa mapping `f`.

The output makes use of the LaTeX macro package amsmath. If the option `Factorization` is set, a factorization of `f` into class shifts, class reflections, class transpositions and prime switches is printed (cp. `FactorizationIntoGenerators` (2.4.1)). For rcwa mappings with modulus larger than 1, an indentation by `Indentation` characters can be specified by setting this option value accordingly.

Example

```
gap> Print(LaTeXObj(a));
n \ \longmapsto \
\begin{cases}
  \frac{3n}{2}     & \text{if} \ n \in 0(2), \\
  \frac{3n + 1}{4} & \text{if} \ n \in 1(4), \\
  \frac{3n - 1}{4} & \text{if} \ n \in 3(4).
\end{cases}
gap> Print(LaTeXObj(Comm(a,ClassShift(0,4)):Factorization));
       \nu_{8(12)} \cdot \nu_{0(12)}^{-1}
 \cdot \tau_{0(12),6(12)} \cdot \tau_{0(12),4(12)}
 \cdot \tau_{0(12),8(12)}
```

2.3 Basic functionality for rcwa mappings

Checking whether two rcwa mappings are equal is cheap. Rcwa mappings can be multiplied, thus there is a method for `*`. Bijective rcwa mappings can also be inverted, thus there is a method for `Inverse`. The latter method is usually accessed by raising a mapping to some power with negative exponent. Multiplying, inverting and exponentiating tame rcwa mappings is cheap. Computing powers of wild mappings is usually expensive – runtime and memory requirements normally grow approximately exponentially with the exponent. How expensive multiplying a couple of wild mappings is, varies very much. In any case, the amount of memory required for storing an rcwa mapping is proportional to its modulus. Whether a given mapping is tame or wild can be determined by the operation `IsTame`. There are methods for `Order`, which can not only compute a finite order, but are also suitable for detecting infinite order.

Example

```
gap> List([-6..6],k->Modulus(f^k));
[ 324, 108, 108, 36, 36, 12, 1, 12, 24, 48, 96, 192, 384 ]
gap> Order(f);
infinity
```

Example

```
gap> List( [ a, u, f ], IsTame );
[ false, false, false ]
gap> f^2*u;
<bijective rcwa mapping of Z with modulus 120>
gap> f^2*u*a;
<bijective rcwa mapping of Z with modulus 240>
gap> f^2*u*a^2*f^-1;
<bijective rcwa mapping of Z with modulus 3840>
gap> Comm(f,ClassShift(6,12)*f)^1000;
<bijective rcwa mapping of Z with modulus 18>
```

There are methods installed for `IsInjective`, `IsSurjective`, `IsBijective` and `Image`.

Example

```
gap> [ IsInjective(T), IsSurjective(T), IsBijective(u) ];
[ false, true, true ]
gap> Image(RcwaMapping([[-4,-8,1]]));
0(4)
```

Images of elements, of finite sets of elements and of unions of finitely many residue classes of the source of an rcwa mapping can be computed with ^ (the same symbol as used for exponentiation and conjugation). The same works for partitions of the source into a finite number of residue classes.

Example

```
gap> [ 15^T, 7^d, (x^3+x^2+x+One(R))^r ];
[ 23, 7/3, x^3+Z(2)^0 ]
gap> A := ResidueClass(Integers,3,2);;
gap> [ A^T, A^u ];
[ 1(3) U 8(9), 1(9) U 3(9) U 14(27) U 20(27) U 26(27) ]
gap> [rc(0,2),rc(1,4),rc(3,4)]^f;
[ 0(6) U 1(6) U 5(6), 2(12) U 4(12) U 9(12), 3(12) U 8(12) U 10(12) ]
```

For computing preimages of elements under rcwa mappings, there are methods for `PreImageElm` and `PreImagesElm`. The preimage of a finite set of ring elements or of a union of finitely many residue classes under an rcwa mapping can be computed using `PreImage`.

Example

```
gap> [ PreImageElm(d,37/17), PreImagesElm(T,8), PreImagesElm(Zero(T),0) ];
[ 111/17, [ 5, 16 ], Integers ]
gap> PreImage(T,ResidueClass(Integers,3,2));
1(2) U 4(6)
gap> M := [1];; l := [1];;
gap> while Length(M) < 10000 do M := PreImage(T,M); Add(l,Length(M)); od; l;
[ 1, 1, 2, 2, 4, 5, 8, 10, 14, 18, 26, 36, 50, 67, 89, 117, 157, 208, 277,
  367, 488, 649, 869, 1154, 1534, 2039, 2721, 3629, 4843, 6458, 8608, 11472 ]
```

There is a method for the operation `MovedPoints` for computing the support of a bijective rcwa mapping, and there is a method for `RestrictedPerm` for computing the restriction of a bijective rcwa mapping to a union of residue classes it fixes setwisely.

Example
```
gap> [ MovedPoints(u), MovedPoints(u^2) ];
[ Z \ [ -1, 0 ], Z \ [ -10, -6, -1, 0, 1, 2, 3, 5 ] ]
gap> MovedPoints(r);
GF(2)[x] \ [ 0*Z(2), Z(2)^0, x, x+Z(2)^0 ]
gap> RestrictedPerm(f,ResidueClassUnion(Integers,36,[7,8]));
<rcwa mapping of Z with modulus 36>
```

Rcwa mappings can be added and subtracted pointwisely. However, please note that the set of rcwa mappings of some ring does not form a ring under `+` and `*`.

Example
```
gap> a  := RcwaMapping([[3,0,2],[3,1,4],[3,0,2],[3,-1,4]]);;
gap> b  := ClassShift(1,4) * a;;
gap> [ Image((a + b)), Image((a - b)) ];
[ 0(6) U 4(6) U 5(6), [ -3, 0 ] ]
gap> d+d+d;
IdentityMapping( Z_( 2 ) )
```

There are operations `Modulus` (abbreviated `Mod`) and `Coefficients` for extracting the modulus resp. the coefficient list of a given rcwa mapping. The meaning of the return values is as described in the previous section. General documentation for most operations mentioned in this section can be found in the GAP reference manual. For rcwa mappings of rings other than $\mathbb{Z}$, not for all operations applicable methods are available.

2.4 Factoring rcwa mappings

Factoring group elements into elements of some “nice” set of generators is often helpful. The following can be seen as an attempt towards getting a satisfactory solution to this problem for the group RCWA($\mathbb{Z}$):

2.4.1 FactorizationIntoGenerators (g)

◊ `FactorizationIntoGenerators( g )` (attribute)

Returns: A factorization of the bijective rcwa mapping `g` into class shifts, class reflections and class transpositions, provided that such a factorization exists and the method finds it.

This may return `fail`, stop with an error message or run into an infinite loop. If it returns a result, this result is always correct. By default, prime switches are taken as one factor. If the option `ExpandPrimeSwitches` is set, they are each decomposed into the 6 class transpositions given in the definition (see `PrimeSwitch` (2.2.5)). By default, the factoring process begins with splitting off factors from the right. This can be changed by setting the option `Direction` to `"from the left"`. By default, the coarsest possible respected partition of the integral mapping occuring in the final stage of the algorithm is computed. This can be suppressed by setting the option `ShortenPartition` equal to `false`. By default, at the end it is checked whether the product of the determined factors indeed equals `g`. This check can be suppressed by setting the option `NC`.

The problem of obtaining a factorization as desired is algorithmically difficult, and this factorization routine is currently perhaps the most sophisticated part of the RCWA package. Information about the progress of the factorization process can be obtained by setting the info level of the Info class `InfoRCWA` (6.2.1) to 2.

Example

```
gap> FactorizationIntoGenerators(Comm(a,b));
[ ClassShift(7,9), ClassShift(1,9)^-1, ClassTransposition(1,9,4,9),
  ClassTransposition(1,9,7,9), ClassTransposition(6,18,15,18),
  ClassTransposition(5,9,15,18), ClassTransposition(4,9,15,18),
  ClassTransposition(5,9,6,18), ClassTransposition(4,9,6,18) ]
```

For purposes of demonstrating the capabilities of the factorization routine, in Section 4.1 a permutation is factored which has already been mentioned by Lothar Collatz in 1932, and whose cycle structure is unknown so far.

2.5 Determinant and sign

2.5.1 Determinant (sigma)

◊ `Determinant( sigma )` (method)
◊ `Determinant( sigma, S )` (method)

Returns: The determinant of the bijective rcwa mapping `sigma`.

The *determinant* of an affine mapping $n \mapsto (an+b)/c$ whose source is a residue class $r(m)$ is defined by $b/|a|m$. This definition is extended additively to determinants of rcwa mappings and their restrictions to unions of residue classes.

Using the notation from the definition of an rcwa mapping, the *determinant* $\det(\sigma)$ of an rcwa mapping σ is given by

$$\frac{1}{m}\left(\sum_{r(m)\in R/mR} \frac{b_{r(m)}}{|a_{r(m)}|}\right).$$

In the author's forthcoming PhD thesis, it will be proved that the determinant induces an epimorphism from the group of all class-wise order-preserving bijective rcwa mappings of $\mathbb{Z}$ onto $(\mathbb{Z},+)$.

If a residue class union `S` is given as an additional argument, the method returns the determinant of the restriction of `sigma` to `S`.

Example

```
gap> nu := ClassShift(0,1);;
gap> List( [ nu, a, b, u ], Determinant );
[ 1, 0, 1, 0 ]
gap> Determinant(u^2*b^-3);
-3
gap> Determinant(nu^7*a^2*nu^-1*b^-1*a^-3);
5
```

2.5.2 Sign (sigma)

◊ Sign(sigma) (attribute)

Returns: The sign of the bijective rcwa mapping sigma.

Using the notation from the definition of an rcwa mapping, the *sign* of a bijective rcwa mapping σ of $\mathbb{Z}$ is defined by

$$(-1)^{\displaystyle \det(\sigma) + \frac{1}{m}\left(\sum_{r(m):\ a_{r(m)}<0} (m-2r)\right)}.$$

In the author's forthcoming PhD thesis, it will be proved that the sign induces an epimorphism from RCWA($\mathbb{Z}$) onto the group $\mathbb{Z}^{\times}$ of units of $\mathbb{Z}$. This means that the kernel of the sign mapping is a normal subgroup of RCWA($\mathbb{Z}$) of index 2.

Example

```
gap> List( [ nu, nu^2, nu^3 ], Sign );
[ -1, 1, -1 ]
gap> List( [ t, nu^3*t ], Sign );
[ -1, 1 ]
gap> List( [ a, a*b, (a*b)^2, Comm(a,b) ], Sign );
[ 1, -1, 1, 1 ]
```

2.6 Attributes and properties derived from the coefficients

2.6.1 Multiplier (f)

◊ Multiplier(f) (attribute)

◊ Mult(f) (attribute)

Returns: The multiplier of the rcwa mapping f.

In the notation used in the definition of an rcwa mapping, the *multiplier* is the lcm of the coefficients $a_{r(m)}$ in the numerators.

Example

```
gap> List( [ g, u, T, d, r ], Multiplier );
[ 1, 9, 3, 1, x^2+x+Z(2)^0 ]
```

2.6.2 Divisor (f)

◊ Divisor(f) (attribute)

◊ Div(f) (attribute)

Returns: The divisor of the rcwa mapping f.

In the notation used in the definition of an rcwa mapping, the *divisor* is the lcm of the coefficients $c_{r(m)}$ in the denominators.

Example

```
gap> List( [ g, u, T, d, r ], Divisor );
[ 1, 5, 2, 1, x^2+Z(2)^0 ]
```

2.6.3 PrimeSet (f)

◊ `PrimeSet( f )` (operation)

Returns: The prime set of the rcwa mapping `f`.

The *prime set* of an rcwa mapping is the set of prime divisors of the product of its modulus, its multiplier and its divisor. See also `PrimeSet` (3.2.3) for rcwa groups.

Example
```
gap> PrimeSet(T);
[ 2, 3 ]
gap> List( [ u, T^u, T^(u^-1) ], PrimeSet );
[ [ 3, 5 ], [ 2, 3 ], [ 2, 3, 5 ] ]
gap> PrimeSet(r);
[ x+Z(2)^0, x^2+x+Z(2)^0 ]
```

2.6.4 IsIntegral (f)

◊ `IsIntegral( f )` (property)

Returns: `true` if the rcwa mapping `f` is integral and `false` otherwise.

An rcwa mapping is called *integral* if its divisor equals 1, thus “if no proper divisions occur”. Computing with such mappings is particularly easy.

Be careful not to confuse this with the term *integral rcwa mapping* for rcwa mappings of the integers; normally it should be rather clear what is meant.

Example
```
gap> List( [ u, t, RcwaMapping([[2,0,1],[3,5,1]]) ], IsIntegral );
[ false, true, true ]
```

2.6.5 IsClassWiseOrderPreserving (f)

◊ `IsClassWiseOrderPreserving( f )` (property)

Returns: `true` if the rcwa mapping `f` is class-wise order-preserving and `false` otherwise.

The term *class-wise order-preserving* is defined only for rcwa mappings of ordered rings, e.g. $\mathbb{Z}$.

Example
```
gap> List( [ g, u, T, t, d ], IsClassWiseOrderPreserving );
[ true, true, true, false, true ]
```

2.7 Functionality related to the affine partial mappings

2.7.1 LargestSourcesOfAffineMappings (f)

◊ LargestSourcesOfAffineMappings(f) (attribute)

Returns: The coarsest partition of `Source(f)` on whose elements the rcwa mapping `f` is affine.

Example

```
gap> LargestSourcesOfAffineMappings(T);
[ 0(2), 1(2) ]
gap> List( [ u, u^-1 ], LargestSourcesOfAffineMappings );
[ [ 0(5), 1(5), 2(5), 3(5), 4(5) ], [ 0(3), 1(3), 2(9), 5(9), 8(9) ] ]
gap> LargestSourcesOfAffineMappings(t);
[ Integers ]
gap> kappa := RcwaMapping([[1,0,1],[1,0,1],[3,2,2],[1,-1,1],
>                          [2,0,1],[1,0,1],[3,2,2],[1,-1,1],
>                          [1,1,3],[1,0,1],[3,2,2],[2,-2,1]]);;
gap> SetName(kappa,"kappa");
gap> LargestSourcesOfAffineMappings(kappa);
[ 2(4), 1(4) U 0(12), 3(12) U 7(12), 4(12), 8(12), 11(12) ]
gap> LargestSourcesOfAffineMappings(r);
[ 0*Z(2) ( mod x^2+Z(2)^0 ), Z(2)^0 ( mod x^2+Z(2)^0 ), x ( mod x^2+Z(2)^0 ),
  x+Z(2)^0 ( mod x^2+Z(2)^0 ) ]
```

2.7.2 Multpk (f, p, k)

◊ Multpk(f, p, k) (operation)

Returns: The union of the residue classes $r(m)$ such that $p^k||a_{r(m)}$ if $k \geq 0$, and the union of the residue classes $r(m)$ such that $p^k||c_{r(m)}$ if $k \leq 0$.

Example

```
gap> [ Multpk(T,2,-1), Multpk(T,3,1) ];
[ Integers, 1(2) ]
gap> [ Multpk(u,3,0), Multpk(u,3,1), Multpk(u,3,2), Multpk(u,5,-1) ];
[ [  ], 0(5) U 2(5), 1(5) U 3(5) U 4(5), Integers ]
gap> [ Multpk(kappa,2,1), Multpk(kappa,2,-1), Multpk(kappa,3,1),
>      Multpk(kappa,3,-1) ];
[ 4(12) U 11(12), 2(4), 2(4), 8(12) ]
```

2.7.3 SetOnWhichMappingIsClassWiseOrderPreserving (f)

◊ `SetOnWhichMappingIsClassWiseOrderPreserving( f )` (attribute)
◊ `SetOnWhichMappingIsClassWiseConstant( f )` (attribute)
◊ `SetOnWhichMappingIsClassWiseOrderReversing( f )` (attribute)

Returns: The union of the residue classes (mod `Modulus(f)`) on which the rcwa mapping `f` is class-wise order-preserving, class-wise constant resp. class-wise order-reversing.

The source of the rcwa mapping `f` must be ordered.

Example

```
gap> List( [ T, u, t ], SetOnWhichMappingIsClassWiseOrderPreserving );
[ Integers, Integers, [  ] ]
gap> SetOnWhichMappingIsClassWiseConstant(RcwaMapping([[2,0,1],[0,4,1]]));
1(2)
```

2.8 Transition graphs and transition matrices

2.8.1 TransitionGraph (f, m)

◊ `TransitionGraph( f, m )` (operation)

Returns: The transition graph for modulus `m` of the rcwa mapping `f`.

The *transition graph* $\Gamma_{f,m}$ of f for modulus m is defined as follows:

1. The vertices are the residue classes (mod m).
2. There is an edge from $r_1(m)$ to $r_2(m)$ if and only if there is some $n_1 \in r_1(m)$ such that $n_1^f \in r_2(m)$.

The assignment of the residue classes (mod m) to the vertices of the graph is given by the ordering of the residues in `AllResidues(Source(f),m)`. The result is returned as a GRAPE-graph.

Example

```
gap> TransitionGraph(a,Modulus(a));
rec( isGraph := true, order := 4, group := Group(()),
  schreierVector := [ -1, -2, -3, -4 ],
  adjacencies := [ [ 1, 3 ], [ 1, 2, 3, 4 ], [ 2, 4 ], [ 1, 2, 3, 4 ] ],
  representatives := [ 1, 2, 3, 4 ], names := [ 1, 2, 3, 4 ] )
```

2.8.2 OrbitsModulo (f, m)

◊ `OrbitsModulo( f, m )` (operation)

Returns: The partition of `AllResidues(Source(f),m)` corresponding to the weakly-connected components of the transition graph for modulus `m` of the rcwa mapping `f`.

See also `OrbitsModulo` (3.5.6) for rcwa groups.

Example

```
gap> OrbitsModulo(Comm(a,b),9);
[ [ 0 ], [ 1, 4, 5, 6, 7 ], [ 2 ], [ 3 ], [ 8 ] ]
```

2.8.3 FactorizationOnConnectedComponents (f, m)

◊ `FactorizationOnConnectedComponents( f, m )` (operation)

Returns: The set of restrictions of the rcwa mapping `f` to the weakly-connected components of its transition graph $\Gamma_{f,m}$.

These mappings have pairwise disjoint supports, hence any two of them commute, and their product equals `f`.

Example

```
gap> sigma := RcwaMapping([[1, 0,1],[1, 1,1],[2,2,1],[3,-3,2],
>                          [1, 0,1],[1,-2,3],[3,6,2],[1,-2,1],
>                          [1, 0,1],[1, 1,1],[1,1,1],[1,-2,1],
>                          [2, 0,1],[1, 1,1],[1,1,1],[3,-3,2],
>                          [1, 0,1],[1, 1,1],[3,6,2],[1,-2,1],
>                          [1, 0,1],[1, 1,1],[1,1,1],[2,-4,1],
>                          [1,-3,3],[1, 1,1],[1,1,1],[3,-3,2],
>                          [1, 0,1],[2, 2,1],[3,6,2],[1,-2,1],
>                          [1, 0,1],[1, 1,1],[1,1,1],[1,-2,1]]);
<rcwa mapping of Z with modulus 36>
gap> fact := FactorizationOnConnectedComponents(sigma,36);
[ <rcwa mapping of Z with modulus 36>, <rcwa mapping of Z with modulus 36>,
  <rcwa mapping of Z with modulus 36> ]
gap> List(fact,MovedPoints);
[ 33(36) U 34(36) U 35(36), 9(36) U 10(36) U 11(36),
  <union of 23 residue classes (mod 36)> \ [ -6, 3 ] ]
```

2.8.4 TransitionMatrix (f, m)

◊ `TransitionMatrix( f, m )` (function)

Returns: The transition matrix of the rcwa mapping `f` for modulus `m`.

Let M be this matrix. Then for any two residue classes $r_1(m), r_2(m) \in R/mR$, the entry $M_{r_1(m),r_2(m)}$ is defined by

$$M_{r_1(m),r_2(m)} \;:=\; \frac{|R/mR|}{|R/\hat{m}R|} \cdot \left|\left\{r(\hat{m}) \in R/\hat{m}R \,|\, r \in r_1(m) \wedge r^f \in r_2(m)\right\}\right|,$$

where $\hat{m}$ is the product of `m` and the square of the modulus of `f`. The assignment of the residue classes (mod `m`) to the rows and columns of the matrix is given by the ordering of the residues in `AllResidues(Source(f),m)`.

The transition matrix is a weighted adjacency matrix of the corresponding transition graph `TransitionGraph(f,m)`. The sums of the rows of a transition matrix are always equal to 1.

