ResClasses

Set-Theoretic Computations with Residue Classes

4.7.3

29 July 2022

Stefan Kohl

Stefan Kohl
Email: sk239@st-andrews.ac.uk
Homepage: https://stefan-kohl.github.io/

Abstract

ResClasses is a package for GAP 4 which provides a fully-featured and easy-to-use implementation of set-theoretic unions of residue classes of the integers and of a few other rings.

The class of sets which ResClasses can deal with includes the open and the closed sets in the topology on the respective ring which is induced by taking the set of all residue classes as a basis, as far as the usual restrictions imposed by the finiteness of computing resources permit this.

The package further provides slightly more specialized functionality for unions of residue classes with distinguished representatives and signed moduli.

The ResClasses package is used in a group theoretical context by the RCWA package [Koh16].

Copyright

Contents

Chapter 1

Set-Theoretic Unions of Residue Classes

1.1 Entering residue classes and set-theoretic unions thereof

1.1.1 ResidueClass (by ring, modulus and residue)

▷ `ResidueClass(R, m, r)` (function)
▷ `ResidueClass(m, r)` (function)
▷ `ResidueClass(r, m)` (function)

Returns: in the three-argument form the residue class *r* mod *m* of the ring *R*, and in the two-argument form the residue class *r* mod *m* of the "default ring" (→ `DefaultRing` in the GAP Reference Manual) of the arguments.

In the two-argument case, *m* is taken to be the larger and *r* is taken to be the smaller of the arguments. For convenience, it is permitted to enclose the argument list in list brackets.

Residue classes have the property `IsResidueClass`. Rings are regarded as residue class 0 (mod 1), and therefore have this property. There are operations `Modulus` and `Residue` to retrieve the modulus *m* resp. residue *r* of a residue class.

Example

```
gap> ResidueClass(2,3);
The residue class 2(3) of Z
gap> ResidueClass(Z_pi([2,5]),2,1);
The residue class 1(2) of Z_( 2, 5 )
gap> R := PolynomialRing(GF(2),1);;
gap> x := Indeterminate(GF(2),1);; SetName(x,"x");
gap> ResidueClass(R,x+One(R),Zero(R));
The residue class 0 ( mod x+1 ) of GF(2)[x]
```

1.1.2 ResidueClassUnion (by ring, modulus and residues)

▷ `ResidueClassUnion(R, m, r)` (function)
▷ `ResidueClassUnion(R, m, r, included, excluded)` (function)
▷ `ResidueClassUnion(R, cls)` (function)
▷ `ResidueClassUnion(R, cls, included, excluded)` (function)

Returns: in the first two cases, the union of the residue classes *r*[*i*] mod *m* of the ring *R*, plus / minus finite sets *included* and *excluded* of elements of *R*. In the last two cases, the union of the residue classes *cls*[i][1] mod *cls*[i][2] of the ring R=$\mathbb{Z}$, plus / minus finite sets *included* and *excluded* of integers.

For unions of residue classes of the integers, two distinct representations are implemented: in the first representation, a union of residue classes is represented by its modulus m and the list of residues r; this is called the "standard" representation. In the second ("sparse") representation, a union of residue classes $r_1(m_1) \cup \ldots \cup r_k(m_k)$ is represented by the list *cls* of the pairs `[r_i,m_i]`. One can switch between the two representations by using the operations `StandardRep` and `SparseRep`, respectively. The sparse representation allows more efficient computation in terms of time- and memory requirements when computing with unions of "relatively few" residue classes where the lcm of the moduli is "large"; otherwise the standard representation is advantageous. For rings other than $\mathbb{Z}$, presently only the standard representation is available.

Example

```
gap> ResidueClassUnion(Integers,5,[1,2],[3,8],[-4,1]);
(Union of the residue classes 1(5) and 2(5) of Z) U [ 3, 8 ] \ [ -4, 1 ]
gap> ResidueClassUnion(Integers,[[1,2],[0,40],[2,1200]]);
Union of the residue classes 1(2), 0(40) and 2(1200) of Z
gap> ResidueClassUnion(Z_pi([2,3]),8,[3,5]);
Union of the residue classes 3(8) and 5(8) of Z_( 2, 3 )
gap> ResidueClassUnion(R,x^2,[One(R),x],[],[One(R)]);
<union of 2 residue classes (mod x^2) of GF(2)[x]> \ [ 1 ]
```

When talking about a *residue class union* in this chapter, we always mean an object as it is returned by this function.