Example

```
gap> Display(TransitionMatrix(a,5));
[ [  1/2,  1/4,    0,    0,  1/4 ],
  [    0,  1/4,    0,  1/4,  1/2 ],
  [  1/4,    0,    0,  3/4,    0 ],
  [  1/4,    0,  3/4,    0,    0 ],
  [    0,  1/2,  1/4,    0,  1/4 ] ]
```

Example

```
gap> Display(TransitionMatrix(T,19)*One(GF(7)));
 4 . . . . . . . . . 4 . . . . . . . .
 . . 4 . . . . . . . 4 . . . . . . . .
 . 4 . . . . . . . . . . . 4 . . . . .
 . . . . . 4 . . . . . 4 . . . . . . .
 . . 4 . . . . . . . . . . . . . 4 . .
 . . . . . . . . 4 . . . 4 . . . . . .
 4 . . 4 . . . . . . . . . . . . . . .
 . . . . . . . . . . . 4 . 4 . . . . .
 . . . 4 4 . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . 1 . . . .
 . . . . . 4 4 . . . . . . . . . . . .
 . . . . . . . . . . . . . . . 4 . 4 .
 . . . . . . 4 . . 4 . . . . . . . . .
 . 4 . . . . . . . . . . . . . . 4 . .
 . . . . . . . 4 . . . . 4 . . . . . .
 . . . . 4 . . . . . . . . . . . . 4 .
 . . . . . . . . 4 . . . . . . 4 . . .
 . . . . . . . 4 . . . . . . . . . . 4
 . . . . . . . . . 4 . . . . . . . . 4
```

2.9 Trajectories

2.9.1 Trajectory (f, n, val, cond)

◊ `Trajectory( f, n, val, cond )` (function)

Returns: The trajectory of the ring element `n` under the rcwa mapping `f`.

Depending on whether `cond = "length"` or `cond = "stop"`, the parameter `val` either is the length of the sequence to be computed or is a "stopping set" such that the computation stops when some iterate `n^(f^k) in val` is reached. In place of the ring element `n`, a union of residue classes is permitted, also.

Example

```
gap> Trajectory(T,27,[1],"stop");
[ 27, 41, 62, 31, 47, 71, 107, 161, 242, 121, 182, 91, 137, 206, 103, 155,
  233, 350, 175, 263, 395, 593, 890, 445, 668, 334, 167, 251, 377, 566, 283,
  425, 638, 319, 479, 719, 1079, 1619, 2429, 3644, 1822, 911, 1367, 2051,
  3077, 4616, 2308, 1154, 577, 866, 433, 650, 325, 488, 244, 122, 61, 92, 46,
  23, 35, 53, 80, 40, 20, 10, 5, 8, 4, 2, 1 ]
gap> Trajectory(T,ResidueClass(Integers,3,0),Integers,"stop");
[ 0(3), 0(3) U 5(9), 0(3) U 5(9) U 7(9) U 8(27),
  <union of 20 residue classes (mod 27)>, <union of 73 residue classes (mod
    81)>, <union of 79 residue classes (mod 81)>, Integers ]
gap> Length(Trajectory(RcwaMapping([[1,0,2],[5,-1,2]]),19,[1],"stop"));
307
```

2.9.2 TrajectoryModulo (f, n, m, lng)

◊ `TrajectoryModulo( f, n, m, lng )` (function)
◊ `TrajectoryModulo( f, n, lng )` (function)

Returns: The sequence $(n_i), i = 0, \dots, lng-1$ with $n_i := n^{(f^i)}$ mod `m` as a list.

If `m` is not given it defaults to the modulus of `f`.

Example

```
gap> TrajectoryModulo(a,8,25);
[ 0, 0, 2, 3, 0, 2, 1, 2, 3, 2, 1, 3, 0, 0, 0, 0, 2, 3, 2, 1, 1, 2, 3, 1, 2 ]
gap> TrajectoryModulo(T,27,2,100);
[ 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0,
  0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0,
  1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1,
  0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0 ]
```

2.9.3 CoefficientsOnTrajectory (f, n, val, cond, all)

◊ `CoefficientsOnTrajectory( f, n, val, cond, all )` (function)

Returns: A list `c` of coefficient triples, such that for any `k` it holds that `n^(f^(k-1)) = (c[k][1]*n + c[k][2])/c[k][3]`, or its last entry.

The meaning of the arguments `val` and `cond` is the same as in `Trajectory` (2.9.1). If `all = true`, the whole sequence of coefficient triples is returned. Otherwise the result is only the last triple.

Example

```
gap> CoefficientsOnTrajectory(T,27,1,"stop",false);
[ 36472996377170786403, 195820718533800070543, 1180591620717411303424 ]
gap> (last[1]*27+last[2])/last[3];
1
gap> CoefficientsOnTrajectory(sigma,37,37,"stop",true);
[ [ 1, 0, 1 ], [ 1, 1, 1 ], [ 2, 4, 1 ], [ 3, 9, 1 ], [ 6, 18, 1 ],
  [ 2, 5, 1 ], [ 2, 3, 1 ], [ 2, 1, 3 ], [ 2, 4, 3 ], [ 2, 7, 3 ],
  [ 1, 2, 1 ], [ 3, 3, 2 ], [ 3, 5, 2 ], [ 3, 7, 2 ], [ 3, 3, 1 ],
  [ 9, 15, 2 ], [ 27, 57, 4 ], [ 27, 57, 2 ], [ 9, 17, 2 ], [ 9, 13, 2 ],
  [ 9, 15, 1 ], [ 3, 4, 1 ], [ 3, 2, 1 ], [ 1, 0, 1 ] ]
gap> List(last,c->(c[1]*37+c[2])/c[3]){[1..23]} = Cycle(sigma,37);
true
gap> CoefficientsOnTrajectory(r,x^3+x^2,x^3+x^2,"stop",true);
[ [ Z(2)^0, 0*Z(2), Z(2)^0 ], [ x^2+x+Z(2)^0, x^2+x, x^2+Z(2)^0 ],
  [ x^4+x^2+Z(2)^0, x^4+x, x^4+Z(2)^0 ],
  [ x^6+x^5+x^3+x+Z(2)^0, x^6+x^4+x^3+x^2, x^6+x^4+x^2+Z(2)^0 ],
  [ x^8+x^4+Z(2)^0, x^7+x^6, x^8+Z(2)^0 ] ]
```

2.9.4 IncreasingOn (f)

◊ IncreasingOn(f) (function)
◊ DecreasingOn(f) (function)

Returns: The union of all residue classes $r(m)$ such that $|R/a_{r(m)}R| > |R/c_{r(m)}R|$ resp. $|R/a_{r(m)}R| < |R/c_{r(m)}R|$, where R denotes the source, m the modulus and $a_{r(m)}$, $b_{r(m)}$ and $c_{r(m)}$ the coefficients of `f` as introduced in the definition of an rcwa mapping.

Example
```
gap> List([1..3],k->IncreasingOn(T^k));
[ 1(2), 3(4), 3(4) U 1(8) U 6(8) ]
gap> List([1..3],k->DecreasingOn(T^k));
[ 0(2), 0(2) U 1(4), 0(4) U 2(8) U 5(8) ]
gap> List([1..3],k->IncreasingOn(a^k));
[ 0(2), 0(2) U 3(8) U 5(8), 0(4) U 2(16) U 5(16) U 11(16) U 14(16) ]
```

2.10 Special functions for non-bijective mappings and miscellanea

2.10.1 LikelyContractionCentre (f, maxn, bound)

◊ LikelyContractionCentre(f, maxn, bound) (operation)

Returns: A list of ring elements – see below.

Tries to compute the *contraction centre* of an rcwa mapping - assuming its existence this is the uniquely-defined finite subset S_0 of the base ring R which is mapped bijectively onto itself under `f` and where for any x in R there is an integer k such that the image of x under the k-th power of f lies in S_0. The mapping `f` is assumed to be *contracting*, i.e. to have such a contraction centre. As this problem seems to be computationally undecidable methods will be probabilistic. The argument `maxn` is a bound on the starting value and `bound` is a bound on the elements of the sequences to be searched. If the limit `bound` is exceeded, an Info message on some Info level of `InfoRCWA` is given.

Example
```
gap> S0 := LikelyContractionCentre(T,100,1000);
#I  Warning: `LikelyContractionCentre' is highly probabilistic.
The returned result can only be regarded as a rough guess.
See ?LikelyContractionCentre for information on how to improve this guess.
[ -136, -91, -82, -68, -61, -55, -41, -37, -34, -25, -17, -10, -7, -5, -1, 0,
  1, 2 ]
```

2.10.2 GuessedDivergence (f)

◊ GuessedDivergence(f) (operation)

Returns: A floating point value, which should be a rough guess on how fast the trajectories of the rcwa mapping f diverge (return value greater than 1) or converge (return value smaller than 1).

Nothing particular is guaranteed.

Example

```
gap> List( [ T, a ], GuessedDivergence );
#I  Warning: GuessedDivergence: no particular return value is guaranteed.
#I  Warning: GuessedDivergence: no particular return value is guaranteed.
[ 0.866025, 1.06066 ]
```

2.10.3 ImageDensity (f)

◊ ImageDensity(f) (attribute)

Returns: The *image density* of the rcwa mapping f.

In the notation introduced in the definition of an rcwa mapping, the *image density* of f is defined by $\frac{1}{m}\sum_{r(m)\in R/mR}|R/c_{r(m)}R|/|R/a_{r(m)}R|$. Injective rcwa mappings have an image density of at most 1, and the image density of a surjective rcwa mapping is at least 1 (this can be seen easily) – thus in particular the image density of a bijective mapping equals 1.

Example

```
gap> List( [ T, a, RcwaMapping([[2,0,1]]) ], ImageDensity );
[ 4/3, 1, 1/2 ]
```

2.11 The categories and families of rcwa mappings

2.11.1 IsRcwaMapping (f)

◊ IsRcwaMapping(f) (filter)
◊ IsIntegralRcwaMapping(f) (filter)
◊ IsSemilocalIntegralRcwaMapping(f) (filter)
◊ IsModularRcwaMapping(f) (filter)

Returns: true if f is an rcwa mapping resp. an rcwa mapping of the ring of integers resp. an rcwa mapping of a semilocalization of the ring of integers resp. an rcwa mapping of a polynomial ring in one variable over a finite field, and false otherwise.

2.11.2 RcwaMappingsFamily (R)

◊ RcwaMappingsFamily(R) (function)

Returns: The family of rcwa mappings of the ring R.

Chapter 3

Residue Class-Wise Affine Groups

This chapter describes the functionality provided by this package for computing with residue class-wise affine groups.

3.1 Constructing residue class-wise affine groups

Residue class-wise affine groups can be constructed using either `Group`, `GroupByGenerators` or `GroupWithGenerators`, as usual (see the reference manual). Note that currently except of the groups RCWA(R) only finitely generated rcwa groups are supported.

Example

```
gap> g := RcwaMapping([[1,0,1],[1,1,1],[3,6,1],
>                      [1,0,3],[1,1,1],[3,6,1],
>                      [1,0,1],[1,1,1],[3,-21,1]]);;
gap> h := RcwaMapping([[1,0,1],[1,1,1],[3,6,1],
>                      [1,0,3],[1,1,1],[3,-21,1],
>                      [1,0,1],[1,1,1],[3,6,1]]);;
gap> List([g,h],Order);
[ 9, 9 ]
gap> G := Group(g,h);
<rcwa group over Z with 2 generators>
gap> Size(G);
infinity
```

For rcwa groups, there are methods available for the operations `Display` and `Print`.

All rcwa groups over the ring R are subgroups of RCWA(R). The group RCWA(R) is not finitely generated, thus cannot be constructed as described above. It is handled as a special case:

3.1.1 RCWA (R)

◊ `RCWA( R )` (function)

Returns: The group RCWA(`R`) of all bijective rcwa mappings of the ring `R`.

Example

```
gap> RCWA_Z := RCWA(Integers);
RCWA(Z)
gap> Size(RCWA_Z);
infinity
gap> IsFinitelyGeneratedGroup(RCWA_Z);
false
gap> One(RCWA_Z);
IdentityMapping( Integers )
gap> IsSolvable(RCWA_Z);
false
gap> IsPerfect(RCWA_Z);
false
gap> Centre(RCWA_Z);
Trivial rcwa group over Z
gap> IsSubgroup(RCWA_Z, Group(RcwaMapping((1,2,3),[1..4]),
>                            RcwaMapping(2,[[0,1],[1,0],[2,3],[3,2]])));
true
```

3.1.2 Random (RCWA(Integers))

◊ Random(RCWA_Z) (method)

Returns: A pseudo-random element of the group RCWA($\mathbb{Z}$).

This method is designed to be suitable for generating interesting examples. No particular distribution is guaranteed – in fact, the author has no idea what a “reasonable” random distribution on RCWA($\mathbb{Z}$) should be.

Example

```
gap> elm := Random(RCWA_Z);
<bijective rcwa mapping of Z with modulus 60>
gap> Display(elm);

Bijective rcwa mapping of Z with modulus 60

               n mod 60                |                n^f
---------------------------------------+---------------------------------------
   0  4  6  8 10 14 16 18 20 24 26 28  |
  30 34 36 38 40 44 46 48 50 54 56 58  | -3n - 5
   1                                   | (n - 1)/5
   2 22 42                             | (3n + 24)/5
   3  9 15 21 27 33 39 45 51 57        | (-5n + 12)/3
   5 11 17 29 35 41 47 59              | -n + 2
   7 13 19 25 37 43 49 55              | -n
  12 32 52                             | (3n + 4)/5
  23                                   | (n - 3)/5
  31                                   | (n + 19)/5
  53                                   | (n + 17)/5
```

The elements returned by this method are obtained by multiplying class shifts (see `ClassShift` (2.2.2)), class reflections (see `ClassReflection` (2.2.3)) and class transpositions (see `ClassTransposition` (2.2.4)). These factors are stored as an attribute value:

Example

```
gap> Perform(FactorizationIntoGenerators(elm),Display);

Rcwa mapping of Z with modulus 6

                n mod 6                |                 n^f
---------------------------------------+---------------------------------------
  0 2 4                                | 3n + 5
  1 3                                  | n
  5                                    | (n - 5)/3


Rcwa mapping of Z with modulus 6

                n mod 6                |                 n^f
---------------------------------------+---------------------------------------
  0 2 4                                | 3n + 3
  1 5                                  | n
  3                                    | (n - 3)/3


Rcwa mapping of Z with modulus 10

               n mod 10                |                 n^f
---------------------------------------+---------------------------------------
   0  2  4  6  8                       | 5n + 1
   1                                   | (n - 1)/5
   3  5  7  9                          | n


Rcwa mapping of Z with modulus 4

                n mod 4                |                 n^f
---------------------------------------+---------------------------------------
  0 1 3                                | n
  2                                    | n + 4

Rcwa mapping of Z: n -> -n

Rcwa mapping of Z with modulus 2

                n mod 2                |                 n^f
---------------------------------------+---------------------------------------
  0                                    | -n
  1                                    | n
```

Another possible way to get an rcwa group is by taking the image of an rcwa representation, or by "translating" a permutation group:

3.1.3 IsomorphismRcwaGroup (G)

◊ `IsomorphismRcwaGroup( G )` (attribute)
◊ `RcwaGroupByPermGroup( G )` (function)

Returns: A monomorphism from the finite group `G` to RCWA($\mathbb{Z}$), resp. an rcwa group over $\mathbb{Z}$, which is isomorphic to the (finite) permutation group `G`, and acts on the range `[1..LargestMovedPoint(G)]` as `G` does.

Example
```
gap> G := GL(2,5);;
gap> IsomorphismRcwaGroup(G);
CompositionMapping(
[ (2,3,5,8)(4,7,12,18)(6,10,17,22)(9,15,21,24)(13,14,19,23),
  (1,2,4)(3,6,11)(5,9,16)(7,13,12)(8,14,20)(10,18,17)(15,22,21)(19,24,23)
 ] -> [ <rcwa mapping of Z with modulus 24>, <rcwa mapping of Z with modulus
    24> ], <action isomorphism> )
gap> H := RcwaGroupByPermGroup(Group((1,2),(3,4),(5,6),(7,8),
>                                     (1,3)(2,4),(1,3,5,7)(2,4,6,8)));
<rcwa group over Z with 6 generators>
gap> Size(H);; List(DerivedSeries(H),Size);
[ 384, 96, 32, 2, 1 ]
```

3.2 Attributes and properties of rcwa groups

3.2.1 Modulus (G)

◊ `Modulus( G )` (method)
◊ `Mod( G )` (method)

Returns: The modulus of the rcwa group `G`.

The *Modulus* of an rcwa group is the lcm of the moduli of its elements in case such an lcm exists and 0 otherwise.

See also `IsTame` (3.2.2).

Example
```
gap> Modulus(Group(g,h));
27
gap> g1 := RcwaMapping((1,2),[1..2]);
<rcwa mapping of Z with modulus 2>
gap> g2 := RcwaMapping((1,2,3),[1..3]);
<rcwa mapping of Z with modulus 3>
gap> g3 := RcwaMapping((1,2,3,4,5),[1..5]);
<rcwa mapping of Z with modulus 5>
```

Example

```
gap> G := Group(g1,g2,g3);
<rcwa group over Z with 3 generators>
gap> Modulus(G);
30
gap> a := RcwaMapping([[3,0,2],[3,1,4],[3,0,2],[3,-1,4]]);; SetName(a,"a");
gap> b := ClassShift(1,4)*a;; SetName(b,"b");
gap> c := ClassShift(3,4)*a;; SetName(c,"c");
gap> Modulus(Group(a,b));
0
```

3.2.2 IsTame (G)

◊ `IsTame( G )` (property)

Returns: `true` if the rcwa group `G` is tame and `false` otherwise.

An rcwa group is called *tame* if its modulus is not equal to 0.

Example

```
gap> IsTame(G);
true
gap> IsTame(Group(a,b));
false
gap> IsTame(Group(Comm(a,b),Comm(a,c)));
true
```

3.2.3 PrimeSet (G)

◊ `PrimeSet( G )` (operation)

Returns: The prime set of the rcwa group `G`.

The *prime set* of an rcwa group is the union of the prime sets of its elements. See also `PrimeSet` (2.6.3) for rcwa mappings.

Example

```
gap> [ PrimeSet(G), PrimeSet(H) ];
[ [ 2, 3, 5 ], [ 2 ] ]
```

An rcwa group is called integral resp. class-wise order-preserving if all of its elements are so. There are corresponding methods available for `IsIntegral` and `IsClassWiseOrderPreserving`.

3.3 Membership testing, order computation, permutation- / matrix representations

3.3.1 \in (g, G)

◊ \in(g, G) (method)

Returns: true if the rcwa mapping g is an element of the rcwa group G and false if not.

Tries to figure out whether g is an element of G or not. This will be successful if G is tame and class-wise order-preserving. For wild groups only a number of easy cases are covered. On InfoLevel(InfoRCWA) = 3 the method will give information about reasons why g is an element of G or not.

Example

```
gap> u := RcwaMapping([[3,0,5],[9,1,5],[3,-1,5],[9,-2,5],[9,4,5]]);;
gap> u in H;
false
```

3.3.2 Size (G)

◊ Size(G) (method)

Returns: The order of the rcwa group G.

Example

```
gap> Size(G);
265252859812191058636308480000000
```

3.3.3 IsomorphismPermGroup (G)

◊ IsomorphismPermGroup(G) (method)

Returns: An isomorphism from the finite rcwa group G to some permutation group.

Example

```
gap> H := Group(g1,g2);
<rcwa group over Z with 2 generators>
gap> phi := IsomorphismPermGroup(H);
[ <bijective rcwa mapping of Z with modulus 2, of order 2>,
  <bijective rcwa mapping of Z with modulus 3, of order 3> ] ->
[ (1,2)(3,4)(5,6), (1,2,3)(4,5,6) ]
```

3.3.4 IsomorphismMatrixGroup (G)

◊ `IsomorphismMatrixGroup( G )` (attribute)

Returns: An isomorphism from the rcwa group `G` to some matrix group.

Currently, there is only a method installed which works for tame rcwa groups.