There are operations `Modulus`, `Residues`, `IncludedElements` and `ExcludedElements` to retrieve the components of a residue class union as they have originally been passed as arguments to `ResidueClassUnion` (1.1.2).

The user has the choice between a longer and more descriptive and a shorter and less bulky output format for residue classes and unions thereof:

Example

```
gap> ResidueClassUnionViewingFormat("short");
gap> ResidueClassUnion(Integers,12,[0,1,4,7,8]);
0(4) U 1(6)
gap> ResidueClassUnionViewingFormat("long");
gap> ResidueClassUnion(Integers,12,[0,1,4,7,8]);
Union of the residue classes 0(4) and 1(6) of Z
```

1.1.3 AllResidueClassesModulo (of a given ring, modulo a given modulus)

▷ `AllResidueClassesModulo(`*R*`, `*m*`)` (function)
▷ `AllResidueClassesModulo(`*m*`)` (function)

Returns: a sorted list of all residue classes (mod m) of the ring R.

If the argument R is omitted it defaults to the default ring of m – cf. the documentation of `DefaultRing` in the GAP reference manual. A transversal for the residue classes (mod m) can be obtained by the operation `AllResidues(`*R*`,`*m*`)`, and their number can be determined by the operation `NumberOfResidues(`*R*`,`*m*`)`.

Example

```
gap> AllResidueClassesModulo(Integers,2);
[ The residue class 0(2) of Z, The residue class 1(2) of Z ]
gap> AllResidueClassesModulo(Z_pi(2),2);
[ The residue class 0(2) of Z_( 2 ), The residue class 1(2) of Z_( 2 ) ]
gap> AllResidueClassesModulo(R,x);
[ The residue class 0 ( mod x ) of GF(2)[x],
  The residue class 1 ( mod x ) of GF(2)[x] ]
gap> AllResidues(R,x^3);
[ 0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1 ]
gap> NumberOfResidues(Z_pi([2,3]),360);
72
```

1.2 Methods for residue class unions

There are methods for `Print`, `String` and `Display` which are applicable to residue class unions. There is a method for `in` which tests whether some ring element lies in a given residue class union.

Example

```
gap> Print(ResidueClass(1,2),"\n");
ResidueClassUnion( Integers, 2, [ 1 ] )
gap> 1 in ResidueClass(1,2);
true
```

There are methods for `Union`, `Intersection`, `Difference` and `IsSubset` available for residue class unions. They also accept finite subsets of the base ring as arguments.

Example

```
gap> S := Union(ResidueClass(0,2),ResidueClass(0,3));
Z \ Union of the residue classes 1(6) and 5(6) of Z
gap> Intersection(S,ResidueClass(0,7));
Union of the residue classes 0(14) and 21(42) of Z
gap> Difference(S,ResidueClass(2,4));
Union of the residue classes 0(4) and 3(6) of Z
gap> IsSubset(ResidueClass(0,2),ResidueClass(4,8));
true
gap> Union(S,[1..10]);
(Union of the residue classes 0(2) and 3(6) of Z) U [ 1, 5, 7 ]
gap> Intersection(S,[1..6]);
[ 2, 3, 4, 6 ]
gap> Difference(S,[1..6]);
(Union of the residue classes 0(2) and 3(6) of Z) \ [ 2, 3, 4, 6 ]
gap> Difference(Integers,[1..10]);
Z \ <set of cardinality 10>
gap> IsSubset(S,[1..10]);
false
```

If the underlying ring has a residue class ring of a given cardinality t, then a residue class can be written as a disjoint union of t residue classes with equal moduli:

1.2.1 SplittedClass (for a residue class and a number of parts)

▷ `SplittedClass(cl, t)` (operation)

Returns: a partition of the residue class `cl` into `t` residue classes with equal moduli, provided that such a partition exists. Otherwise `fail`.