Example

```
gap> g := RcwaMapping([[2,2,1],[1, 4,1],[1,0,2],[2,2,1],[1,-4,1],[1,-2,1]]);;
gap> h := RcwaMapping([[2,2,1],[1,-2,1],[1,0,2],[2,2,1],[1,-1,1],[1, 1,1]]);;
gap> SetName(g,"g"); SetName(h,"h");
gap> G := Group(g,h);
<rcwa group over Z with 2 generators>
gap> phi := IsomorphismMatrixGroup(G);;
gap> FieldOfMatrixGroup(Image(phi));
Rationals
gap> DegreeOfMatrixGroup(Image(phi));
28
gap> List(GeneratorsOfGroup(Image(phi)),Order);
[ 7, 12 ]
gap> Display(GeneratorsOfGroup(Image(phi))[1]*One(GF(5)));
 . . . . . . 1 1 . . . . . . . . . . . . . . . . . . . .
 . . . . . . . 1 . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . 3 . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . 1 . . . . . .
 . . . . . . . . 1 3 . . . . . . . . . . . . . . . . . .
 . . . . . . . . . 1 . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . 3 . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . 1 . . . .
 . . 1 4 . . . . . . . . . . . . . . . . . . . . . . . .
 . . . 1 . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . . . 1 1 . . . . . . . . . .
 . . . . . . . . . . . . . . . . . 1 . . . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . 3 . . .
 . . . . . . . . . . . . . . . . . . . . . . . . . 1 . .
 . . . . . . . . . . . . . . . . . . 1 3 . . . . . . . .
 . . . . . . . . . . . . . . . . . . . 1 . . . . . . . .
 . . . . . . . . . . . . . . . . . . . . . . . . . . 3 .
 . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
 . . . . . . . . . . . . 1 4 . . . . . . . . . . . . . .
 . . . . . . . . . . . . . 1 . . . . . . . . . . . . . .
 2 2 . . . . . . . . . . . . . . . . . . . . . . . . . .
 . 1 . . . . . . . . . . . . . . . . . . . . . . . . . .
 . . . . 2 2 . . . . . . . . . . . . . . . . . . . . . .
 . . . . . 1 . . . . . . . . . . . . . . . . . . . . . .
 . . . . . . . . . . 2 2 . . . . . . . . . . . . . . . .
 . . . . . . . . . . . 1 . . . . . . . . . . . . . . . .
 . . . . . . . . . . . . . . 2 2 . . . . . . . . . . . .
 . . . . . . . . . . . . . . . 1 . . . . . . . . . . . .
```

3.4 Factoring elements into generators

3.4.1 PreImagesRepresentative (phi, g)

◊ `PreImagesRepresentative( phi, g )` (method)

Returns: A representative of the set of preimages of `g` under the homomorphism `phi` from a free group to an rcwa group over $\mathbb{Z}$.

This method can be used for factoring elements of rcwa groups over $\mathbb{Z}$ into generators, and for finding non-trivial relations among the generators if the respective group is not free and the method returns a factorization which does not happen to be equal to one which is known a priori. The homomorphism `phi` must map the generators of the free group to the generators of the rcwa group one-by-one. This method is also suitable for wild groups. The implementation is based on `RepresentativeActionPreImage` (3.5.3).

Example

```
gap> G := Group(g,h);
<rcwa group over Z with 2 generators>
gap> phi := EpimorphismFromFreeGroup(G);
[ g, h ] -> [ g, h ]
gap> PreImagesRepresentative(phi,h*g^3*h^2*g^-1*h*g*h^-3);
h*g^3*h^2*g^-1*h*g*h^-3
gap> nu := RcwaMapping([[1,1,1]]);
Rcwa mapping of Z: n -> n + 1
gap> SetName(nu,"nu");
gap> G := Group(a,nu);
<rcwa group over Z with 2 generators>
gap> IsTame(G);
false
gap> phi := EpimorphismFromFreeGroup(G);
[ a, nu ] -> [ a, nu ]
gap> F := Source(phi);
<free group on the generators [ a, nu ]>
gap> w := Comm(F.1,Comm(F.1,F.2^2));
a^-1*nu^-2*a^-1*nu^2*a*nu^-2*a*nu^2
gap> f := w^phi;
<bijective rcwa mapping of Z with modulus 18>
gap> IsTame(f);
false
gap> pre := PreImagesRepresentative(phi,f);
a^-2*nu^-2*a^2*nu^2
gap> one := w*pre^-1; # pre <> w --> We have a non-trivial relation!
a^-1*nu^-2*a^-1*nu^2*a*nu^-2*a^-1*nu^2*a^2
gap> one^phi;
IdentityMapping( Integers )
```

3.4.2 PreImagesRepresentatives (phi, g)

◊ `PreImagesRepresentatives( phi, g )` (operation)

Returns: A list of representatives of the set of preimages of `g` under the homomorphism `phi` from a free group to an rcwa group over ℤ.

Quite frequently, computing several preimages is not harder than computing just one, i.e. often several preimages are found simultaneously. This operation is called by `PreImagesRepresentative` (3.4.1), which simply chooses the shortest representative – for a slightly more concise description see there.

Example

```
gap> G := Group(g,h);
<rcwa group over Z with 2 generators>
gap> phi := EpimorphismFromFreeGroup(G);
[ g, h ] -> [ g, h ]
gap> f := g^3*h*g^-4*h^5*g;
<bijective rcwa mapping of Z with modulus 12>
gap> RCWAInfo(2);
gap> pre := PreImagesRepresentatives(phi,f);
#I  Orbit lengths after extension step 1: [ 4, 5 ]
#I  |Candidates| = 1
#I  Orbit lengths after extension step 1: [ 5, 5 ]
#I  Orbit lengths after extension step 2: [ 17, 15 ]
#I  Orbit lengths after extension step 3: [ 52, 39 ]
#I  |Candidates| = 1
#I  Orbit lengths after extension step 1: [ 5, 5 ]
#I  Orbit lengths after extension step 2: [ 17, 15 ]
#I  Orbit lengths after extension step 3: [ 53, 43 ]
#I  Orbit lengths after extension step 4: [ 158, 119 ]
#I  |Candidates| = 1
#I  Orbit lengths after extension step 1: [ 5, 5 ]
#I  Orbit lengths after extension step 2: [ 17, 17 ]
#I  Orbit lengths after extension step 3: [ 53, 53 ]
#I  Orbit lengths after extension step 4: [ 159, 158 ]
#I  Orbit lengths after extension step 5: [ 472, 462 ]
#I  Orbit lengths after extension step 6: [ 1356, 1309 ]
#I  Orbit lengths after extension step 7: [ 3822, 3643 ]
#I  |Candidates| = 11
[ g^3*h*g^3*h^5*g, g^-3*h^-4*g^-3*h^-1*g*h*g, g^3*h*g^-4*h^5*g ]
gap> RCWAInfo(0);
gap> List(pre,Length);
[ 13, 14, 14 ]
gap> Set(List(pre,w->w^phi)) = [f];
true
gap> w := pre[1]*pre[2]^-1;
g^3*h*g^3*h^4*g^-1*h*g^3*h^4*g^3
gap> Length(w);
23
gap> w^phi; # A relation of length 23.
IdentityMapping( Integers )
```

3.5 The action of an rcwa group on the underlying ring R

The support (set of moved points) of an rcwa group can be determined by `MovedPoints`.

In some cases – in particular in the case that the group in question is tame – testing for transitivity on the underlying ring is feasible. Furthermore it is often possible to determine group elements which map a given tuple of elements of the underlying ring to a given other tuple, if such elements exist.

3.5.1 IsTransitive (G, Integers)

◊ `IsTransitive( G, Integers )` (method)

Returns: `true` if the rcwa group `G` acts transitive on $\mathbb{Z}$ and `false` otherwise.

Depending on the particular group this might fail or run into an infinite loop, but for tame groups things should work.

Example

```
gap> G := Group(g,h);;
gap> RCWAInfo(3);
gap> IsTransitive(G,Integers);
#I  IsTransitive: testing for finiteness and searching short orbits ...
#I  IsTame: balancedness criterion.
#I  IsTame:`dead end' criterion.
#I  IsTame: loop criterion.
#I  IsTame:`finite order or integral power' criterion.

  [ ... ]

#I  Order: the 4th power of the argument is RcwaMapping(
[ [ 1, 12, 1 ], [ 1, 12, 1 ], [ 1, -6, 2 ], [ 2, -10, 1 ], [ 1, -7, 1 ],
  [ 2, -8, 1 ], [ 1, 12, 1 ], [ 1, 12, 1 ], [ 1, -10, 2 ], [ 2, -10, 1 ],
  [ 1, -7, 1 ], [ 2, -8, 1 ] ] );
There is a `class shift' on the residue class 0(12).
#I  Trying probabilistic random walk, initial m = 12
#I  checking modulus ...
#I  Size: use action on respected partition.
#I  KernelOfActionOnRespectedPartition: gen. #1, lng = 1
#I  KernelOfActionOnRespectedPartition: gen. #2, lng = 2

                      [ ... ]

#I  KernelOfActionOnRespectedPartition: gen. #14, lng = 10
#I  Searching for class shifts ...
#I  ... in generators
#I  ... in commutators of the generators
#I  The cyclic group generated by RcwaMapping(
[ [ 1, -9, 1 ], [ 1, 0, 1 ], [ 1, 6, 1 ], [ 1, -3, 1 ], [ 1, 0, 1 ],
  [ 1, 6, 1 ] ] ) acts transitively on the residue class 2(6).
#I  OrbitUnion: initial set = ResidueClassUnion( Integers, 6, [ 2 ] )
#I  Image = Integers
true
gap> RCWAInfo(0);
```

3.5.2 RepresentativeAction (G, src, dest, act)

◊ `RepresentativeAction( G, src, dest, act )` (method)

Returns: An element of `G` which maps `src` to `dest` under the action given by `act`.

If an element satisfying this condition does not exist, this method either returns `fail` or runs into an infinite loop. The problem of whether `src` and `dest` lie in the same orbit under the action of `G` in general seems to be hard. The given method is based on `RepresentativeActionPreImage` (3.5.3), and it basically just computes an image under an homomorphism. As this involves multiplications of rcwa mappings, this can be quite expensive if the group `G` is wild, the preimage is a rather long word and coefficient explosion happens to occur.

Example

```
gap> G := Group(a,b);
<rcwa group over Z with 2 generators>
gap> elm := RepresentativeAction(G,[7,4,9],[4,5,13],OnTuples);
<bijective rcwa mapping of Z with modulus 12>
gap> Display(elm);

Bijective rcwa mapping of Z with modulus 12

               n mod 12                |                 n^f
---------------------------------------+--------------------------------------
   0  2  3  6  8 11                    | n
   1  7 10                             | n - 3
   4                                   | n + 1
   5  9                                | n + 4

gap> List([7,4,9],n->n^elm);
[ 4, 5, 13 ]
gap> elm := RepresentativeAction(G,[5,4,9],[13,5,4],OnTuples);
<bijective rcwa mapping of Z with modulus 9>
gap> Display(elm);

Bijective rcwa mapping of Z with modulus 9

                n mod 9                |                 n^f
---------------------------------------+--------------------------------------
  0                                    | 4n/9
  1                                    | (8n - 26)/9
  2                                    | (8n + 2)/9
  3                                    | (8n + 3)/9
  4                                    | (16n - 19)/9
  5                                    | (16n + 37)/9
  6                                    | (8n + 33)/9
  7                                    | (16n - 49)/9
  8                                    | (16n + 7)/9

gap> List([5,4,9],n->n^elm);
[ 13, 5, 4 ]
gap> RepresentativeAction(G,[7,4,9],[4,5,8],OnTuples);
<bijective rcwa mapping of Z with modulus 256>
```

3.5.3 RepresentativeActionPreImage (G, src, dest, act, F)

◊ `RepresentativeActionPreImage( G, src, dest, act, F )` (operation)

Returns: The result of `RepresentativeAction( G, src, dest, act )` as a word in the generators.

The argument `F` is a free group to be used to express the resulting word. Note that the dependency is just the other way round than suggested above (`RepresentativeAction` calls `RepresentativeActionPreImage`) and that the evaluation of the word sometimes takes much more time than its determination (the latter however depends very much on the particular case and is hard to predict). This causes `RepresentativeActionPreImage` sometimes to be much faster than `RepresentativeAction`. The employed algorithm is not inefficient, as the last two of the examples below suggest – it is based on separate progressive computations of the orbits of `src` and `dest` until they intersect non-trivially. It avoids multiplying rcwa mappings. Of course the other warnings given in the description of `RepresentativeAction` (3.5.2) apply to this operation, too.

Example

```
gap> F := FreeGroup("a","b");;
gap> w := RepresentativeActionPreImage(G,[7,4,9],[4,5,13],OnPoints,F);
b^-1*a*b*a^-1
gap> w := RepresentativeActionPreImage(G,[5,4,9],[13,5,4],OnTuples,F);
b^-1*a^-1*b*a^-1
gap> w := RepresentativeActionPreImage(G,[7,4,9],[4,5,8],OnPoints,F);
b^2*a^2
gap> phi := GroupHomomorphismByImages(F,G,[F.1,F.2],[a,b]);
[ a, b ] -> [ a, b ]
gap> w^phi;
<bijective rcwa mapping of Z with modulus 256>
gap> w^phi = RepresentativeAction(G,[7,4,9],[4,5,8],OnPoints);
true
gap> List([7,4,9],n->n^(w^phi));
[ 4, 5, 8 ]
gap> w := RepresentativeActionPreImage(G,[37,4,9],[4,51,8],OnPoints,F);
a^-1*b^-1*a*b^4*a
gap> w^phi;
<bijective rcwa mapping of Z with modulus 4608>
gap> w := RepresentativeActionPreImage(G,[37,4,9],[4,51,8],OnTuples,F);
b*a^6*b*a^-3*b^-3*a^-1*b*a^2
gap> w := RepresentativeActionPreImage(G,[17,14,9],[4,51,8],OnPoints,F);
a^-1*b^-1*a^3*b^2*a*b*a*b^-1*a^2
```

3.5.4 RepresentativeAction (RCWA(Integers), P1, P2)

◊ `RepresentativeAction( RCWA_Z, P1, P2 )` (method)

Returns: An element of RCWA($\mathbb{Z}$) which maps the partition `P1` to `P2`.

The arguments `P1` and `P2` must be partitions of the underlying ring R into the same number of disjoint residue classes. It recognizes the option `IsTame` – if this option is set, the returned mapping is tame provided that there is a tame mapping satisfying the required condition.

Example

```
gap> rc := function(r,m) return ResidueClass(DefaultRing(m),m,r); end;;
gap> P1 := [rc(0,3),rc(1,3),rc(2,9),rc(5,9),rc(8,9)];
[ 0(3), 1(3), 2(9), 5(9), 8(9) ]
gap> P2 := [rc(0,2),rc(1,8),rc(5,16),rc(3,4),rc(13,16)];
[ 0(2), 1(8), 5(16), 3(4), 13(16) ]
gap> elm := RepresentativeAction(RCWA(Integers),P1,P2);
<rcwa mapping of Z with modulus 9>
gap> Display(elm);

Rcwa mapping of Z with modulus 9

                n mod 9                 |                n^f
---------------------------------------+---------------------------------------
  0 3 6                                 | 2n/3
  1 4 7                                 | (8n - 5)/3
  2                                     | (16n + 13)/9
  5                                     | (4n + 7)/9
  8                                     | (16n - 11)/9

gap> P1^elm = P2;
true
gap> elm := RepresentativeAction(RCWA(Integers),P1,P2:IsTame);
<tame rcwa mapping of Z with modulus 1152>
gap> P := RespectedPartition(elm);;
gap> Length(P);
313
```

3.5.5 ShortOrbits (G, S, maxlng)

◊ `ShortOrbits( G, S, maxlng )` (operation)

Returns: A list of all finite orbits of the rcwa group `G` of maximal length `maxlng`, which intersect non-trivially with the set `S`.

Example

```
gap> A5 := IntegralRcwaGroupByPermGroup(AlternatingGroup(5));;
gap> ShortOrbits(A5,[-10..10],100);
[ [ -14, -13, -12, -11, -10 ], [ -9, -8, -7, -6, -5 ], [ -4, -3, -2, -1, 0 ],
  [ 1, 2, 3, 4, 5 ], [ 6, 7, 8, 9, 10 ] ]
gap> Action(A5,last[2]);
Group([ (1,2,3,4,5), (3,4,5) ])
gap> G := Group(Comm(a,b),Comm(a,c));;
gap> orb := ShortOrbits(G,[-15..15],100);
[ [ -15, -12, -7, -6, -5, -4, -3, -2, -1, 1 ],
  [ -33, -30, -24, -21, -16, -14, -13, -11, -10, -8 ], [ -9 ], [ 0 ],
  [ 2, 3, 4, 5, 6, 7, 8, 10, 12, 15 ], [ 9 ],
  [ 11, 13, 14, 16, 17, 19, 21, 24, 30, 33 ] ]
```

Example

```
gap> Action(G,orb[1]);
Group([ (2,5,8,10,7,6), (1,3,6,9,4,5) ])
gap> ShortOrbits(Group(u),[-30..30],100);
[ [ -13, -8, -7, -5, -4, -3, -2 ], [ -10, -6 ], [ -1 ], [ 0 ], [ 1, 2 ],
  [ 3, 5 ], [ 24, 36, 39, 40, 44, 48, 60, 65, 67, 71, 80, 86, 93, 100, 112,
      128, 138, 155, 167, 187, 230, 248, 312, 446, 520, 803, 867, 1445 ] ]
```

3.5.6 OrbitsModulo (G, m)

◊ `OrbitsModulo( G, m )` (method)

Returns: A partition of `[0..m-1]`, such that i and j lie in the same subset if and only if there is an element g of `G` which moves an element from the residue class $i(m)$ to the residue class $j(m)$.

The argument `G` must be an rcwa group over $\mathbb{Z}$. See also `OrbitsModulo` (2.8.2) for rcwa mappings.

Example

```
gap> OrbitsModulo(G,36);
[ [ 0 ], [ 1, 11, 13, 14, 16, 17, 19, 21, 24, 29, 30, 31, 32, 33, 34, 35 ],
  [ 2, 3, 4, 5, 6, 7, 8, 10, 12, 15, 20, 22, 23, 25, 26, 28 ], [ 9 ], [ 18 ],
  [ 27 ] ]
```

3.6 Conjugacy in RCWA(R)

3.6.1 IsConjugate (RCWA(Integers), f, g)

◊ `IsConjugate( RCWA_Z, f, g )` (method)

Returns: `true` if the bijective rcwa mappings `f` and `g` are conjugate in RCWA($\mathbb{Z}$), and `false` otherwise.

This may fail or run into an infinite loop. In particular the support for wild rcwa mappings is currently very poor, since the author does not know a way to solve the conjugacy problem for these. Some easy cases are handled anyway.

Example

```
gap> IsConjugate(RCWA(Integers),g,h);
false
gap> IsConjugate(RCWA(Integers),g,g^a);
true
gap> IsConjugate(RCWA(Integers),a,b);
false
```

3.6.2 RepresentativeAction (RCWA(Integers), f, g)

◊ `RepresentativeAction( RCWA_Z, f, g )` (method)

Returns: An rcwa mapping `x` such that `f^x = g`, if such an `x` exists and `fail` otherwise.

This method currently works only for tame rcwa mappings of $\mathbb{Z}$, since the author does not know a way to solve the conjugacy problem for wild rcwa mappings.

Example

```
gap> elm := RepresentativeAction(RCWA(Integers),h,h^g);
<bijective rcwa mapping of Z with modulus 24>
gap> h^elm = h^g; # check ...
true
gap> Order(elm);
10
gap> cent := g*elm^-1;
<bijective rcwa mapping of Z with modulus 24>
gap> Comm(cent,h); # cent must lie in the centralizer of h in RCWA(Z)
IdentityMapping( Integers )
gap> Order(cent);
12
gap> Display(cent);

Bijective rcwa mapping of Z with modulus 24, of order 12

               n mod 24                |                n^f
---------------------------------------+--------------------------------------
   0  4 10 12 16 22                    | n - 1
   1 13                                | 2n
   2                                   | (n - 2)/2
   3  9 15 21                          | 2n + 2
   5  6  7 14 17 18 19                 | n
   8 20                                | n/2
  11 23                                | n + 2
```

3.6.3 ShortCycles (f, maxlng)

◊ `ShortCycles( f, maxlng )` (operation)

Returns: All “single” finite cycles of the rcwa mapping `f` of length at most `maxlng`.

In this context, “single” finite cycles are finite cycles not belonging to an infinite series, i.e. there is no constant m such that adding any multiple of m to the elements of the cycle always yields a new cycle.

Since GAP-permutations cannot move negative integers, rationals or even polynomials, the cycles are returned as lists. For example, the list `[-3,1,2,-2]` denotes the cycle `(-3,1,2,-2)`. Mappings having different sets of finite cycle lengths are obviously not conjugate in RCWA(R).

Example

```
gap> ShortCycles(a,5);
[ [ 0 ], [ 1 ], [ -1 ], [ 2, 3 ], [ -3, -2 ], [ 4, 6, 9, 7, 5 ],
  [ -9, -7, -5, -4, -6 ] ]
gap> ShortCycles(u,2);
[ [ 0 ], [ -1 ], [ 1, 2 ], [ 3, 5 ], [ -10, -6 ] ]
gap> ShortCycles(Comm(a,b),10);
[ ]
gap> ShortCycles(a*b,2);
[ [ 0 ], [ 2 ], [ 3 ], [ -26 ], [ 7 ], [ -3 ], [ -1 ] ]
gap> v := RcwaMapping([[-1,2,1],[1,-1,1],[1,-1,1]]);;
gap> w := RcwaMapping([[-1,3,1],[1,-1,1],[1,-1,1],[1,-1,1]]);;
gap> List( [ v, w ], Order );
[ 6, 8 ]
gap> [ ShortCycles(v,10), ShortCycles(w,10) ];
[ [ [ 0, 2, 1 ] ], [ [ 0, 3, 2, 1 ] ] ]
```

3.6.4 NrConjugacyClassesOfRCWAZOfOrder (ord)

◊ `NrConjugacyClassesOfRCWAZOfOrder( ord )` (function)

Returns: The number of conjugacy classes of RCWA($\mathbb{Z}$) of elements of order `ord`.

Example

```
gap> NrConjugacyClassesOfRCWAZOfOrder(2);
infinity
gap> NrConjugacyClassesOfRCWAZOfOrder(105);
218
```

3.7 Restriction monomorphisms

3.7.1 Restriction (g, f)

◊ `Restriction( g, f )` (operation)

Returns: The *restriction* of `g` by `f`.

By definition, the restriction g_f of `g` by `f` is the uniquely-determined rcwa mapping satisfying $f \cdot g_f = g \cdot f$ and pointwisely fixing the complement of the image of `f`. The mapping `f` has to be injective. If `f` is bijective, the returned mapping is just the conjugate of `g` by `f`. See also `Restriction` (3.7.2) for rcwa groups.

Example

```
gap> Comm(Restriction(a,RcwaMapping([[2,0,1]])),
>         Restriction(u,RcwaMapping([[2,1,1]])));
IdentityMapping( Integers )
```

3.7.2 Restriction (G, f)

◊ `Restriction( G, f )` (operation)

Returns: The restriction of `G` by `f`.

The mapping `f` has to be injective. The returned group acts on the image of `f` and fixes its complement pointwise. If `f` is bijective the returned group is just the conjugate of `G` by `f`.

The elements of the restricted group are the restrictions of the elements of `G` by `f`. For a definition see `Restriction` (3.7.1) for rcwa mappings.

Example

```
gap> G := Restriction(Group(a,b),RcwaMapping([[5,3,1]]));
<rcwa group over Z with 2 generators>
gap> MovedPoints(G);
3(5) \ [ -2, 3 ]
```

Restriction monomorphisms allow to form direct products of any rcwa groups (regardless of whether they are tame or not):

3.7.3 DirectProduct (G1, G2, ...)

◊ `DirectProduct( G1, G2, ... )` (operation)

Returns: The direct product of the rcwa groups over $\mathbb{Z}$ given as arguments.

As there is no unique or canonical way to embed a direct product of rcwa groups into RCWA($\mathbb{Z}$), this method may choose any such embedding.

Example

```
gap> G := DirectProduct(Group(g,h),Group(a,b),Group(u));;
gap> Embedding(G,1);
[ g, h ] -> [ <bijective rcwa mapping of Z with modulus 18, of order 7>,
  <bijective rcwa mapping of Z with modulus 18, of order 12> ]
gap> Projection(G,2);
[ <bijective rcwa mapping of Z with modulus 18, of order 7>,
  <bijective rcwa mapping of Z with modulus 18, of order 12>,
  <wild bijective rcwa mapping of Z with modulus 12>,
  <wild bijective rcwa mapping of Z with modulus 12>,
  <bijective rcwa mapping of Z with modulus 15> ] ->
[ IdentityMapping( Integers ), IdentityMapping( Integers ), a, b,
  IdentityMapping( Integers ) ]
gap> List([1..3],i->MovedPoints(Image(Embedding(G,i))));
[ 0(3), 1(3) \ [ -2, 1 ], 2(3) \ [ -1, 2 ] ]
```

3.8 Special attributes for tame rcwa groups

There is a couple of attributes which a priori make only sense for tame rcwa groups. In the sequel, these attributes are described in detail.

With their help, various structural information about a given tame rcwa group can be obtained. For example, there are methods for `IsSolvable` and `IsPerfect` available for tame rcwa groups (the latter works in some cases by other means also for wild groups). Often it is also feasible to compute the derived subgroup of a tame rcwa group.

3.8.1 RespectedPartition (G)

◊ `RespectedPartition( G )` (attribute)
◊ `RespectedPartition( sigma )` (attribute)

Returns: A *respected partition* of `G` resp. `sigma`.

This is a partition of the base ring R into a finite number of residue classes which the group `G` resp. the permutation `sigma` acts on, and on those elements all elements of `G` resp. the group generated by `sigma` are affine.

In the author's forthcoming PhD thesis, it will be proved that such a partition exists if and only if `G` resp. `sigma` is tame.

Example

```
gap> G := Group(g,h);;
gap> Size(G);
infinity
gap> P := RespectedPartition(G);
[ 0(12), 1(12), 3(12), 4(12), 5(12), 6(12), 7(12), 9(12), 10(12), 11(12),
  2(24), 8(24), 14(24), 20(24) ]
gap> Permutation(g,P);
(1,11,2,5,3,12,4)(6,13,7,10,8,14,9)
```

3.8.2 ActionOnRespectedPartition (G)

◊ `ActionOnRespectedPartition( G )` (attribute)

Returns: The action of the tame rcwa group `G` on `RespectedPartition(G)`.

Example

```
gap> H := ActionOnRespectedPartition(G);
Group([ (1,11,2,5,3,12,4)(6,13,7,10,8,14,9), (1,11,2,10)(3,12,4)(5,6,13,7)(8,
    14,9) ])
gap> H = Action(G,P);
true
gap> [ Size(H), Size(DerivedSubgroup(H)), IsPerfect(DerivedSubgroup(H)) ];
[ 322560, 161280, true ]
```

3.8.3 IntegralConjugate (G)

◊ `IntegralConjugate( G )` (attribute)
◊ `IntegralConjugate( f )` (attribute)

Returns: Some integral conjugate of the tame rcwa group `G` resp. of the tame bijective rcwa mapping `f` in the group RCWA($\mathbb{Z}$).

In the author's forthcoming PhD thesis, it will be proved that such a representative exists. Since the result is not defined uniquely, methods for this operation will just choose one such mapping.

Example

```
gap> Print(LaTeXObj(IntegralConjugate(g)));
n \ \longmapsto \
\begin{cases}
  n + 10 & \text{if} \ n \in 0(14), \\
  n + 3  & \text{if} \ n \in 1(14) \cup 6(14), \\
  n + 9  & \text{if} \ n \in 2(14), \\
  n - 3  & \text{if} \ n \in 3(14) \cup 8(14), \\
  n - 2  & \text{if} \ n \in 4(14) \cup 9(14), \\
  n + 7  & \text{if} \ n \in 5(14), \\
  n + 6  & \text{if} \ n \in 7(14), \\
  n - 9  & \text{if} \ n \in 10(14), \\
  n - 8  & \text{if} \ n \in 11(14), \\
  n - 6  & \text{if} \ n \in 12(14), \\
  n - 5  & \text{if} \ n \in 13(14).
\end{cases}
```

3.8.4 IntegralizingConjugator (G)

◊ `IntegralizingConjugator( G )` (attribute)
◊ `IntegralizingConjugator( f )` (attribute)

Returns: An rcwa mapping mapping `x` such that `G^x` resp. `f^x` is integral.

This is of course not defined uniquely, but it holds that `G^IntegralizingConjugator(G) = IntegralConjugate(G)` resp. `f^IntegralizingConjugator(f) = IntegralConjugate(f)`.

Example

```
gap> Print(LaTeXObj(IntegralizingConjugator(g)));
n \ \longmapsto \
\begin{cases}
  \frac{7n}{6}        & \text{if} \ n \in 0(12), \\
  \frac{7n - 1}{6}    & \text{if} \ n \in 1(12), \\
  \frac{7n + 106}{12} & \text{if} \ n \in 2(24), \\
  \frac{7n - 9}{6}    & \text{if} \ n \in 3(12), \\
  \frac{7n - 10}{6}   & \text{if} \ n \in 4(12), \\
  \frac{7n - 11}{6}   & \text{if} \ n \in 5(12), \\
  \frac{7n - 12}{6}   & \text{if} \ n \in 6(12), \\
  \frac{7n - 13}{6}   & \text{if} \ n \in 7(12), \\
  \frac{7n + 76}{12}  & \text{if} \ n \in 8(24), \\
  \frac{7n - 21}{6}   & \text{if} \ n \in 9(12), \\
  \frac{7n - 22}{6}   & \text{if} \ n \in 10(12), \\
  \frac{7n - 23}{6}   & \text{if} \ n \in 11(12), \\
```

```
  \frac{7n + 46}{12}  & \text{if} \ n \in 14(24), \\
  \frac{7n + 16}{12}  & \text{if} \ n \in 20(24).
\end{cases}
```

3.9 The categories and families of rcwa groups

3.9.1 IsRcwaGroup (G)

◊ `IsRcwaGroup( G )` (filter)
◊ `IsIntegralRcwaGroup( G )` (filter)
◊ `IsSemilocalIntegralRcwaGroup( G )` (filter)
◊ `IsModularRcwaGroup( G )` (filter)

Returns: `true` if `G` is an rcwa group resp. an rcwa group over the ring of integers resp. an rcwa group over a semilocalization of the ring of integers resp. an rcwa group over a polynomial ring in one variable over a finite field, and `false` otherwise.

3.9.2 IntegralRcwaGroupsFamily

◊ `IntegralRcwaGroupsFamily` (family)

The family of all rcwa groups over the ring of integers.

Chapter 4

Examples

This chapter lists some "nice" examples of rcwa mappings and -groups. The rcwa mappings mentioned in this chapter can be found in the file `pkg/rcwa/examples/examples.g`, so there is no need to extract them from the manual files. This file can be read into the current GAP session by issueing `RCWAReadExamples( );`.

4.1 Factoring Collatz' permutation of the integers

In 1932, Lothar Collatz mentioned in his notebook the following permutation of the integers:

Example

```
gap> Collatz := RcwaMapping([[2,0,3],[4,-1,3],[4,1,3]]);;
gap> SetName(Collatz,"Collatz");
gap> Display(Collatz);

Rcwa mapping of Z with modulus 3

                n mod 3                |              n^Collatz
---------------------------------------+--------------------------------------
  0                                    | 2n/3
  1                                    | (4n - 1)/3
  2                                    | (4n + 1)/3
```

This permutation has a few finite cycles, but its cycle structure has not been determined yet. In particular it is not known whether the cycle containing 8 is finite or infinite.

Example

```
gap> List(ShortOrbits(Group(Collatz),[-100..100],100),
>          orb->Cycle(Collatz,Minimum(orb)));
[ [ -111, -74, -99, -66, -44, -59, -79, -105, -70, -93, -62, -83 ],
  [ -9, -6, -4, -5, -7 ], [ -3, -2 ], [ -1 ], [ 0 ], [ 1 ], [ 2, 3 ],
  [ 4, 5, 7, 9, 6 ], [ 44, 59, 79, 105, 70, 93, 62, 83, 111, 74, 99, 66 ] ]
gap> List(last,Length);
[ 12, 5, 2, 1, 1, 1, 2, 5, 12 ]
```

Nevertheless, the factorization routine included in this package can determine a factorization of this permutation into involutions interchanging two disjoint residue classes, each (for reasons of saving a bit space in this manual, we factor the inverse mapping instead and revert the list afterwards):

Example

```
gap> RCWAInfo(2); # Switch Info output on.
gap> Reversed(FactorizationIntoGenerators(Collatz^-1:ExpandPrimeSwitches));
#I  Modulus(<g>) = 4, Multiplier(<g>) = 3, Divisor(<g>) = 4
#I  Dividing by PrimeSwitch(3) from the right.
#I  Modulus(<g>) = 16, Multiplier(<g>) = 3, Divisor(<g>) = 4
#I  Dividing by PrimeSwitch(3) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 6, Divisor(<g>) = 4
#I  Dividing by PrimeSwitch(3) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 12, Divisor(<g>) = 12
#I  p = 3, kmult = 1, kdiv = 1
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 3(6), 5(12), 7(12), 8(12), 0(48) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 6(8) ]
#I  Found 5 pairs.
#I  After filtering and splitting: 5 pairs.
#I  Dividing by ClassTransposition(3,6,6,8) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 12, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(6,8,5,12) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 12, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(6,8,7,12) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 12, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(6,8,8,12) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 12, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(6,8,0,48) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 6, Divisor(<g>) = 4
#I  Dividing by PrimeSwitch(3) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 12, Divisor(<g>) = 12
#I  p = 3, kmult = 1, kdiv = 1
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 7(12), 8(12), 0(96) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 6(8) ]
#I  Found 3 pairs.
#I  After filtering and splitting: 3 pairs.
#I  Dividing by ClassTransposition(6,8,7,12) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 12, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(6,8,8,12) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 12, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(6,8,0,96) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 12, Divisor(<g>) = 4
#I  Dividing by PrimeSwitch(3) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 24, Divisor(<g>) = 12
#I  p = 3, kmult = 1, kdiv = 1
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 7(12), 0(192) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 2(4) ]
```

```
#I  Found 2 pairs.
#I  After filtering and splitting: 2 pairs.
#I  Dividing by ClassTransposition(2,4,7,12) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 24, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(2,4,0,192) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 24, Divisor(<g>) = 4
#I  Dividing by PrimeSwitch(3) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 48, Divisor(<g>) = 12
#I  p = 3, kmult = 1, kdiv = 1
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 0(384) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 2(4) ]
#I  Found 1 pairs.
#I  After filtering and splitting: 1 pairs.
#I  Dividing by ClassTransposition(2,4,0,384) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 128, Divisor(<g>) = 4
#I  p = 2, kmult = 7, kdiv = 2
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 384(1536) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 2(4), 3(6), 1(12), 11(12) ]
#I  Found 3 pairs.
#I  After filtering and splitting: 5 pairs.
#I  Dividing by ClassTransposition(2,12,384,1536) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 32, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(1,12,384,1536) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 32, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(6,12,384,1536) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 32, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(10,12,384,1536) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 32, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(11,12,384,1536) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 32, Divisor(<g>) = 4
#I  p = 2, kmult = 5, kdiv = 2
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 0(384) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 3(6), 1(12), 6(12), 10(12), 11(12) ]
#I  Found 4 pairs.
#I  After filtering and splitting: 4 pairs.
#I  Dividing by ClassTransposition(1,12,0,384) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 16, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(6,12,0,384) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 16, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(10,12,0,384) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 16, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(11,12,0,384) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 16, Divisor(<g>) = 4
#I  p = 2, kmult = 4, kdiv = 2
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 48(192), 192(384) ]
#I  Image of classes being divided by q*p^kdiv:
```

```
#I  [ 3(6), 6(12), 10(12), 11(12) ]
#I  Found 3 pairs.
#I  After filtering and splitting: 3 pairs.
#I  Dividing by ClassTransposition(6,12,48,192) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 16, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(10,12,48,192) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 16, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(11,12,48,192) from the right.
#I  Modulus(<g>) = 48, Multiplier(<g>) = 16, Divisor(<g>) = 4
#I  p = 2, kmult = 4, kdiv = 2
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 192(384) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 3(6), 10(12), 11(12) ]
#I  Splitting classes being divided by q*p^kdiv.
#I  Found 4 pairs.
#I  After filtering and splitting: 4 pairs.
#I  Dividing by ClassTransposition(10,24,192,384) from the right.
#I  Modulus(<g>) = 96, Multiplier(<g>) = 8, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(11,24,192,384) from the right.
#I  Modulus(<g>) = 96, Multiplier(<g>) = 8, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(22,24,192,384) from the right.
#I  Modulus(<g>) = 96, Multiplier(<g>) = 8, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(23,24,192,384) from the right.
#I  Modulus(<g>) = 96, Multiplier(<g>) = 8, Divisor(<g>) = 4
#I  p = 2, kmult = 3, kdiv = 2
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 72(96), 96(192), 0(384) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 3(6), 11(12), 22(24) ]
#I  Found 2 pairs.
#I  After filtering and splitting: 2 pairs.
#I  Dividing by ClassTransposition(11,12,72,96) from the right.
#I  Dividing by ClassTransposition(22,24,96,192) from the right.
#I  Modulus(<g>) = 96, Multiplier(<g>) = 8, Divisor(<g>) = 4
#I  p = 2, kmult = 3, kdiv = 2
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 0(384) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 3(6) ]
#I  Splitting classes being divided by q*p^kdiv.
#I  Splitting classes being divided by q*p^kdiv.
#I  Splitting classes being divided by q*p^kdiv.
#I  Found 8 pairs.
#I  After filtering and splitting: 8 pairs.
#I  Dividing by ClassTransposition(3,48,0,384) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 4, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(9,48,0,384) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 4, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(15,48,0,384) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 4, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(21,48,0,384) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 4, Divisor(<g>) = 4
```

```
#I  Dividing by ClassTransposition(27,48,0,384) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 4, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(33,48,0,384) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 4, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(39,48,0,384) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 4, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(45,48,0,384) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 4, Divisor(<g>) = 4
#I  p = 2, kmult = 2, kdiv = 2
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 44(48), 48(96), 192(384) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 9(12), 15(24), 27(48) ]
#I  Found 2 pairs.
#I  After filtering and splitting: 2 pairs.
#I  Dividing by ClassTransposition(9,12,44,48) from the right.
#I  Dividing by ClassTransposition(15,24,48,96) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 4, Divisor(<g>) = 4
#I  p = 2, kmult = 2, kdiv = 2
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 192(384) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 27(48) ]
#I  Splitting classes being divided by q*p^kdiv.
#I  Found 2 pairs.
#I  After filtering and splitting: 2 pairs.
#I  Dividing by ClassTransposition(27,96,192,384) from the right.
#I  Modulus(<g>) = 384, Multiplier(<g>) = 2, Divisor(<g>) = 4
#I  Dividing by ClassTransposition(75,96,192,384) from the right.
#I  Modulus(<g>) = 384, Multiplier(<g>) = 2, Divisor(<g>) = 4
#I  p = 2, kmult = 1, kdiv = 2
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 4(24), 17(24), 24(48), 96(192), 0(384) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 75(96) ]
#I  Found 1 pairs.
#I  After filtering and splitting: 1 pairs.
#I  Dividing by ClassTransposition(75,96,0,384) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 2, Divisor(<g>) = 2
#I  p = 2, kmult = 1, kdiv = 1
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 4(24), 17(24), 24(48), 96(192) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 7(12), 5(24), 12(24), 16(24), 75(96) ]
#I  Found 6 pairs.
#I  After filtering and splitting: 6 pairs.
#I  Dividing by ClassTransposition(7,12,4,24) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 2, Divisor(<g>) = 2
#I  Dividing by ClassTransposition(7,12,17,24) from the right.
#I  Dividing by ClassTransposition(5,24,24,48) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 2, Divisor(<g>) = 2
#I  Dividing by ClassTransposition(12,24,24,48) from the right.
#I  Modulus(<g>) = 192, Multiplier(<g>) = 2, Divisor(<g>) = 2
```

```
#I  Dividing by ClassTransposition(16,24,24,48) from the right.
#I  Dividing by ClassTransposition(75,96,96,192) from the right.
#I  Modulus(<g>) = 96, Multiplier(<g>) = 2, Divisor(<g>) = 2
#I  p = 2, kmult = 1, kdiv = 1
#I  Image of classes being multiplied by q*p^kmult:
#I  [ 17(24) ]
#I  Image of classes being divided by q*p^kdiv:
#I  [ 12(24), 16(24) ]
#I  Splitting classes being multiplied by q*p^kmult.
#I  Found 4 pairs.
#I  After filtering and splitting: 4 pairs.
#I  Dividing by ClassTransposition(12,24,17,48) from the right.
#I  Modulus(<g>) = 96, Multiplier(<g>) = 2, Divisor(<g>) = 2
#I  Dividing by ClassTransposition(16,24,17,48) from the right.
#I  Dividing by ClassTransposition(12,24,41,48) from the right.
#I  Modulus(<g>) = 96, Multiplier(<g>) = 2, Divisor(<g>) = 2
#I  Dividing by ClassTransposition(16,24,41,48) from the right.
#I  Modulus(<g>) = 96, Multiplier(<g>) = 1, Divisor(<g>) = 1
#I  Determining largest sources of affine mappings.
#I  Computing respected partition.
#I  Computing induced permutation on respected partition
[ 12(24), 14(24), 18(24), 13(48), 22(48), 23(48), 28(48), 31(48), 32(48),
  33(48), 34(48), 35(48), 39(48), 44(48), 45(48), 0(96), 1(96), 2(96), 3(96),
  4(96), 5(96), 6(96), 7(96), 8(96), 9(96), 10(96), 11(96), 15(96), 16(96),
  17(96), 19(96), 20(96), 21(96), 24(96), 25(96), 26(96), 27(96), 29(96),
  30(96), 37(96), 40(96), 41(96), 43(96), 46(96), 47(96), 48(96), 49(96),
  50(96), 51(96), 52(96), 53(96), 54(96), 55(96), 56(96), 57(96), 58(96),
  59(96), 63(96), 64(96), 65(96), 67(96), 68(96), 69(96), 72(96), 73(96),
  74(96), 75(96), 77(96), 78(96), 85(96), 88(96), 89(96), 91(96), 94(96),
  95(96) ].
#I  Factoring the rest into class shifts.
#I  Checking the result.
[ ClassTransposition(4,6,7,12), ClassTransposition(2,6,1,12),
  ClassTransposition(2,4,5,6), ClassTransposition(0,4,1,6),
  ClassTransposition(5,6,4,8), ClassTransposition(1,6,0,8),
  ClassTransposition(4,6,7,12), ClassTransposition(2,6,1,12),
  ClassTransposition(2,4,5,6), ClassTransposition(0,4,1,6),
  ClassTransposition(5,6,4,8), ClassTransposition(1,6,0,8),
  ClassTransposition(4,6,7,12), ClassTransposition(2,6,1,12),
  ClassTransposition(2,4,5,6), ClassTransposition(0,4,1,6),
  ClassTransposition(5,6,4,8), ClassTransposition(1,6,0,8),
  ClassTransposition(3,6,6,8), ClassTransposition(6,8,5,12),
  ClassTransposition(6,8,7,12), ClassTransposition(6,8,8,12),
  ClassTransposition(6,8,0,48), ClassTransposition(4,6,7,12),
  ClassTransposition(2,6,1,12), ClassTransposition(2,4,5,6),
  ClassTransposition(0,4,1,6), ClassTransposition(5,6,4,8),
  ClassTransposition(1,6,0,8), ClassTransposition(6,8,7,12),
  ClassTransposition(6,8,8,12), ClassTransposition(6,8,0,96),
  ClassTransposition(4,6,7,12), ClassTransposition(2,6,1,12),
  ClassTransposition(2,4,5,6), ClassTransposition(0,4,1,6),
  ClassTransposition(5,6,4,8), ClassTransposition(1,6,0,8),
  ClassTransposition(2,4,7,12), ClassTransposition(2,4,0,192),
  ClassTransposition(4,6,7,12), ClassTransposition(2,6,1,12),
```

```
ClassTransposition(2,4,5,6), ClassTransposition(0,4,1,6),
ClassTransposition(5,6,4,8), ClassTransposition(1,6,0,8),
ClassTransposition(2,4,0,384), ClassTransposition(2,12,384,1536),
ClassTransposition(1,12,384,1536), ClassTransposition(6,12,384,1536),
ClassTransposition(10,12,384,1536), ClassTransposition(11,12,384,1536),
ClassTransposition(1,12,0,384), ClassTransposition(6,12,0,384),
ClassTransposition(10,12,0,384), ClassTransposition(11,12,0,384),
ClassTransposition(6,12,48,192), ClassTransposition(10,12,48,192),
ClassTransposition(11,12,48,192), ClassTransposition(10,24,192,384),
ClassTransposition(11,24,192,384), ClassTransposition(22,24,192,384),
ClassTransposition(23,24,192,384), ClassTransposition(11,12,72,96),
ClassTransposition(22,24,96,192), ClassTransposition(3,48,0,384),
ClassTransposition(9,48,0,384), ClassTransposition(15,48,0,384),
ClassTransposition(21,48,0,384), ClassTransposition(27,48,0,384),
ClassTransposition(33,48,0,384), ClassTransposition(39,48,0,384),
ClassTransposition(45,48,0,384), ClassTransposition(9,12,44,48),
ClassTransposition(15,24,48,96), ClassTransposition(27,96,192,384),
ClassTransposition(75,96,192,384), ClassTransposition(75,96,0,384),
ClassTransposition(7,12,4,24), ClassTransposition(7,12,17,24),
ClassTransposition(5,24,24,48), ClassTransposition(12,24,24,48),
ClassTransposition(16,24,24,48), ClassTransposition(75,96,96,192),
ClassTransposition(12,24,17,48), ClassTransposition(16,24,17,48),
ClassTransposition(12,24,41,48), ClassTransposition(16,24,41,48),
ClassTransposition(3,96,43,96), ClassTransposition(3,96,40,96),
ClassTransposition(3,96,26,96), ClassTransposition(3,96,30,96),
ClassTransposition(3,96,24,96), ClassTransposition(3,96,25,96),
ClassTransposition(3,96,29,96), ClassTransposition(3,96,11,96),
ClassTransposition(3,96,10,96), ClassTransposition(3,96,15,96),
ClassTransposition(3,96,27,96), ClassTransposition(3,96,51,96),
ClassTransposition(3,96,91,96), ClassTransposition(3,96,88,96),
ClassTransposition(3,96,74,96), ClassTransposition(3,96,78,96),
ClassTransposition(3,96,72,96), ClassTransposition(3,96,73,96),
ClassTransposition(3,96,77,96), ClassTransposition(3,96,59,96),
ClassTransposition(3,96,58,96), ClassTransposition(3,96,63,96),
ClassTransposition(3,96,75,96), ClassTransposition(0,96,95,96),
ClassTransposition(0,96,94,96), ClassTransposition(0,96,85,96),
ClassTransposition(0,96,89,96), ClassTransposition(0,96,49,96),
ClassTransposition(0,96,53,96), ClassTransposition(0,96,57,96),
ClassTransposition(0,96,56,96), ClassTransposition(0,96,55,96),
ClassTransposition(0,96,52,96), ClassTransposition(0,96,69,96),
ClassTransposition(0,96,68,96), ClassTransposition(0,96,65,96),
ClassTransposition(0,96,67,96), ClassTransposition(0,96,64,96),
ClassTransposition(0,96,50,96), ClassTransposition(0,96,54,96),
ClassTransposition(0,96,48,96), ClassTransposition(0,96,47,96),
ClassTransposition(0,96,46,96), ClassTransposition(0,96,37,96),
ClassTransposition(0,96,41,96), ClassTransposition(0,96,1,96),
ClassTransposition(0,96,5,96), ClassTransposition(0,96,9,96),
ClassTransposition(0,96,8,96), ClassTransposition(0,96,7,96),
ClassTransposition(0,96,4,96), ClassTransposition(0,96,21,96),
ClassTransposition(0,96,20,96), ClassTransposition(0,96,17,96),
ClassTransposition(0,96,19,96), ClassTransposition(0,96,16,96),
ClassTransposition(0,96,2,96), ClassTransposition(0,96,6,96),
ClassTransposition(13,48,35,48), ClassTransposition(13,48,34,48),
```

```
  ClassTransposition(13,48,39,48), ClassTransposition(13,48,33,48),
  ClassTransposition(13,48,32,48), ClassTransposition(13,48,31,48),
  ClassTransposition(13,48,28,48), ClassTransposition(13,48,45,48),
  ClassTransposition(13,48,44,48), ClassTransposition(13,48,23,48),
  ClassTransposition(13,48,22,48), ClassTransposition(12,24,14,24),
  ClassTransposition(12,24,18,24) ]
gap> Product(last) = Collatz; # Check the result.
true
gap> Length(last2);
159
gap> RCWAInfo(0); # Switch Info output off again.
```