Example

```
gap> SplittedClass(ResidueClass(1,2),2);
[ The residue class 1(4) of Z, The residue class 3(4) of Z ]
gap> SplittedClass(ResidueClass(Z_pi(3),3,0),2);
fail
```

Often one needs a partition of a given residue class union into "few" residue classes. The following operation takes care of this:

1.2.2 AsUnionOfFewClasses (for a residue class union)

▷ `AsUnionOfFewClasses(U)` (operation)

Returns: a set of disjoint residue classes whose union is equal to `U`, up to the finite sets `IncludedElements(U)` and `ExcludedElements(U)`.

As the name of the operation suggests, it is taken care that the number of residue classes in the returned list is kept "reasonably small". It is not guaranteed that it is minimal.

Example

```
gap> ResidueClassUnionViewingFormat("short");
gap> AsUnionOfFewClasses(Difference(Integers,ResidueClass(0,30)));
[ 1(2), 2(6), 4(6), 6(30), 12(30), 18(30), 24(30) ]
gap> Union(last);
Z \ 0(30)
```

One can compute the sets of sums, differences, products and quotients of the elements of a residue class union and an element of the base ring:

Example

```
gap> ResidueClass(0,2) + 1;
1(2)
gap> ResidueClass(0,2) - 2 = ResidueClass(0,2);
true
gap> 3 * ResidueClass(0,2);
0(6)
gap> ResidueClass(0,2)/2;
Integers
```

1.2.3 PartitionsIntoResidueClasses (of a given ring, of given length)

▷ `PartitionsIntoResidueClasses(R, length)` (operation)
▷ `PartitionsIntoResidueClasses(R, length, primes)` (operation)

Returns: in the 2-argument version a sorted list of all partitions of the ring *R* into *length* residue classes. In the 3-argument version a sorted list of all partitions of the ring *R* into *length* residue classes whose moduli have only prime factors in the list *primes*.

Example

```
gap> PartitionsIntoResidueClasses(Integers,4);
[ [ 0(2), 1(4), 3(8), 7(8) ], [ 0(2), 3(4), 1(8), 5(8) ],
  [ 0(2), 1(6), 3(6), 5(6) ], [ 1(2), 0(4), 2(8), 6(8) ],
  [ 1(2), 2(4), 0(8), 4(8) ], [ 1(2), 0(6), 2(6), 4(6) ],
  [ 0(3), 1(3), 2(6), 5(6) ], [ 0(3), 2(3), 1(6), 4(6) ],
  [ 1(3), 2(3), 0(6), 3(6) ], [ 0(4), 1(4), 2(4), 3(4) ] ]
```

1.2.4 RandomPartitionIntoResidueClasses (of a given ring, of given length)

▷ `RandomPartitionIntoResidueClasses(R, length, primes)` (operation)

Returns: a “random” partition of the ring *R* into *length* residue classes whose moduli have only prime factors in *primes*, respectively `fail` if no such partition exists.

Example

```
gap> RandomPartitionIntoResidueClasses(Integers,30,[2,3,5,7]);
[ 0(7), 2(7), 5(7), 3(14), 10(14), 1(21), 8(21), 15(21), 18(21), 20(21),
  6(63), 13(63), 25(63), 27(63), 32(63), 34(63), 46(63), 48(63), 53(63),
  55(63), 4(126), 67(126), 137(189), 74(567), 200(567), 263(567),
  389(567), 452(567), 11(1134), 578(1134) ]
gap> Union(last);
Integers
gap> Sum(List(last2,Density));
1
```

1.2.5 CoverByResidueClasses (of the integers, by residue classes with given moduli)

▷ `CoverByResidueClasses(Integers, moduli)` (method)
▷ `CoversByResidueClasses(Integers, moduli)` (method)

Returns: in the first form a cover of the integers by residue classes with moduli *moduli* if such cover exists, and `fail` otherwise; in the second form a list of all covers of the integers by residue classes with moduli *moduli*.

Since there are often very many such covers, computing all of them can take a lot of time and memory.

Example

```
gap> CoverByResidueClasses(Integers,[2,3,4,6,8,12]);
[ 0(2), 0(3), 1(4), 1(6), 3(8), 11(12) ]
gap> Union(last);
Integers
```

```
gap> CoversByResidueClasses(Integers,[2,3,3,6]);
[ [ 0(2), 0(3), 1(3), 5(6) ], [ 0(2), 0(3), 2(3), 1(6) ],
  [ 0(2), 1(3), 2(3), 3(6) ], [ 1(2), 0(3), 1(3), 2(6) ],
  [ 1(2), 0(3), 2(3), 4(6) ], [ 1(2), 1(3), 2(3), 0(6) ] ]
gap> List(last,Union);
[ Integers, Integers, Integers, Integers, Integers, Integers ]
```

1.2.6 Density (of a residue class union)

▷ `Density(U)` (operation)

Returns: the natural density of `U` as a subset of the underlying ring.