See the end of Section 4.6 for a much smaller factorization task performed “manually” for purposes of illustration.

4.2 An rcwa mapping which seems to be contracting, but very slow

The iterates of an integer under the Collatz mapping T seem to approach its contraction centre – this is the finite set where all trajectories end up after a finite number of steps – rather quickly and do not get very large before doing so (of course this is a purely heuristic statement as the Collatz conjecture has not been proved so far!):

Example

```
gap> T := RcwaMapping([[1,0,2],[3,1,2]]);;
gap> S0 := LikelyContractionCentre(T,100,1000);
#I  Warning: `LikelyContractionCentre' is highly probabilistic.
The returned result can only be regarded as a rough guess.
See ?LikelyContractionCentre for information on how to improve this guess.
[ -136, -91, -82, -68, -61, -55, -41, -37, -34, -25, -17, -10, -7, -5, -1, 0,
  1, 2 ]
gap> S0^T = S0; # This holds by definition of the contraction centre.
true
gap> Trajectory(T,27,S0,"stop");
[ 27, 41, 62, 31, 47, 71, 107, 161, 242, 121, 182, 91, 137, 206, 103, 155,
  233, 350, 175, 263, 395, 593, 890, 445, 668, 334, 167, 251, 377, 566, 283,
  425, 638, 319, 479, 719, 1079, 1619, 2429, 3644, 1822, 911, 1367, 2051,
  3077, 4616, 2308, 1154, 577, 866, 433, 650, 325, 488, 244, 122, 61, 92, 46,
  23, 35, 53, 80, 40, 20, 10, 5, 8, 4, 2 ]
gap> List([1..40],n->Length(Trajectory(T,n,S0,"stop")));
[ 1, 1, 5, 2, 4, 6, 11, 3, 13, 5, 10, 7, 7, 12, 12, 4, 9, 14, 14, 6, 6, 11,
  11, 8, 16, 8, 70, 13, 13, 13, 67, 5, 18, 10, 10, 15, 15, 15, 23, 7 ]
gap> Maximum(List([1..1000],n->Length(Trajectory(T,n,S0,"stop"))));
113
gap> Maximum(List([1..1000],n->Maximum(Trajectory(T,n,S0,"stop"))));
125252
```

The following mapping also seems to be contracting, but its trajectories are much longer:

Example

```
gap> f6 := RcwaMapping([[ 1,0,6],[ 5, 1,6],[ 7,-2,6],
>                       [11,3,6],[11,-2,6],[11,-1,6]]);;
gap> SetName(f6,"f6");
gap> Display(f6);

Integral rcwa mapping with modulus 6

                n mod 6                |               n^f6
---------------------------------------+--------------------------------------
  0                                    | n/6
  1                                    | (5n + 1)/6
  2                                    | (7n - 2)/6
  3                                    | (11n + 3)/6
  4                                    | (11n - 2)/6
  5                                    | (11n - 1)/6

gap> S0 := LikelyContractionCentre(f6,1000,100000);;
#I  Warning: `LikelyContractionCentre' is highly probabilistic.
The returned result can only be regarded as a rough guess.
gap> Trajectory(f6,25,S0,"stop");
[ 25, 21, 39, 72, 12, 2 ]
gap> List([1..100],n->Length(Trajectory(f6,n,S0,"stop")));
[ 2, 2, 3, 4, 2, 2, 3, 2, 2, 5, 7, 2, 8, 17, 3, 16, 2, 4, 17, 6, 5, 2, 5, 5,
  6, 2, 4, 2, 15, 2, 2, 3, 2, 5, 13, 3, 2, 3, 4, 2, 8, 4, 4, 2, 7, 19, 23517,
  3, 9, 3, 2, 18, 14, 2, 20, 23512, 14, 2, 6, 6, 2, 4, 19, 12, 23511, 8,
  23513, 10, 2, 13, 13, 3, 2, 23517, 7, 20, 7, 9, 9, 6, 12, 8, 6, 18, 14,
  23516, 31, 12, 23545, 4, 21, 19, 5, 2, 17, 17, 13, 19, 6, 23515 ]
gap> Maximum(Trajectory(f6,47,S0,"stop"));;
736339177776247330443187705477107581873369010805146980871580925673774229545698\
886054
```

Computing the trajectory of 3224 takes quite a while – this trajectory ascends to about $3 \cdot 10^{2197}$, before it approaches the fixed point 2 after 19949562 steps.

When constructing the mapping `f6`, the denominators of the partial mappings have been chosen to be equal and the numerators have been chosen to be numbers coprime to the common denominator, whose product is just a little bit smaller than the `Modulus(f6)`th power of the denominator. In the example we have $5 \cdot 7 \cdot 11^3 = 46585$ and $6^6 = 46656$.

Although the trajectories of `T` are much shorter than those of `f6`, it seems likely that this does not make the problem of deciding whether the mapping `T` is contracting essentially easier – even for mappings with much shorter trajectories than `T` the problem seems to be equally hard. A solution can usually only be found in trivial cases, i.e. for example when there is some k such that applying the kth power of the respective mapping to any integer decreases its absolute value.

4.3 Checking a result by P. Andaloro

In [And00], P. Andaloro has shown that proving that trajectories of integers $n \in 1(16)$ under the Collatz mapping always end up with 1 would be sufficient for proving the Collatz conjecture. In the sequel, this result is verified with RCWA. Checking that the union of the images of the residue class 1(16) under powers of the Collatz mapping T contains $\mathbb{Z} \setminus 0(3)$ is obviously sufficient. Thus we proceed by setting $S := 1(16)$ and successively uniting the set S with its image under T:

Example

```
gap> S := ResidueClass(Integers,16,1);
1(16)
gap> S := Union(S,S^T);
1(16) U 2(24)
gap> S := Union(S,S^T);
1(12) U 2(24) U 17(48) U 33(48)
gap> S := Union(S,S^T);
<union of 30 residue classes (mod 144)>
gap> S := Union(S,S^T);
<union of 42 residue classes (mod 144)>
gap> S := Union(S,S^T);
<union of 172 residue classes (mod 432)>
gap> S := Union(S,S^T);
<union of 676 residue classes (mod 1296)>
gap> S := Union(S,S^T);
<union of 810 residue classes (mod 1296)>
gap> S := Union(S,S^T);
<union of 2638 residue classes (mod 3888)>
gap> S := Union(S,S^T);
<union of 33 residue classes (mod 48)>
gap> S := Union(S,S^T);
<union of 33 residue classes (mod 48)>
gap> Union(S,ResidueClass(Integers,3,0)); # Et voila ...
Integers
```

Further similar computations are shown in Section 4.14.

4.4 Two examples by Matthews and Leigh

In [ML87], K. R. Matthews and G. M. Leigh have shown that two trajectories of the following (surjective, but not injective) mappings are acyclic (mod x) and divergent:

Example

```
gap> x := Indeterminate(GF(4),1);; SetName(x,"x");
gap> R := PolynomialRing(GF(2),1);
GF(2)[x]
gap> ML1 := RcwaMapping(R,x,[[1,0,x],[(x+1)^3,1,x]]*One(R));;
gap> ML2 := RcwaMapping(R,x,[[1,0,x],[(x+1)^2,1,x]]*One(R));;
gap> SetName(ML1,"ML1"); SetName(ML2,"ML2");
gap> Display(ML1);

Rcwa mapping of GF(2)[x] with modulus x

         P mod x          |                      P^ML1
--------------------------+---------------------------------------------------
 0*Z(2)                   | P/x
 Z(2)^0                   | ((x^3+x^2+x+Z(2)^0)*P + Z(2)^0)/x

gap> Display(ML2);

Rcwa mapping of GF(2)[x] with modulus x

         P mod x          |                      P^ML2
--------------------------+---------------------------------------------------
 0*Z(2)                   | P/x
 Z(2)^0                   | ((x^2+Z(2)^0)*P + Z(2)^0)/x

gap> List([ML1,ML2],IsSurjective);
[ true, true ]
gap> List([ML1,ML2],IsInjective);
[ false, false ]
gap> traj1 := Trajectory(ML1,One(R),16,"length");
[ Z(2)^0, x^2+x+Z(2)^0, x^4+x^2+x, x^3+x+Z(2)^0, x^5+x^4+x^2, x^4+x^3+x,
  x^3+x^2+Z(2)^0, x^5+x^2+Z(2)^0, x^7+x^6+x^5+x^3+Z(2)^0,
  x^9+x^7+x^6+x^5+x^3+x+Z(2)^0, x^11+x^10+x^8+x^7+x^6+x^5+x^2,
  x^10+x^9+x^7+x^6+x^5+x^4+x, x^9+x^8+x^6+x^5+x^4+x^3+Z(2)^0,
  x^11+x^8+x^7+x^6+x^4+x+Z(2)^0, x^13+x^12+x^11+x^8+x^7+x^6+x^4,
  x^12+x^11+x^10+x^7+x^6+x^5+x^3 ]
gap> traj2 := Trajectory(ML2,(x^3+x+1)*One(R),16,"length");
[ x^3+x+Z(2)^0, x^4+x+Z(2)^0, x^5+x^3+x^2+x+Z(2)^0, x^6+x^3+Z(2)^0,
  x^7+x^5+x^4+x^2+x, x^6+x^4+x^3+x+Z(2)^0, x^7+x^4+x^3+x+Z(2)^0,
  x^8+x^6+x^5+x^4+x^3+x+Z(2)^0, x^9+x^6+x^3+x+Z(2)^0,
  x^10+x^8+x^7+x^5+x^4+x+Z(2)^0, x^11+x^8+x^7+x^5+x^4+x^3+x^2+x+Z(2)^0,
  x^12+x^10+x^9+x^8+x^7+x^5+Z(2)^0, x^13+x^10+x^7+x^4+x,
  x^12+x^9+x^6+x^3+Z(2)^0, x^13+x^11+x^10+x^8+x^7+x^5+x^4+x^2+x,
  x^12+x^10+x^9+x^7+x^6+x^4+x^3+x+Z(2)^0 ]
```

The pattern which Matthews and Leigh used to show the divergence of the above trajectories can be recognized easily by looking at the corresponding Markov chains with the two states 0 mod x and 1 mod x:

Example

```
gap> traj1modx := Trajectory(ML1,One(R),400,"length") mod x;;
gap> traj2modx := Trajectory(ML2,(x^3+x+1)*One(R),600,"length") mod x;;
gap> List(traj1modx{[1..200]},val->Position([Zero(R),One(R)],val)-1);
[ 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
  1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ]
gap> List(traj2modx{[1..200]},val->Position([Zero(R),One(R)],val)-1);
[ 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 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, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
  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, 1, 1, 1, 1, 1, 1, 1, 1, 0,
  1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
  0, 1, 0, 1, 0, 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, 1, 1, 1, 1 ]
```

What is important here are the lengths of the intervals between two changes from one state to the other:

Example

```
gap> ChangePoints := l -> Filtered([1..Length(l)-1],pos->l[pos]<>l[pos+1]);;
gap> Diffs := l -> List([1..Length(l)-1],pos->l[pos+1]-l[pos]);;
gap> Diffs(ChangePoints(traj1modx)); # The pattern in the first ...
[ 1, 1, 2, 4, 2, 2, 4, 8, 4, 4, 8, 16, 8, 8, 16, 32, 16, 16, 32, 64, 32, 32,
  64 ]
gap> Diffs(ChangePoints(traj2modx)); # ... and in the second example.
[ 1, 7, 1, 1, 1, 13, 1, 1, 1, 1, 1, 1, 1, 25, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 49, 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, 97, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 193, 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, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1 ]
gap> Diffs(ChangePoints(last)); # Make this a bit more obvious.
[ 1, 3, 1, 7, 1, 15, 1, 31, 1, 63, 1 ]
```

This looks clearly acyclic, thus the trajectories diverge. Needless to say however that this computational evidence does not replace the proof along these lines given in the article cited above, but just sheds a light on the idea behind it.

4.5 Exploring the structure of a wild rcwa group

In this example, a simple attempt to should be made to investigate the structure of a given wild group by finding orders of torsion elements. In general, determining the structure of a given wild group computationally seems to be a very hard task. First of all, the group in question has to be defined:

Example

```
gap> u := RcwaMapping([[3,0,5],[9,1,5],[3,-1,5],[9,-2,5],[9,4,5]]);;
gap> SetName(u,"u");
gap> Display(u);

Rcwa mapping of Z with modulus 5

                n mod 5                |                 n^u
---------------------------------------+---------------------------------------
  0                                    | 3n/5
  1                                    | (9n + 1)/5
  2                                    | (3n - 1)/5
  3                                    | (9n - 2)/5
  4                                    | (9n + 4)/5

gap> nu := RcwaMapping([[1,1,1]]);
Rcwa mapping of Z: n -> n + 1
gap> SetName(nu,"nu");
gap> G := Group(u,nu);
<rcwa group over Z with 2 generators>
gap> IsTame(G);
false
```

Now we would like to know which orders torsion elements of G can have – taking a look at the above generators it seems to make sense to try commutators:

Example

```
gap> l := Filtered([0..100],k->IsTame(Comm(u,nu^k)));
[ 0, 2, 3, 5, 6, 9, 10, 12, 13, 15, 17, 18, 20, 21, 24, 25, 27, 28, 30, 32,
  33, 35, 36, 39, 40, 42, 43, 45, 47, 48, 50, 51, 54, 55, 57, 58, 60, 62, 63,
  65, 66, 69, 70, 72, 73, 75, 77, 78, 80, 81, 84, 85, 87, 88, 90, 92, 93, 95,
  96, 99, 100 ]
gap> List(l,k->Order(Comm(u,nu^k)));
[ 1, 6, 5, 3, 5, 5, 3, infinity, 7, infinity, 7, 5, 3, infinity, infinity, 3,
  5, 7, infinity, 7, infinity, 3, 5, 5, 3, 5, infinity, infinity, infinity,
  5, 3, 5, 5, 3, infinity, 7, infinity, 7, 5, 3, infinity, infinity, 3, 5, 7,
  infinity, 7, infinity, 3, 5, 5, 3, 5, infinity, infinity, infinity, 5, 3,
  5, 5, 3 ]
```

Example

```
gap> Display(Comm(u,nu^13));

Bijective rcwa mapping of Z with modulus 9

                n mod 9                |                  n^f
---------------------------------------+--------------------------------------
  0 3 6                                | n + 5
  1 4 7                                | 3n - 9
  2 8                                  | n - 11
  5                                    | (n + 16)/3

gap> Order(Comm(u,nu^13));
7
gap> u2 := u^2;
<wild bijective rcwa mapping of Z with modulus 25>
gap> Filtered([1..16],k->IsTame(Comm(u2,nu^k))); # k < 15 -> commutator wild!
[ 15 ]
gap> Order(Comm(u2,nu^15));
infinity
gap> u2nu17 := Comm(u2,nu^17);
<bijective rcwa mapping of Z with modulus 81>
gap> orbs := ShortOrbits(Group(u2nu17),[-100..100],100);;
gap> List(orbs,Length);
[ 72, 72, 73, 72, 73, 72, 72, 73, 72, 72, 72, 73, 72, 72, 73, 72, 72, 73, 72,
  72, 73, 72, 72 ]
gap> Lcm(last);
5256
gap> u2nu17^5256; # This element has indeed order 2^3*3^2*73 = 5256.
IdentityMapping( Integers )
gap> u2nu18 := Comm(u2,nu^18);
<bijective rcwa mapping of Z with modulus 81>
gap> orbs := ShortOrbits(Group(u2nu18),[-100..100],100);;
gap> List(orbs,Length);
[ 22, 22, 22, 21, 22, 22, 22, 21, 21, 22, 22, 21, 22, 21, 22, 22, 21, 22, 22,
  21, 22, 22, 21 ]
gap> Lcm(last);
462
gap> u2nu18^462; # This is an element of order 2*3*7*11 = 462.
IdentityMapping( Integers )
gap> Order(Comm(u2,nu^20));
29
gap> Order(Comm(u2,nu^25));
9
gap> Order(Comm(u2,nu^30));
15
```

Thus even this rather simple-minded approach reveals various different orders of torsion elements, and the involved primes are also not all quite “small”.

4.6 A wild rcwa mapping which has only finite cycles

Some wild rcwa mappings of $\mathbb{Z}$ have only finite cycles. In this section, a permutation is examined which can be shown to be such a mapping and which is likely to be something like a "minimal" example.

Over $R = \mathrm{GF}(q)[x]$, constructing such mappings is easy since the degree function gives rise to a partition of R into finite sets which is left invariant by suitable wild rcwa mappings. Over $R = \mathbb{Z}$ however the situation looks different – there is no such "natural" partition into finite sets which can be fixed by a wild rcwa mapping.

Example

```
gap> kappa := RcwaMapping([[1,0,1],[1,0,1],[3,2,2],[1,-1,1],
>                          [2,0,1],[1,0,1],[3,2,2],[1,-1,1],
>                          [1,1,3],[1,0,1],[3,2,2],[2,-2,1]]);;
gap> SetName(kappa,"kappa");
gap> List([-5..5],k->Modulus(kappa^k));
[ 7776, 1296, 432, 72, 24, 1, 12, 72, 144, 864, 1728 ]
gap> Display(kappa);

Bijective rcwa mapping of Z with modulus 12

               n mod 12                |               n^kappa
---------------------------------------+--------------------------------------
   0  1  5  9                          | n
   2  6 10                             | (3n + 2)/2
   3  7                                | n - 1
   4                                   | 2n
   8                                   | (n + 1)/3
  11                                   | 2n - 2

gap> List([-32..32],n->Length(Cycle(kappa,n)));
[ 4, 1, 4, 4, 7, 1, 10, 10, 1, 1, 4, 4, 7, 1, 10, 10, 4, 1, 7, 7, 1, 1, 7, 7,
  4, 1, 4, 4, 2, 1, 1, 2, 1, 1, 4, 4, 4, 1, 7, 7, 4, 1, 7, 7, 1, 1, 10, 10,
  7, 1, 4, 4, 7, 1, 10, 10, 1, 1, 4, 4, 4, 1, 13, 13, 7 ]
gap> List([2..14],k->Maximum(List([1..2^k],n->Length(Cycle(kappa,n)))));
[ 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40 ]
gap> List([2..14],k->Length(Cycle(kappa,2^k-2)));
[ 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40 ]
gap> Cycle(kappa,2^12-2);
[ 4094, 6142, 9214, 13822, 20734, 31102, 46654, 69982, 104974, 157462,
  236194, 354292, 708584, 236195, 472388, 157463, 314924, 104975, 209948,
  69983, 139964, 46655, 93308, 31103, 62204, 20735, 41468, 13823, 27644,
  9215, 18428, 6143, 12284, 4095 ]
gap> last mod 12;
[ 2, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 4, 8, 11, 8, 11, 8, 11, 8, 11,
  8, 11, 8, 11, 8, 11, 8, 11, 8, 11, 8, 11, 8, 3 ]
gap> lengthstatistics := Collected(List(ShortOrbits(Group(kappa),
>                                                  [1..12^4],100),Length));
[ [ 1, 6912 ], [ 4, 1728 ], [ 7, 864 ], [ 10, 432 ], [ 13, 216 ],
  [ 16, 108 ], [ 19, 54 ], [ 22, 27 ], [ 25, 13 ], [ 28, 7 ], [ 31, 3 ],
  [ 34, 2 ], [ 37, 1 ], [ 40, 1 ] ]
```

We would like to determine a partition of ℤ into unions of cycles of equal length:

Example

```
gap> C := [Difference(Integers,MovedPoints(kappa))];; pow := [kappa^0];;
gap> rc := function(r,m) return ResidueClass(r,m); end;;
gap> for i in [1..3] do
>      Add(pow,kappa^i);
>      C[i+1] := Difference(rc(2,4),
>                           Union(Union(C{[1..i]}),
>                                 Union(List([0..i],
>                                       j->Intersection(rc(2,4)^pow[j+1],
>                                                       rc(2,4)^(pow[i-j+1]^-1))))));
>    od;
gap> C;
[ 1(4) U 0(12) U [ -2 ], 2(24) U 18(24), 6(48) U 38(48) U 10(72) U 58(72),
  <union of 38 residue classes (mod 864)> ]
gap> List(C,S->Length(Cycle(kappa,S)));
[ 1, 4, 7, 10 ]
gap> Cycle(kappa,C[1]);
[ 1(4) U 0(12) U [ -2 ] ]
gap> Cycle(kappa,C[2]);
[ 2(24) U 18(24), 4(36) U 28(36), 8(72) U 56(72), 3(24) U 19(24) ]
gap> cycle7 := Cycle(kappa,C[3]);;
gap> for S in cycle7 do View(S); Print("\n"); od;
6(48) U 38(48) U 10(72) U 58(72)
10(72) U 58(72) U 16(108) U 88(108)
16(108) U 88(108) U 32(216) U 176(216)
11(72) U 59(72) U 32(216) U 176(216)
11(72) U 59(72) U 20(144) U 116(144)
7(48) U 39(48) U 20(144) U 116(144)
6(48) U 7(48) U 38(48) U 39(48)
gap> cycle10 := Cycle(kappa,C[4]);;
gap> for S in cycle10 do View(S); Print("\n"); od;
<union of 38 residue classes (mod 864)>
<union of 38 residue classes (mod 1296)>
<union of 12 residue classes (mod 648)>
<union of 12 residue classes (mod 648)>
<union of 22 residue classes (mod 1296)>
<union of 12 residue classes (mod 432)>
<union of 22 residue classes (mod 864)>
<union of 12 residue classes (mod 288)>
<union of 14 residue classes (mod 288)>
<union of 16 residue classes (mod 288)>
gap> List(cycle10,Density);
[ 19/432, 19/648, 1/54, 1/54, 11/648, 1/36, 11/432, 1/24, 7/144, 1/18 ]
gap> List(last,Float);
[ 0.0439815, 0.029321, 0.0185185, 0.0185185, 0.0169753, 0.0277778, 0.025463,
  0.0416667, 0.0486111, 0.0555556 ]
gap> Sum(last2);
47/144
gap> Density(Union(cycle10));
47/432
```

Example

```
gap> P := List(C,S->Union(Cycle(kappa,S)));;
gap> for S in P do View(S); Print("\n"); od;
1(4) U 0(12) U [ -2 ]
<union of 18 residue classes (mod 72)>
<union of 78 residue classes (mod 432)>
<union of 282 residue classes (mod 2592)>
gap> P2 := AsUnionOfFewClasses(P[2]);
[ 2(24), 3(24), 18(24), 19(24), 4(36), 28(36), 8(72), 56(72) ]
gap> Permutation(kappa,P2);
(1,5,7,2)(3,6,8,4)
gap> P3 := AsUnionOfFewClasses(P[3]);
[ 6(48), 7(48), 38(48), 39(48), 10(72), 11(72), 58(72), 59(72), 16(108),
  88(108), 20(144), 116(144), 32(216), 176(216) ]
gap> Permutation(kappa,P3);
(1,5,9,13,6,11,2)(3,7,10,14,8,12,4)
gap> P4 := AsUnionOfFewClasses(P[4]);
[ 14(96), 15(96), 78(96), 79(96), 22(144), 23(144), 118(144), 119(144),
  34(216), 35(216), 178(216), 179(216), 44(288), 236(288), 52(324), 268(324),
  68(432), 356(432), 104(648), 536(648) ]
gap> Permutation(kappa,P4);
(1,5,9,15,19,10,17,6,13,2)(3,7,11,16,20,12,18,8,14,4)
gap> List(P,S->Set(List(Intersection([1..12^4],S),n->Length(Cycle(kappa,n)))));
[ [ 1 ], [ 4 ], [ 7 ], [ 10 ] ]
gap> Set(List(Intersection([1..12^4],Difference(Integers,Union(P))),
>              n->Length(Cycle(kappa,n))));
[ 13, 16, 19, 22, 25, 28, 31, 34, 37, 40 ]
```

Finally, the permutation `kappa` should be factored into involutions (this time "by hand", for purposes of illustration):

Example

```
gap> elm1 := kappa;
kappa
gap> Multpk(elm1,2,1)^elm1;
8(12)
gap> Multpk(elm1,2,-1)^elm1;
4(6)
gap> Multpk(elm1,3,1)^elm1;
4(6)
gap> Multpk(elm1,3,-1)^elm1;
3(4)
gap> fact1 := RcwaMapping([[rc(4,6),rc(8,12)]]);
<rcwa mapping of Z with modulus 12>
```

Example

```
gap> elm2 := elm1/fact1;
<bijective rcwa mapping of Z with modulus 12>
gap> Display(elm2);

Bijective rcwa mapping of Z with modulus 12

               n mod 12                |                 n^f
---------------------------------------+---------------------------------------
   0  1  4  5  9                       | n
   2  6 10                             | 3n + 2
   3  7 11                             | n - 1
   8                                   | (n + 1)/3

gap> Multpk(elm2,3,1)^elm2;
8(12)
gap> Multpk(elm2,3,-1)^elm2;
3(4)
gap> fact2 := RcwaMapping([[rc(3,4),rc(8,12)]]);
<rcwa mapping of Z with modulus 12>
gap> elm3 := elm2/fact2;
<bijective rcwa mapping of Z with modulus 4>
gap> Display(elm3);

Bijective rcwa mapping of Z with modulus 4

                n mod 4                |                 n^f
---------------------------------------+---------------------------------------
  0 1                                  | n
  2                                    | n + 1
  3                                    | n - 1

gap> fact3 := RcwaMapping([[rc(2,4),rc(3,4)]]);
<rcwa mapping of Z with modulus 4>
gap> elm4 := elm3/fact3;
IdentityMapping( Integers )
gap> kappafacts := [ fact3, fact2, fact1 ];
[ <bijective rcwa mapping of Z with modulus 4>,
  <bijective rcwa mapping of Z with modulus 12>,
  <bijective rcwa mapping of Z with modulus 12> ]
gap> List(kappafacts,Order);
[ 2, 2, 2 ]
gap> kappa = Product(kappafacts);
true
```

4.7 An abelian rcwa group over a polynomial ring

In this section, a wild rcwa group over GF(4)[x] should be invstigated, which happens to be abelian. Of course in general, rcwa groups also over this ring are usually far from being abelian (see below). We start by defining this group:

Example

```
gap> x := Indeterminate(GF(4),1);; SetName(x,"x");
gap> R := PolynomialRing(GF(4),1);
GF(4)[x]
gap> e := One(GF(4));;
gap> p := x^2 + x + e;;    q := x^2 + e;;
gap> r := x^2 + x + Z(4);; s := x^2 + x + Z(4)^2;;
gap> cg := List( AllResidues(R,x^2), pol -> [ p, p * pol mod q, q ] );;
gap> ch := List( AllResidues(R,x^2), pol -> [ r, r * pol mod s, s ] );;
gap> g := RcwaMapping( R, q, cg );
<rcwa mapping of GF(4)[x] with modulus x^2+Z(2)^0>
gap> h := RcwaMapping( R, s, ch );
<rcwa mapping of GF(4)[x] with modulus x^2+x+Z(2^2)^2>
gap> List([g,h],Order);
[ infinity, infinity ]
gap> List([g,h],IsTame);
[ false, false ]
gap> G := Group(g,h);
<rcwa group over GF(4)[x] with 2 generators>
gap> IsAbelian(G);
true
```

Now we compute the action of the group `G` on one of its orbits, and make some statistics of the orbits of `G` containing polynomials of degree less than 4:

Example

```
gap> orb := Orbit(G,x^5);
[ x^5, x^5+x^4+x^2+Z(2)^0, x^5+x^3+x^2+Z(2^2)*x+Z(2)^0, x^5+x^3,
  x^5+x^4+x^3+x^2+Z(2^2)^2*x+Z(2^2)^2, x^5+x, x^5+x^4+x^3, x^5+x^2+Z(2^2)^2*x,
  x^5+x^4+x^2+x, x^5+x^3+x^2+Z(2^2)^2*x+Z(2)^0, x^5+x^4+Z(2^2)*x+Z(2^2),
  x^5+x^3+x, x^5+x^4+x^3+x^2+Z(2^2)*x+Z(2^2), x^5+x^4+x^3+x+Z(2)^0,
  x^5+x^2+Z(2^2)*x, x^5+x^4+Z(2^2)^2*x+Z(2^2)^2 ]
gap> H := Action(G,orb);
Group([ (1,2,4,7,6,9,12,14)(3,5,8,11,10,13,15,16),
  (1,3,6,10)(2,5,9,13)(4,8,12,15)(7,11,14,16) ])
gap> IsAbelian(H); # check ...
true
gap> Exponent(H);
8
gap> Collected(List(ShortOrbits(G,AllResidues(R,x^4),100),Length));
[ [ 1, 4 ], [ 2, 6 ], [ 4, 12 ], [ 8, 24 ] ]
```

Changing the generators a little causes the group structure to change a lot:

Example

```
gap> cg[1][2] := cg[1][2] + (x^2 + e) * p * q;;
gap> ch[7][2] := ch[7][2] + x * r * s;;
gap> g := RcwaMapping( R, q, cg );; h := RcwaMapping( R, s, ch );;
gap> G := Group(g,h);
<rcwa group over GF(4)[x] with 2 generators>
gap> orb := Orbit(G,Zero(R));;
gap> Length(orb);
87
gap> Collected(List(orb,DegreeOfLaurentPolynomial));
[ [ 1, 2 ], [ 2, 4 ], [ 3, 16 ], [ 4, 64 ], [ infinity, 1 ] ]
gap> H := Action(G,orb);
<permutation group with 2 generators>
gap> IsNaturalAlternatingGroup(H);
true
gap> orb := Orbit(G,x^6);;
gap> Length(orb);
512
gap> H := Action(G,orb);
<permutation group with 2 generators>
gap> IsNaturalSymmetricGroup(H) or IsNaturalAlternatingGroup(H);
false
gap> blk := Blocks(H,[1..512]);;
gap> List(blk,Length);
[ 128, 128, 128, 128 ]
gap> Action(H,blk,OnSets);
Group([ (1,2)(3,4), (1,3)(2,4) ])
```

Thus the modified group has a quotient isomorphic to the alternating group of degree 87, and a quotient isomorphic to some wreath product or a subgroup thereof acting transitively, but not primitively on 512 points.

4.8 An rcwa representation of a small group

In the sequel, an rcwa representation of the 3-Sylow-subgroup of the symmetric group on 9 points is given. Of course this group has a very nice permutation representation, hence for computational purposes one does not gain anything here.

Example

```
gap> r := RcwaMapping([[1,0,1],[1,1,1],[3,-3,1],
>                      [1,0,3],[1,1,1],[3,-3,1],
>                      [1,0,1],[1,1,1],[3,-3,1]]);;
gap> s := RcwaMapping([[1,0,1],[1,1,1],[3,6,1],
>                      [1,0,3],[1,1,1],[3,6,1],
>                      [1,0,1],[1,1,1],[3,-21,1]]);;
gap> SetName(r,"r"); SetName(s,"s");
```

Example

```
gap> Display(r);

Rcwa mapping of Z with modulus 9

                n mod 9                 |                 n^r
---------------------------------------+---------------------------------------
  0 6                                   | n
  1 4 7                                 | n + 1
  2 5 8                                 | 3n - 3
  3                                     | n/3

gap> Display(s);

Rcwa mapping of Z with modulus 9

                n mod 9                 |                 n^s
---------------------------------------+---------------------------------------
  0 6                                   | n
  1 4 7                                 | n + 1
  2 5                                   | 3n + 6
  3                                     | n/3
  8                                     | 3n - 21

gap> G := Group(r,s);
<rcwa group over Z with 2 generators>
gap> H := SylowSubgroup(SymmetricGroup(9),3);
Group([ (1,2,3), (4,5,6), (7,8,9), (1,4,7)(2,5,8)(3,6,9) ])
gap> phi := InverseGeneralMapping(IsomorphismGroups(G,H));;
gap> (1,2,3)^phi;
<bijective rcwa mapping of Z with modulus 27>
```

4.9 An rcwa representation of the symmetric group on 10 points

In this section, an rcwa representation of the symmetric group on 10 points should be investigated. We start by defining some bijections of infinite order and computing commutators:

Example

```
gap> a := RcwaMapping([[3,0,2],[3, 1,4],[3,0,2],[3,-1,4]]);;
gap> b := RcwaMapping([[3,0,2],[3,13,4],[3,0,2],[3,-1,4]]);;
gap> c := RcwaMapping([[3,0,2],[3, 1,4],[3,0,2],[3,11,4]]);;
gap> SetName(a,"a"); SetName(b,"b"); SetName(c,"c");
gap> List([a,b,c],Order);
[ infinity, infinity, infinity ]
gap> ab := Comm(a,b);; ac := Comm(a,c);; bc := Comm(b,c);;
gap> SetName(ab,"[a,b]"); SetName(ac,"[a,c]"); SetName(bc,"[b,c]");
gap> List([ab,ac,bc],Order);
[ 6, 6, 12 ]
```

Now we would like to have a look at [a,b] ...

Example

```
gap> Display(ab);

Bijective rcwa mapping of Z with modulus 18, of order 6

              n mod 18                |               n^[a,b]
--------------------------------------+--------------------------------------
   0  2  3  8  9 11 12 17             | n
   1 10                               | 2n - 5
   4  7 13 16                         | n + 3
   5 14                               | 2n - 4
   6                                  | (n + 2)/2
  15                                  | (n - 5)/2
```

... form the group generated by [a,b] and [a,c] and compute its action on one of its orbits:

Example

```
gap> G := Group(ab,ac);
<rcwa group over Z with 2 generators>
gap> orb := Orbit(G,1);
[ -15, -12, -7, -6, -5, -4, -3, -2, -1, 1 ]
gap> H := Action(G,orb);
Group([ (2,5,8,10,7,6), (1,3,6,9,4,5) ])
gap> Size(H);
3628800
gap> Size(G); # G acts faithful on orb.
3628800
```

Hence the group G is isomorphic to the symmetric group on 10 points and acts faithfully on the orbit containing 1. Another question is which groups arise if we take as generators either ab, ac or bc and the mapping t, which maps each integer to its additive inverse:

Example

```
gap> t := RcwaMapping([[-1,0,1]]);
Rcwa mapping of Z: n -> -n
gap> Order(t);
2
gap> G := Group(ab,t);
<rcwa group over Z with 2 generators>
gap> Size(G);
7257600
gap> H := Image(IsomorphismPermGroup(G));;
gap> H2 := Group((1,2),(1,2,3,4,5,6,7,8,9,10),(11,12));;
gap> IsomorphismGroups(H,H2) <> fail; # H = C2 x S10
true
```

Thus the group generated by `ab` and `t` is isomorphic to $C_2 \times S_{10}$. The next group is an extension of a perfect group of order 960:

Example

```
gap> G := Group(ac,t);;
gap> Size(G);
3840
gap> H := Image(IsomorphismPermGroup(G));;
gap> P := DerivedSubgroup(H);;
gap> Size(P);
960
gap> IsPerfect(P);
true
gap> PerfectGroup(PerfectIdentification(P));
A5 2^4'
```

The last group is infinite:

Example

```
gap> G := Group(bc,t);;
gap> Size(G);
infinity
gap> Order(bc*t);
infinity
gap> Modulus(G);
18
gap> ResidueClassUnionViewingFormat("short");
gap> RespectedPartition(G);
[ 0(18), 1(18), 2(18), 4(18), 5(18), 7(18), 8(18), 9(18), 10(18), 11(18),
  13(18), 14(18), 16(18), 17(18), 3(36), 6(36), 12(36), 15(36), 21(36),
  24(36), 30(36), 33(36) ]
gap> D := DerivedSubgroup(ActionOnRespectedPartition(G));;
gap> DegreeAction(D);
20
gap> IsPerfect(D);
true
gap> Size(D);
928972800
gap> RankOfKernelOfActionOnRespectedPartition(G:ProperSubgroupAllowed);
9
```

4.10 Checking for solvability

Is the group generated by the mappings `a` and `b` from the last paragraph solvable?