The *natural density* of a residue class $r(m)$ of a ring R is defined by $1/|R/mR|$, and the *natural density* of a union U of finitely many residue classes is defined by the sum of the densities of the elements of a partition of U into finitely many residue classes.

Example

```
gap> Density(ResidueClass(0,2));
1/2
gap> Density(Difference(Integers,ResidueClass(0,5)));
4/5
```

For looping over residue class unions of the integers, there are methods for the operations `Iterator` and `NextIterator`.

1.3 On residue class unions of $\mathbb{Z}^2$

Residue class unions of $\mathbb{Z}^2$ are treated similar as those of any other ring. Also there is roughly the same functionality available for them. However there are some differences and a few additional features, which are described in this section.

The elements of $\mathbb{Z}^2$ are represented as lists of length 2 with integer entries. The modulus of a residue class union of $\mathbb{Z}^2$ is a lattice. This lattice is stored as a 2×2 integer matrix of full rank in Hermite normal form, whose rows are the spanning vectors. Residue classes of $\mathbb{Z}^2$ modulo principal ideals are presently not implemented. Residue class unions of $\mathbb{Z}^2$ can be multiplied by matrices of full rank from the right. A snippet of a residue class union of $\mathbb{Z}^2$ is shown in "ASCII art" when one `Display`'s it with option `AsGrid`. We give some illustrative examples:

Example

```
gap> R := Integers^2;
( Integers^2 )
gap> 5*R+[2,3];
(2,3)+(5,0)Z+(0,5)Z
gap> Difference(R,last);
Z^2 \ (2,3)+(5,0)Z+(0,5)Z
gap> Density(last);
24/25
gap> L1 := [[2,1],[-1,2]];;
gap> L2 := [[6,2],[0,6]];;
```

```
gap> AllResidueClassesModulo(R,L1); # The modulus is transformed to HNF.
[ (0,0)+(1,3)Z+(0,5)Z, (0,1)+(1,3)Z+(0,5)Z, (0,2)+(1,3)Z+(0,5)Z,
  (0,3)+(1,3)Z+(0,5)Z, (0,4)+(1,3)Z+(0,5)Z ]
gap> cl1 := ResidueClass(R,L1,[0,0]);
(0,0)+(1,3)Z+(0,5)Z
gap> cl2 := ResidueClass(R,L2,[0,0]);
(0,0)+(6,2)Z+(0,6)Z
gap> cl3 := Intersection(cl1,cl2);
(0,0)+(6,8)Z+(0,30)Z
gap> S1 := Difference(cl1,cl2);
<union of 35 residue classes (mod (6,8)Z+(0,30)Z)>
gap> S2 := Difference(cl2,cl1);
<union of 4 residue classes (mod (6,8)Z+(0,30)Z)>
gap> Display(S1); # The set is written as union of "few" residue classes:
(0,5)+(1,3)Z+(0,10)Z U (1,3)+(2,6)Z+(0,10)Z U (2,6)+(6,8)Z+(0,10)Z U
(4,2)+(6,8)Z+(0,10)Z U (0,10)+(6,8)Z+(0,30)Z U (0,20)+(6,8)Z+(0,30)Z
gap> Display(S2);
(0,6)+(6,8)Z+(0,30)Z U (0,12)+(6,8)Z+(0,30)Z U (0,18)+(6,8)Z+(0,30)Z
 U (0,24)+(6,8)Z+(0,30)Z
gap> cls := AsUnionOfFewClasses(S1);
[ (0,5)+(1,3)Z+(0,10)Z, (1,3)+(2,6)Z+(0,10)Z, (2,6)+(6,8)Z+(0,10)Z,
  (4,2)+(6,8)Z+(0,10)Z, (0,10)+(6,8)Z+(0,30)Z, (0,20)+(6,8)Z+(0,30)Z ]
gap> Union(cls) = S1;
true
gap> S3 := S1*[[3,5],[2,4]];
<union of 35 residue classes (mod (2,46)Z+(0,180)Z)>
gap> Display(S1:AsGrid);
    *    *    *    *    *    *    *    *    *    *    *    *    *    *
 *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
   *    *    *    *    *    *    *    *    *    *    *    *    *    *
*    *    *    *    *    *    *    *    *    *    *    *    *    *    *
  *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
    *    *    *    *         *    *    *    *    *         *    *    *
 *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
   *    *    *    *    *    *    *    *    *    *    *    *    *    *
*    *    *    *    *    *    *    *    *    *    *    *    *    *    *
  *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
    *    *    *    *    *    *    *    *    *    *    *    *    *    *
 *    *    *         *    *    *    *    *         *    *    *    *    *
   *    *    *    *    *    *    *    *    *    *    *    *    *    *
*    *    *    *    *    *    *    *    *    *    *    *    *    *    *
  *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
    *    *    *    *    *    *    *    *    *    *    *    *    *    *
 *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
   *         *    *    *    *    *         *    *    *    *    *
*    *    *    *    *    *    *    *    *    *    *    *    *    *    *
  *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
    *    *    *    *    *    *    *    *    *    *    *    *    *    *
 *    *    *    *    *    *    *    *    *    *    *    *    *    *    *
   *    *    *    *    *    *    *    *    *    *    *    *    *    *
     *    *    *    *    *         *    *    *    *    *         *    *
```