This group is wild. Presently there is no general method available for testing wild rcwa groups for solvability. But nevertheless, for the given group this question can be decided to the negative. The idea is to find a subgroup `U` which acts on a finite set `S` of integers, and induces on `S` a non-solvable finite permutation group:

Example

```
gap> G := Group(a,b);;
gap> ShortOrbits(Group(Comm(a,b)),[-10..10],100);
[ [ -10 ], [ -9 ], [ -30, -21, -14, -13, -11, -8 ], [ -7 ], [ -6 ],
  [ -12, -5, -4, -3, -2, 1 ], [ -1 ], [ 0 ], [ 2 ], [ 3 ],
  [ 4, 5, 6, 7, 10, 15 ], [ 8 ], [ 9 ] ]
gap> S := [ 4, 5, 6, 7, 10, 15 ];;
gap> Cycle(Comm(a,b),4);
[ 4, 7, 10, 15, 5, 6 ]
gap> elm := RepresentativeAction(G,S,Permuted(S,(1,4)),OnTuples);
<bijective rcwa mapping of Z with modulus 81>
gap> List(S,n->n^elm);
[ 7, 5, 6, 4, 10, 15 ]
gap> U := Group(Comm(a,b),elm);
<rcwa group over Z with 2 generators>
gap> Action(U,S);
Group([ (1,4,5,6,2,3), (1,4) ])
gap> IsNaturalSymmetricGroup(last);
true
```

Thus, the subgroup `U` induces on `S` a natural symmetric group of degree 6. Hence the group `G` is not solvable, as claimed. We finish this example by factoring the group element `elm` into generators:

Example

```
gap> F := FreeGroup("a","b");
<free group on the generators [ a, b ]>
gap> RepresentativeActionPreImage(G,S,Permuted(S,(1,4)),OnTuples,F);
a^-2*b^-2*a*b*a^-1*b*a*b^-2*a
gap> a^-2*b^-2*a*b*a^-1*b*a*b^-2*a = elm;
true
```

4.11 Some examples over (semi)localizations of the integers

We start with something one can observe when trying to “transfer” an rcwa mapping from the ring of integers to one of its localizations (we take the mapping a from the previous examples):

Example

```
gap> a2 := RcwaMapping(Z_pi(2),ShallowCopy(Coefficients(a)));
<rcwa mapping of Z_( 2 ) with modulus 4>
gap> IsSurjective(a2); # As expected
true
gap> IsInjective(a2); # Why not??
false
gap> 0^a2;
0
gap> (1/3)^a2; # That's the reason!
0
```

The above can also be explained easily by pointing out that the modulus of the inverse of a is 3, and that 3 is a unit of $\mathbb{Z}_{(2)}$. Moving to $\mathbb{Z}_{(2,3)}$ solves this problem:

Example

```
gap> a23 := RcwaMapping(Z_pi([2,3]),ShallowCopy(Coefficients(a)));
<rcwa mapping of Z_( 2, 3 ) with modulus 4>
gap> IsBijective(a23);
true
```

We get additional finite cycles, e.g.:

Example

```
gap> List(ShortOrbits(Group(a23),[0..50]/5,50),orb->Cycle(a23,orb[1]));
[ [ 0 ], [ 1/5, 2/5, 3/5 ],
  [ 4/5, 6/5, 9/5, 8/5, 12/5, 18/5, 27/5, 19/5, 13/5, 11/5, 7/5 ], [ 1 ],
  [ 2, 3 ], [ 14/5, 21/5, 17/5 ],
  [ 16/5, 24/5, 36/5, 54/5, 81/5, 62/5, 93/5, 71/5, 52/5, 78/5, 117/5, 89/5,
      68/5, 102/5, 153/5, 116/5, 174/5, 261/5, 197/5, 149/5, 113/5, 86/5,
      129/5, 98/5, 147/5, 109/5, 83/5, 61/5, 47/5, 34/5, 51/5, 37/5, 29/5,
      23/5 ], [ 4, 6, 9, 7, 5 ] ]
gap> List(last,Length);
[ 1, 3, 11, 1, 2, 3, 34, 5 ]
gap> List(ShortOrbits(Group(a23),[0..50]/7,50),orb->Cycle(a23,orb[1]));
[ [ 0 ], [ -1/7, 1/7 ], [ 2/7, 3/7, 4/7, 6/7, 9/7, 5/7 ], [ 1 ], [ 2, 3 ],
  [ 4, 6, 9, 7, 5 ] ]
gap> List(last,Length);
[ 1, 2, 6, 1, 2, 5 ]
```

But the group structure remains invariant under the “transfer” of a group with prime set $\{2,3\}$ from $\mathbb{Z}$ to $\mathbb{Z}_{(2,3)}$:

Example

```
gap> b23 := RcwaMapping(Z_pi([2,3]),ShallowCopy(Coefficients(b)));;
gap> c23 := RcwaMapping(Z_pi([2,3]),ShallowCopy(Coefficients(c)));;
gap> ab23 := Comm(a23,b23);
<rcwa mapping of Z_( 2, 3 ) with modulus 18>
gap> ac23 := Comm(a23,c23);
<rcwa mapping of Z_( 2, 3 ) with modulus 18>
gap> G := Group(ab23,ac23);
<rcwa group over Z_( 2, 3 ) with 2 generators>
gap> S := Intersection(Enumerator(Rationals){[1..200]},Z_pi([2,3]));
[ -12, -11, -10, -9, -8, -7, -6, -5, -4, -3, -12/5, -11/5, -2, -9/5, -12/7,
  -8/5, -11/7, -10/7, -7/5, -9/7, -6/5, -8/7, -12/11, -1, -10/11, -6/7,
  -9/11, -4/5, -8/11, -5/7, -7/11, -3/5, -4/7, -6/11, -5/11, -3/7, -2/5,
  -4/11, -2/7, -3/11, -1/5, -2/11, -1/7, -1/11, 0, 1/13, 1/11, 1/7, 2/13,
  2/11, 1/5, 3/13, 3/11, 2/7, 4/13, 4/11, 5/13, 2/5, 3/7, 5/11, 6/13, 7/13,
  6/11, 4/7, 3/5, 8/13, 7/11, 9/13, 5/7, 8/11, 10/13, 4/5, 9/11, 11/13, 6/7,
  10/11, 12/13, 1, 12/11, 8/7, 13/11, 6/5, 9/7, 7/5, 10/7, 11/7, 8/5, 12/7,
  9/5, 2, 11/5, 12/5, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 ]
gap> orbs := ShortOrbits(G,S,50);;
gap> List(orbs,Length);
[ 10, 10, 1, 10, 1, 10, 10, 10, 10, 10, 1, 10, 10, 10, 1, 10, 10, 10, 10, 10,
  10, 10, 10, 10, 10, 10, 10, 10, 1, 10, 10, 10, 10, 10, 10, 10, 10, 10, 1,
  10, 1, 10, 10, 10, 1, 1, 10, 1, 10 ]
gap> ForAll(orbs,orb->IsNaturalSymmetricGroup(Action(G,orb)));
true
```

“Transferring” a non-invertible rcwa mapping from the ring of integers to some of its (semi)localizations can also turn it into an invertible one:

Example

```
gap> v := RcwaMapping([[6,0,1],[1,-7,2],[6,0,1],[1,-1,1],
>                      [6,0,1],[1, 1,2],[6,0,1],[1,-1,1]]);;
gap> SetName(v,"v");
gap> Display(v);

Rcwa mapping of Z with modulus 8

                n mod 8                |                 n^v
---------------------------------------+--------------------------------------
  0 2 4 6                              | 6n
  1                                    | (n - 7)/2
  3 7                                  | n - 1
  5                                    | (n + 1)/2
```

Example

```
gap> IsInjective(v);
true
gap> IsSurjective(v);
false
gap> Image(v);
1(2) U 2(4) U 0(12)
gap> Difference(Integers,last);
4(12) U 8(12)
gap> v2 := RcwaMapping(Z_pi(2),ShallowCopy(Coefficients(v)));
<rcwa mapping of Z_( 2 ) with modulus 8>
gap> IsBijective(v2);
true
gap> Display(v2^-1);

Bijective rcwa mapping of Z_( 2 ) with modulus 4

                n mod 4               |                 n^f
--------------------------------------+--------------------------------------
  0                                   | 1/3 n / 2
  1                                   | 2 n + 7
  2                                   | n + 1
  3                                   | 2 n - 1

gap> S := ResidueClass(Z_pi(2),2,0);; l := [S];;
gap> for i in [1..10] do Add(l,l[Length(l)]^v2); od;
gap> l; # Visibly v2 is wild ...
[ 0(2), 0(4), 0(8), 0(16), 0(32), 0(64), 0(128), 0(256), 0(512), 0(1024),
  0(2048) ]
gap> w2 := RcwaMapping(Z_pi(2),[[1,0,2],[2,-1,1],[1,1,1],[2,-1,1]]);;
gap> v2w2 := Comm(v2,w2);; SetName(v2w2,"[v2,w2]"); v2w2^-1;;
gap> Display(v2w2);

Bijective rcwa mapping of Z_( 2 ) with modulus 8

                n mod 8               |              n^[v2,w2]
--------------------------------------+--------------------------------------
  0 3 4 7                             | n
  1                                   | n + 4
  2 6                                 | 3 n
  5                                   | n - 4
```

Again, viewed as an rcwa mapping of the integers the commutator given at the end of the example would not be surjective.

4.12 Twisting 257-cycles into an rcwa mapping with modulus 32

We define an rcwa mapping `x` of order 257 with modulus 32. The easiest way to construct such a mapping is to prescribe a transition graph and then to assign suitable affine mappings to its vertices.

Example

```
gap> x := RcwaMapping(
>           [[ 16,  2,  1], [ 16, 18,  1], [  1, 16,  1], [ 16, 18,  1],
>            [  1, 16,  1], [ 16, 18,  1], [  1, 16,  1], [ 16, 18,  1],
>            [  1, 16,  1], [ 16, 18,  1], [  1, 16,  1], [ 16, 18,  1],
>            [  1, 16,  1], [ 16, 18,  1], [  1, 16,  1], [ 16, 18,  1],
>            [  1,  0, 16], [ 16, 18,  1], [  1,-14,  1], [ 16, 18,  1],
>            [  1,-14,  1], [ 16, 18,  1], [  1,-14,  1], [ 16, 18,  1],
>            [  1,-14,  1], [ 16, 18,  1], [  1,-14,  1], [ 16, 18,  1],
>            [  1,-14,  1], [ 16, 18,  1], [  1,-14,  1], [  1,-31,  1]]);;
gap> SetName(x,"x"); Display(x);

Rcwa mapping of Z with modulus 32

               n mod 32                |                n^x
---------------------------------------+---------------------------------------
   0                                   | 16n + 2
   1  3  5  7  9 11 13 15 17 19 21 23  |
  25 27 29                             | 16n + 18
   2  4  6  8 10 12 14                 | n + 16
  16                                   | n/16
  18 20 22 24 26 28 30                 | n - 14
  31                                   | n - 31

gap> Order(x);
257
gap> Cycle(x,[1],0);
[ 0, 2, 18, 4, 20, 6, 22, 8, 24, 10, 26, 12, 28, 14, 30, 16, 1, 34, 50, 36,
  52, 38, 54, 40, 56, 42, 58, 44, 60, 46, 62, 48, 3, 66, 82, 68, 84, 70, 86,
  72, 88, 74, 90, 76, 92, 78, 94, 80, 5, 98, 114, 100, 116, 102, 118, 104,
  120, 106, 122, 108, 124, 110, 126, 112, 7, 130, 146, 132, 148, 134, 150,
  136, 152, 138, 154, 140, 156, 142, 158, 144, 9, 162, 178, 164, 180, 166,
  182, 168, 184, 170, 186, 172, 188, 174, 190, 176, 11, 194, 210, 196, 212,
  198, 214, 200, 216, 202, 218, 204, 220, 206, 222, 208, 13, 226, 242, 228,
  244, 230, 246, 232, 248, 234, 250, 236, 252, 238, 254, 240, 15, 258, 274,
  260, 276, 262, 278, 264, 280, 266, 282, 268, 284, 270, 286, 272, 17, 290,
  306, 292, 308, 294, 310, 296, 312, 298, 314, 300, 316, 302, 318, 304, 19,
  322, 338, 324, 340, 326, 342, 328, 344, 330, 346, 332, 348, 334, 350, 336,
  21, 354, 370, 356, 372, 358, 374, 360, 376, 362, 378, 364, 380, 366, 382,
  368, 23, 386, 402, 388, 404, 390, 406, 392, 408, 394, 410, 396, 412, 398,
  414, 400, 25, 418, 434, 420, 436, 422, 438, 424, 440, 426, 442, 428, 444,
  430, 446, 432, 27, 450, 466, 452, 468, 454, 470, 456, 472, 458, 474, 460,
  476, 462, 478, 464, 29, 482, 498, 484, 500, 486, 502, 488, 504, 490, 506,
  492, 508, 494, 510, 496, 31 ]
gap> Length(last);
257
```

4.13 The behaviour of the moduli of powers

In this section some examples are given, which illustrate how different the series of the moduli of powers of a given rcwa mapping of the integers can look like.

Example

```
gap> List([0..4],i->Modulus(a^i));
[ 1, 4, 16, 64, 256 ]
gap> List([0..6],i->Modulus(ab^i));
[ 1, 18, 18, 18, 18, 18, 1 ]
gap> List([0..3],i->Modulus(r^i));
[ 1, 9, 9, 1 ]
gap> List([0..9],i->Modulus(s^i));
[ 1, 9, 9, 27, 27, 27, 27, 27, 27, 1 ]
gap> g := RcwaMapping([[2,2,1],[1,4,1],[1,0,2],[2,2,1],[1,-4,1],[1,-2,1]]);;
gap> List([0..7],i->Modulus(g^i));
[ 1, 6, 12, 12, 12, 12, 6, 1 ]
gap> u := RcwaMapping([[3,0,5],[9,1,5],[3,-1,5],[9,-2,5],[9,4,5]]);;
gap> List([0..3],i->Modulus(u^i));
[ 1, 5, 25, 125 ]
gap> v6 := RcwaMapping([[-1,2,1],[1,-1,1],[1,-1,1]]);;
gap> List([0..6],i->Modulus(v6^i));
[ 1, 3, 3, 3, 3, 3, 1 ]
gap> w8 := RcwaMapping([[-1,3,1],[1,-1,1],[1,-1,1],[1,-1,1]]);;
gap> List([0..8],i->Modulus(w8^i));
[ 1, 4, 4, 4, 4, 4, 4, 4, 1 ]
gap> z := RcwaMapping([[2,  1, 1],[1,  1,1],[2, -1,1],[2, -2,1],
>                      [1,  6, 2],[1,  1,1],[1, -6,2],[2,  5,1],
>                      [1,  6, 2],[1,  1,1],[1,  1,1],[2, -5,1],
>                      [1,  0, 1],[1, -4,1],[1,  0,1],[2,-10,1]]);;
gap> SetName(z,"z");
gap> IsBijective(z);
true
gap> Display(z);

Bijective rcwa mapping of Z with modulus 16

               n mod 16                |               n^z
---------------------------------------+---------------------------------------
   0                                   | 2n + 1
   1  5  9 10                          | n + 1
   2                                   | 2n - 1
   3                                   | 2n - 2
   4  8                                | (n + 6)/2
   6                                   | (n - 6)/2
   7                                   | 2n + 5
  11                                   | 2n - 5
  12 14                                | n
  13                                   | n - 4
  15                                   | 2n - 10
```

Example

```
gap> List([0..25],i->Modulus(z^i));
[ 1, 16, 32, 64, 64, 128, 128, 128, 128, 128, 128, 256, 256, 256, 256, 256,
  256, 512, 512, 512, 512, 512, 512, 1024, 1024, 1024 ]
gap> e1 := RcwaMapping([[1,4,1],[2,0,1],[1,0,2],[2,0,1]]);;
gap> e2 := RcwaMapping([[1,4,1],[2,0,1],[1,0,2],[1,0,1],
>                      [1,4,1],[2,0,1],[1,0,1],[1,0,1]]);;
gap> List([e1,e2],Order);
[ infinity, infinity ]
gap> List([1..20],i->Modulus(e1^i));
[ 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4 ]
gap> List([1..20],i->Modulus(e2^i));
[ 8, 4, 8, 4, 8, 4, 8, 4, 8, 4, 8, 4, 8, 4, 8, 4, 8, 4, 8, 4 ]
gap> SetName(e1,"e1"); SetName(e2,"e2");
gap> Display(e2);

Bijective rcwa mapping of Z with modulus 8, of order infinity

                n mod 8                |                 n^e2
---------------------------------------+--------------------------------------
  0 4                                  | n + 4
  1 5                                  | 2n
  2                                    | n/2
  3 6 7                                | n

gap> e2^2 = Restriction(RcwaMapping([[1,2,1]]),RcwaMapping([[4,0,1]]));
true
```

4.14 Images and preimages under the Collatz mapping

We have a look at the images of the residue class 1(2) under powers of the Collatz mapping.

Example

```
gap> T := RcwaMapping([[1,0,2],[3,1,2]]);; S0 := ResidueClass(Integers,2,1);;
gap> S1 := S0^T;
2(3)
gap> S2 := S1^T;
1(3) U 8(9)
gap> S3 := S2^T;
2(3) U 4(9)
gap> S4 := S3^T;
1(3) U 2(9) U 8(9)
gap> S5 := S4^T;
2(3) U 1(9) U 4(9)
gap> S6 := S5^T;
1(3) U 2(3)
gap> S7 := S6^T;
1(3) U 2(3)
```

Thus the image gets stable after applying the mapping T for the 6th time. Hence T^6 maps the residue class 1(2) surjectively onto the union of the residue classes 1(3) and 2(3), which is setwisely stabilized by T. Now we would like to determine the preimages of 1(3) resp. 2(3) in 1(2) under T^6. The residue class 1(2) has to be the disjoint union of these sets.

Example

```
gap> U := Intersection(PreImage(T^6,ResidueClass(Integers,3,1)),S0);
<union of 11 residue classes (mod 64)>
gap> V := Intersection(PreImage(T^6,ResidueClass(Integers,3,2)),S0);
<union of 21 residue classes (mod 64)>
gap> AsUnionOfFewClasses(U);
[ 1(64), 5(64), 7(64), 9(64), 21(64), 23(64), 29(64), 31(64), 49(64), 51(64),
  59(64) ]
gap> AsUnionOfFewClasses(V);
[ 3(32), 11(32), 13(32), 15(32), 25(32), 17(64), 19(64), 27(64), 33(64),
  37(64), 39(64), 41(64), 53(64), 55(64), 61(64), 63(64) ]
gap> Union(U,V) = S0 and Intersection(U,V) = [];  # consistency check
true
```

The images of the residue class 0(3) under powers of T look as follows:

Example

```
gap> S0 := ResidueClass(Integers,3,0);
0(3)
gap> S1 := S0^T;
0(3) U 5(9)
gap> S2 := S1^T;
0(3) U 5(9) U 7(9) U 8(27)
gap> S3 := S2^T;
<union of 20 residue classes (mod 27)>
gap> S4 := S3^T;
<union of 73 residue classes (mod 81)>
gap> S5 := S4^T;
<union of 79 residue classes (mod 81)>
gap> S6 := S5^T;
Integers
gap> S7 := S6^T;
Integers
```

Thus, every integer is the image of a multiple of 3 under T^6. This means that it would be sufficient to prove the Collatz conjecture for multiples of 3. We can obtain the corresponding result for multiples of 5 as follows:

Example

```
gap> S := [ResidueClass(Integers,5,0)];
[ 0(5) ]
gap> for i in [1..12] do Add(S,S[i]^T); od;
```

Example

```
gap> for s in S do View(s); Print("\n"); od;
0(5)
0(5) U 8(15)
0(5) U 4(15) U 8(15)
0(5) U 2(15) U 4(15) U 8(15) U 29(45)
<union of 73 residue classes (mod 135)>
<union of 244 residue classes (mod 405)>
<union of 784 residue classes (mod 1215)>
<union of 824 residue classes (mod 1215)>
<union of 2593 residue classes (mod 3645)>
<union of 2647 residue classes (mod 3645)>
<union of 2665 residue classes (mod 3645)>
<union of 2671 residue classes (mod 3645)>
1(3) U 2(3) U 0(15)
gap> Union(S[13],ResidueClass(Integers,3,0));
Integers
gap> List(S,Si->Float(Density(Si)));
[ 0.2, 0.266667, 0.333333, 0.422222, 0.540741, 0.602469, 0.645267, 0.678189,
  0.711385, 0.7262, 0.731139, 0.732785, 0.733333 ]
```

Chapter 5

The Algorithms Implemented in RCWA

Almost all mathematically interesting algorithms implemented in this package will be described in the author's forthcoming thesis [Koh05] in the form of constructive proofs. This chapter provides references to the corresponding theorems, and lists short descriptions of the other algorithms and methods implemented in this package. The word "trivial" as a description means that essentially nothing is done except of storing or recalling one or several values, and "straight forward" means that no sophisticated algorithm is used. The descriptions are kept very informal and short. They are listed in alphabetical order.

`ActionOnRespectedPartition(G)` "Straight forward" after having computed a respected partition by using `RespectedPartition`. One only needs to know how to compute images of residue classes under affine mappings.

`ClassReflection(r,m)` "Trivial".

`ClassShift(r,m)` "Trivial".

`ClassTransposition(r1,m1,r2,m2)` See Remark 2.9.2 in [Koh05].

`Coefficients(f)` "Trivial".

`CoefficientsOnTrajectory(f,n,val,cond,all)` Iterated application of an rcwa mapping, and composition of affine mappings.

`ContractionCentre(f,maxn,bound)` Compute trajectories with starting values from a given interval, until a cycle is reached. Abort if the trajectory exceeds the prescribed bound. Form the union of the detected cycles.

`DecreasingOn(f)` Form the union of the residue classes which are determined by the coefficients as indicated.

`Determinant(sigma)` Evaluation of the given expression. For the mathematical meaning (epimorphism!), see Theorem 2.11.9 in [Koh05].

`DirectProduct(G1,G2, ... )` Restrict the groups `G1`, `G2`, ... to disjoint residue classes. See `Restriction` and Corollary 2.3.3 in [Koh05].

`Display(f)` "Trivial".

`Divergence(f)` Numerical computation of the limit of some series, which seems to converge "often". Caution!!!

`Divisor(f)` Lcm of coefficients, as indicated.

`FactorizationIntoGenerators(f)` This uses a rather sophisticated method which will perhaps some time be published elsewhere. At the moment, termination is not guaranteed. But if it terminates, the result is certain. The strategy is roughly first to make the mapping class-wise order-preserving and balanced, and then to remove all prime factors from multiplier and divisor one after the other in decreasing order by dividing by appropriate class transpositions. The remaining integral mapping can be factored almost similarly easy as a permutation of a finite set can be factored into transpositions.

`FactorizationOnConnectedComponents(f,m)` Call GRAPE to get the connected components of the transition graph, and then compute a partition of the suitably "blown up" coefficient list corresponding to the connected components.

`Image(f)`, `Image(f,S)` "Straight forward" if one can compute images of residue classes under affine mappings and unite and intersect residue classes (Chinese Remainder Theorem). See Lemma 1.2.1 in [Koh05].

`ImageDensity(f)` Evaluation of the given expression.

`g in G` (membership test) Test whether the mapping `g` or its inverse is in the list of generators of `G`. If it is, return `true`. Test whether its prime set is a subset of the prime set of `G`. If not, return `false`. Test if `G` is class-wise order-preserving, and `g` is not. If so, return `false`. Test whether the support of `g` is a subset of the support of `G`. If not, return `false`.

If `G` is not tame, try to factor `g` into generators of `G` using `PreImagesRepresentative`. If successful, return `true`. If `g` is in `G`, this terminates after a finite number of steps. Both runtime and memory requirements are exponential in the word length. If `g` is not in `G`, it runs into an infinite loop. If `G` is tame, proceed as follows:

Test whether the modulus of `g` divides the modulus of `G`. If not, return `false`. Test whether `G` is finite and `g` has infinite order. If so, return `false`. Test whether `g` is tame. If not, return `false`. Compute a respected partition `P` of `G` and the finite permutation group `H` induced by `G` on it (see `RespectedPartition`). Check whether `g` permutes `P`. If not, return `false`. Let `h` be the permutation induced by `g` on `P`. Check whether `h` lies in `H`. If not, return `false`.

If `G` is class-wise order-preserving, do the following: Compute an element `g1` of `G` which acts on `P` like `g`. For this purpose, factor `h` into generators of `H` using `PreImagesRepresentative`. Compute the corresponding product of generators of `G`. Set `k := g/g1`. The mapping `k` is necessarily integral. Compute the kernel `K` of the action of `G` on `P` using `KernelOfActionOnRespectedPartition`. Check whether `k` lies in the kernel of the action of `G` on `P` by using `SolutionIntMat` to decide membership of the coefficient vector (second entry of each triple) of `k` in the lattice spanned by the rows of the matrix `KernelOfActionOnRespectedPartitionHNFMat(G)`. If it is contained, return `true`.

If membership still has not been decided yet, try to factor `g` into generators of `G` using `PreImagesRepresentative`. If successful, return `true`. If `g` is in `G`, this terminates after a finite number of steps. Both runtime and memory requirements are exponential in the word length. If `g` is not in `G`, it runs into an infinite loop.

IncreasingOn(f) Form the union of the residue classes which are determined by the coefficients as indicated.

IntegralConjugate(f), **IntegralConjugate(G)** Uses the algorithm described in the proof of Theorem 2.5.14 in [Koh05].

IntegralizingConjugator(f), **IntegralizingConjugator(G)** Uses the algorithm described in the proof of Theorem 2.5.14 in [Koh05].

Inverse(f) Essentially inversion of affine mappings. See Lemma 1.3.1, Part (b) in [Koh05].

IsClassWiseOrderPreserving(f) Test whether the first entry of all coefficient triples is positive.

IsConjugate(RCWA(Integers),f,g) Test whether `f` and `g` have the same order, and whether either both or none of them is tame. If not, return `false`.

If the mappings are wild, use `ShortCycles` to search for finite cycles not belonging to an infinite series, until their numbers for a particular length differ. This may run into an infinite loop. If it terminates, return `false`.

If the mappings are tame, use the method described in the proof of Theorem 2.5.14 in [Koh05] to construct integral conjugates of `f` and `g`. Then essentially use the algorithm described in the proof of Theorem 2.6.7 in [Koh05] to compute "standard representatives" of the conjugacy classes which the integral conjugates of `f` and `g` belong to. Finally compare these standard representatives, and return `true` if they are equal and `false` if not.

IsInjective(f) See `Image`.

IsIntegral(f) "Trivial".

IsomorphismMatrixGroup(G) Use the algorithm described in the proof of Theorem 2.6.3 in [Koh05].

IsomorphismPermGroup(G) If `G` is wild or `KernelOfActionOnRespectedPartition` is not trivial, return `fail`. Otherwise use `ActionOnRespectedPartition`.

IsomorphismRcwaGroup(G) Uses `RcwaMapping`, Part (d). Currently this works only for finite groups.

IsSurjective(f) See `Image`.

IsTame(G) Checks whether the modulus of the group is non-zero.

IsTame(f) Application of the criteria given in Corollary 2.5.10 and 2.5.12 and Theorem B.8 and B.11 in [Koh05]. For applying the last-mentioned criterium (existence of weakly-connected components of the transition graph which are not strongly-connected), GRAPE is needed.

In addition, some probabilistic methods are used. If the result depends on one of these, a warning is displayed.

IsTransitive(G,Integers) Look for finite orbits, using `ShortOrbits` on a couple of intervals. If a finite orbit is found, return `false`. Test if `G` is finite. If yes, return `false`.

Search for an element `g` and a residue class $r(m)$ such that the restriction of `g` to $r(m)$ is given by $n \mapsto n+m$. Then the cyclic group generated by `g` acts transitively on $r(m)$. The element `g` is searched among the generators of `G`, its powers, its commutators, powers of its commutators and products of few different generators. The search for such an element may run into an infinite loop, as there is no guarantee that the group has a suitable element.

If suitable `g` and $r(m)$ are found, proceed as follows:

Set $S := r(m)$. Set $S := S \cup S^g$ for all generators g of `G`, and repeat this until S remains constant. This may run into an infinite loop.

If it terminates: If $S = \mathbb{Z}$, return `true`, otherwise return `false`.

KernelOfActionOnRespectedPartition(G) Use a random walk through the group `G`. Compute powers of elements encountered along the way which fix the respected partition of `G` which has been computed by `RespectedPartition`. Get vectors from these powers by taking the second entry of each coefficient triple. Form a lattice out of these vectors. Stop if for a while all found vectors already belong to this lattice (this is probabilistic). Bring the lattice to Hermite Normal Form, and transform the rows of the resulting matrix back to rcwa mappings generating the kernel.

KernelOfActionOnRespectedPartitionHNFMat(G) This is a "spin-off" of `KernelOfActionOnRespectedPartition`.

LargestSourcesOfAffineMappings(f) Form unions of residue classes modulo the modulus of the mapping, whose corresponding coefficient triples are equal.

LaTeXObj(f) Collect residue classes those corresponding coefficient triples are equal.

Modulus(G) Searches for a wild element in the group. If unsuccessful, tries to construct a respected partition (see `RespectedPartition`).

Modulus(f) "Trivial".

MovedPoints(G) Needs only forming unions of residue classes and determining fixed points of affine mappings.

Multiplier(f) Lcm of coefficients, as indicated.

Multpk(f,p,k) Form the union of the residue classes modulo the modulus of the mapping, which are determined by the given divisibility criteria for the coefficients of the corresponding affine mapping.

NrConjugacyClassesOfRCWAZOfOrder(ord) The class numbers are taken from Corollary 2.7.1 in [Koh05].

OrbitsModulo(f,m) Use GRAPE to compute the connected components of the transition graph.

OrbitsModulo(G,m) "Straight forward".

Order(f) Test for `IsTame`. If the mapping is not tame, then return `infinity`. Otherwise use Corollary 2.5.10 in [Koh05].

PreImage(f,S) See `Image`.

PreImagesRepresentative(phi,g), PreImagesRepresentatives(phi,g) As indicated in the documentation of these methods. The underlying idea to successively compute two balls around 1 and `g` until they intersect non-trivially is standard in computational group theory. For rcwa groups it would mean wasting both memory and runtime to actually compute group elements. Thus only images of tuples of points are computed and stored.

PrimeSet(f), PrimeSet(G) “Straight forward”.

PrimeSwitch(p) Multiplication of rcwa mappings as indicated.

Print(f) “Trivial”.

f*g Essentially composition of affine mappings. See Lemma 1.3.1, Part (a) in [Koh05].

Random(RCWA(Integers)) Computes a product of “randomly” chosen class shifts, class reflections and class transpositions. This seems to be suitable for generating reasonably good examples.

RankOfKernelOfActionOnRespectedPartition(G) This is a “spin-off” of `KernelOfActionOnRespectedPartition`.

RCWA(R) Attributes are set according to Theorem 2.1.1, Theorem 2.1.2, Corollary 2.1.6 and Theorem 2.12.8 in [Koh05].

RcwaGroupByPermGroup(G) Uses `RcwaMapping`, Part (d).

RcwaMapping (a)-(c): “trivial”, (d): `n^perm - n` for determining the coefficients, (e): “affine mappings by values at two given points”, (f) and (g): “trivial”, (h) and (i): correspond to Lemma 2.1.4 in [Koh05].

RepresentativeAction(G,src,dest,act), RepresentativeActionPreImage As indicated in the documentation of these methods. The underlying idea to successively compute two balls around `src` and `dest` until they intersect non-trivially is standard in computational group theory. Words standing for products of generators of `G` are stored for any image of `src` or `dest`.

RepresentativeAction(G,P1,P2) Arbitrary mapping: see Lemma 2.1.4 in [Koh05]. Tame mapping: see proof of Theorem 2.8.9 in [Koh05]. The former is almost trivial, while the latter is a bit complicate and takes usually also much more time.

RepresentativeAction(RCWA(Integers),f,g) The algorithm used by `IsConjugate` constructs actually also an element `x` such that `f^x = g`.

RespectedPartition(f), RespectedPartition(G) Uses the algorithm described in the proof of Theorem 2.5.8 in [Koh05].

Restriction(g,f) Computes images `(n^g)^f` and preimages `n^f` for sufficiently many integers `n` under the image of `g` under the restriction monomorphism associated to `f`. Then it constructs the desired mapping by `RcwaMapping(m,values)`. Finally, the result is checked by a direct verification that the diagram in Definition 2.3.1 in [Koh05] commutes.

Restriction(G,f) Get a set of generators by applying `Restriction(g,f)` to the generators `g` of `G`.

ShortCycles(f,maxlng) Look for fixed points of affine partial mappings of powers of `f`.

ShortOrbits(G,S,maxlng) "Straight forward".

SetOnWhichMappingIsClassWiseOrderPreserving(f), etc. Form the union of the residue classes modulo the modulus of the mapping, in whose corresponding coefficient triple the first entry is positive, zero resp. negative.

Sign(sigma) Evaluation of the given expression. For the mathematical meaning (epimorphism!), see Theorem 2.12.8 in [Koh05].

Size(G) Test whether the group `G` is tame. If not, return `infinity`. Otherwise use `ActionOnRespectedPartition` to compute the permutation group `H` induced by `G` on a respected partition `P`, and `KernelOfActionOnRespectedPartition` to compute the kernel `K` of the action of `G` on `P`. The group `K` is infinite if and only if one of its generators has infinite order. Return the product of the order of `H` and the order of `K`.

f+g Pointwise addition of affine mappings.

Trajectory(f,n,val,cond), TrajectoryModulo(f,n,m,lng) Iterated application of an rcwa mapping.

TransitionGraph(f,m) "Straight forward" – just run over a suffiently long interval.

TransitionMatrix(f,m) Evaluation of the given expression.

ViewObj(f) "Trivial".

Chapter 6

Installation and auxiliary functions

6.1 Installation

Like any other GAP package, RCWA must be installed in the `pkg` subdirectory of the GAP distribution. This is accomplished by extracting the distribution file in this directory. If you have done this, you can load the package as usual via `LoadPackage( "rcwa" );`. The RCWA Package needs at least version 4.4 of GAP, is completely written in the GAP language and does neither contain nor require external binaries. It needs the package ResClasses for dealing with set-theoretic unions of residue classes, and the package GRAPE [Soi02] for dealing with certain graphs associated to rcwa mappings (cp. `TransitionGraph` (2.8.1)). The binaries of GRAPE are not needed, thus RCWA runs under Windows and on the MacIntosh as well. Warnings concerning missing binaries when GRAPE is loaded can savely be ignored. For the documentation the package GAPDoc [LN02] is needed.

6.2 The Info class of the package

6.2.1 InfoRCWA

◊ `InfoRCWA` (info class)

This is the Info class of the RCWA package. See section *Info Functions* in the GAP Reference Manual for a description of the Info mechanism.

For convenience: `RCWAInfo(n)` is a shorthand for `SetInfoLevel(InfoRCWA,n)`.

6.3 Building the manual

6.3.1 RCWABuildManual

◊ `RCWABuildManual( )` (function)

Returns: Nothing.

This function builds the manual of the RCWA package in the file formats LaTeX, DVI, Postscript, PDF, HTML and ASCII text. This is accomplished using the GAPDoc package by Frank Lübeck and Max Neunhöffer. Building the manual is possible only on UNIX systems, but should normally not be necessary as all files generated by this function are included in the distribution file anyway.

6.4 The testing routine

6.4.1 RCWATest

◊ `RCWATest( )` (function)

Returns: Nothing.

Performs tests of the RCWA package. Errors, i.e. differences to the correct results of the test computations, are reported. The processed test files are in the directory `pkg/rcwa/tst`.

References

[And00] P. Andaloro. On total stopping times under 3x+1 iteration. *Fibonacci Quarterly*, 38:73–78, 2000. 56

[Koh05] Stefan Kohl. *Restklassenweise affine Gruppen*. PhD thesis, Universität Stuttgart, to appear, 2005. 7, 79, 80, 81, 82, 83, 84

[Lag04] Jeffrey C. Lagarias. 3x+1 problem annotated bibliography, 2004. http://arxiv.org/abs/math.NT/0309224. 6

[LN02] Frank Lübeck and Max Neunhöffer. *GAPDoc (version 0.99)*. RWTH Aachen, 2002. GAP package, available at http://www.math.rwth-aachen.de/ Frank.Luebeck. 85

[ML87] K. R. Matthews and G. M. Leigh. A generalization of the Syracuse algorithm in $q[x]$. *J. Number Theory*, 25:274–278, 1987. 57

[Soi02] Leonard Soicher. *GRAPE – GRaph Algorithms using PErmutation groups (version 4.1)*. Queen Mary, University of London, 2002. GAP package, available at http://www.gap-system.org. 85

Index

ActionOnRespectedPartition
  G, 44

ClassReflection
  r, m, 12
ClassShift
  r, m, 12
ClassTransposition
  r1, m1, r2, m2, 13
CoefficientsOnTrajectory
  f, n, val, cond, all, 24
Collatz conjecture, 6
Collatz mapping, 6

DecreasingOn
  f, 25
Determinant
  sigma, 17
  sigma, S, 17
DirectProduct
  G1, G2, ..., 43
Div
  f, 18
Divisor
  f, 18

FactorizationIntoGenerators
  g, 16
FactorizationOnConnectedComponents
  f, m, 22

GuessedDivergence
  f, 26

ImageDensity
  f, 26
\in
  g, G, 32
IncreasingOn
  f, 25
InfoRCWA, 85
IntegralConjugate
  f, 45
  G, 45
IntegralizingConjugator
  f, 45
  G, 45
IntegralRcwaGroupsFamily, 46
IsClassWiseOrderPreserving
  f, 19
IsConjugate
  RCWA(Integers), f, g, 40
IsIntegral
  f, 19
IsIntegralRcwaGroup
  G, 46
IsIntegralRcwaMapping
  f, 26
IsModularRcwaGroup
  G, 46
IsModularRcwaMapping
  f, 26
IsomorphismMatrixGroup
  G, 33
IsomorphismPermGroup
  G, 32
IsomorphismRcwaGroup
  G, 30
IsRcwaGroup
  G, 46
IsRcwaMapping
  f, 26
IsSemilocalIntegralRcwaGroup
  G, 46
IsSemilocalIntegralRcwaMapping
  f, 26
IsTame
  G, 31
IsTransitive

G, Integers, 36

LargestSourcesOfAffineMappings
f, 20
LaTeXObj
f, 14
LikelyContractionCentre
f, maxn, bound, 25

Mod
G, 30
Modulus
G, 30
Mult
f, 18
Multiplier
f, 18
Multpk
f, p, k, 20

NrConjugacyClassesOfRCWAZOfOrder
ord, 42

OrbitsModulo
f, m, 21
G, m, 40

PreImagesRepresentative
phi, g, 34
PreImagesRepresentatives
phi, g, 35
PrimeSet
f, 19
G, 31
PrimeSwitch
p, 13
p, k, 13

Random
RCWA(Integers), 28
RCWA
R, 27
rcwa group
definition, 8
element testing, 32
rcwa mapping
definition, 8
modulus, 8
transition graph, 21
transition matrix, 22
RCWABuildManual, 85
RcwaGroupByPermGroup
G, 30
RcwaMapping
coeffs, 10
cycles, 10
m, values, 10
P1, P2, 10
perm, range, 10
pi, coeffs, 10
q, m, coeffs, 10
R, coeffs, 10
R, m, coeffs, 10
RcwaMappingsFamily
R, 26
RCWATest, 86
RepresentativeAction
G, src, dest, act, 37
RCWA(Integers), f, g, 41
RCWA(Integers), P1, P2, 38
RepresentativeActionPreImage
G, src, dest, act, F, 38
RespectedPartition
G, 44
sigma, 44
Restriction
G, f, 43
g, f, 42

SetOnWhichMappingIsClassWiseConstant
f, 21
SetOnWhichMappingIsClassWiseOrderPreserving
f, 21
SetOnWhichMappingIsClassWiseOrderReversing
f, 21
ShortCycles
f, maxlng, 41
ShortOrbits
G, S, maxlng, 39
Sign
sigma, 18
Size
G, 32

Trajectory

f, n, val, cond, 23
`TrajectoryModulo`
f, n, lng, 24
f, n, m, lng, 24
`TransitionGraph`
f, m, 21
`TransitionMatrix`
f, m, 22