Note that in GAP multiplying lists of integers means computing their scalar product as vectors. The consequence is that technically the free module $\mathbb{Z}^2$ is not a ring in GAP.

1.4 The categories and families of residue class unions

1.4.1 IsResidueClassUnion

▷ `IsResidueClassUnion(U)` (filter)
▷ `IsResidueClassUnionOfZ(U)` (filter)
▷ `IsResidueClassUnionOfZxZ(U)` (filter)
▷ `IsResidueClassUnionOfZ_pi(U)` (filter)
▷ `IsResidueClassUnionOfGFqx(U)` (filter)

Returns: `true` if *U* is a residue class union, a residue class union of $\mathbb{Z}$, a residue class union of $\mathbb{Z}^2$, a residue class union of a semilocalization of $\mathbb{Z}$ or a residue class union of a polynomial ring in one variable over a finite field, respectively, and `false` otherwise.

Often the same methods can be used for residue class unions of the ring of integers and of its semilocalizations. For this reason, there is a category `IsResidueClassUnionOfZorZ_pi` which is the union of `IsResidueClassUnionOfZ` and `IsResidueClassUnionOfZ_pi`. The internal representation of residue class unions is called `IsResidueClassUnionResidueListRep`. There are methods available for `ExtRepOfObj` and `ObjByExtRep`.

1.4.2 ResidueClassUnionsFamily (of a ring)

▷ `ResidueClassUnionsFamily(R)` (function)
▷ `ResidueClassUnionsFamily(R, fixedreps)` (function)

Returns: the family of residue class unions or the family of unions of residue classes with fixed representatives of the ring *R*, depending on whether *fixedreps* is present and `true` or not.

The ring *R* can be retrieved as `UnderlyingRing(ResidueClassUnionsFamily(R))`. There is no coercion between residue class unions or unions of residue classes with fixed representatives which belong to different families. Unions of residue classes with fixed representatives are described in the next chapter.

Chapter 2

Unions of Residue Classes with Fixed Representatives

ResClasses supports computations with unions of residue classes which are endowed with distinguished ("fixed") representatives. These unions of residue classes can be viewed as multisets of ring elements. The residue classes forming such a union do not need to be disjoint or even only distinct.

2.1 Entering unions of residue classes with fixed representatives

2.1.1 ResidueClassWithFixedRepresentative (by ring, modulus and residue)

▷ `ResidueClassWithFixedRepresentative(R, m, r)` (function)
▷ `ResidueClassWithFixedRepresentative(m, r)` (function)

Returns: the residue class *r* mod *m* of the ring *R*, with the fixed representative *r*.

If the argument *R* is omitted, it defaults to `Integers`. Residue classes with fixed representatives have the property `IsResidueClassWithFixedRepresentative`. The fixed representative *r* can be retrieved by the operation `Residue`, and the modulus *m* can be retrieved by the operation `Modulus`.

Example

```
gap> ResidueClassWithFixedRepresentative(Integers,2,1);
[1/2]
```

2.1.2 UnionOfResidueClassesWithFixedReps (by ring and list of classes)

▷ `UnionOfResidueClassesWithFixedReps(R, classes)` (function)
▷ `UnionOfResidueClassesWithFixedReps(classes)` (function)

Returns: the union of the residue classes *classes*[i][2] mod *classes*[i][1] of the ring *R*, with fixed representatives *classes*[i][2].

The argument *classes* must be a list of pairs of elements of the ring *R*. Their first entries – the moduli – must be nonzero. If the argument *R* is omitted, it defaults to `Integers`.

Example

```
gap> UnionOfResidueClassesWithFixedReps(Integers,[[2,4],[3,9]]);
[4/2] U [9/3]
```

There is a method for the operation `Modulus` which returns the lcm of the moduli of the residue classes forming such a union. Further there is an operation `Classes` for retrieving the list of classes which has been passed as an argument to `UnionOfResidueClassesWithFixedReps`. The operation `AsListOfClasses` does the same, except that the returned list contains residue classes instead of pairs `[modulus,residue]`. There are methods for `Print`, `String` and `Display` available for unions of residue classes with fixed representatives.

2.1.3 AllResidueClassesWithFixedRepsModulo (by ring and modulus)

▷ `AllResidueClassesWithFixedRepsModulo(R, m)` (function)
▷ `AllResidueClassesWithFixedRepsModulo(m)` (function)

Returns: a sorted list of all residue classes (mod `m`) of the ring `R`, with fixed representatives.

If the argument `R` is omitted it defaults to the default ring of `m`, cf. the documentation of `DefaultRing` in the GAP reference manual. The representatives are the same as those chosen by the operation `mod`. See also `AllResidueClassesModulo` (1.1.3).

Example

```
gap> AllResidueClassesWithFixedRepsModulo(4);
[ [0/4], [1/4], [2/4], [3/4] ]
```

2.2 Methods for unions of residue classes with fixed representatives

Throughout this chapter, the argument `R` denotes the underlying ring, and the arguments `U`, `U1` and `U2` denote unions of residue classes of `R` with fixed representatives.

Unions of residue classes with fixed representatives are multisets. Elements and residue classes can be contained with multiplicities:

2.2.1 Multiplicity (of an element in a union of residue classes with fixed rep's)

▷ `Multiplicity(x, U)` (method)
▷ `Multiplicity(cl, U)` (method)

Returns: the multiplicity of `x` in `U` regarded as a multiset of ring elements, resp. the multiplicity of the residue class `cl` in `U` regarded as a multiset of residue classes.

Example

```
gap> U := UnionOfResidueClassesWithFixedReps(Integers,[[2,0],[3,0]]);
[0/2] U [0/3]
gap> List([0..20],n->Multiplicity(n,U));
[ 2, 0, 1, 1, 1, 0, 2, 0, 1, 1, 1, 0, 2, 0, 1, 1, 1, 0, 2, 0, 1 ]
gap> Multiplicity(ResidueClassWithFixedRep(2,0),U);
```

```
1
```

Let U be a union of residue classes with fixed representatives. The multiset U can have an attribute Density which denotes its *natural density* as a multiset, i.e. elements with multiplicity k count k-fold. The multiset U has the property IsOverlappingFree if it consists of pairwise disjoint residue classes. The set-theoretic union of the residue classes forming U can be determined by the operation AsOrdinaryUnionOfResidueClasses. The object returned by this operation is an “ordinary” residue class union as described in Chapter 1.

Example

```
gap> U := UnionOfResidueClassesWithFixedReps(Integers,[[2,0],[3,0]]);
[0/2] U [0/3]
gap> Density(U);
5/6
gap> IsOverlappingFree(U);
false
gap> AsOrdinaryUnionOfResidueClasses(U);
Z \ 1(6) U 5(6)
gap> Density(last);
2/3
```

In the sequel we abbreviate the term “the multiset of ring elements endowed with the structure of a union of residue classes with fixed representatives” by “the multiset”.

There are methods for + and - available for computing the multiset of sums $u+x$, $u \in U$, the multiset of differences $u-x$ resp. $x-u$, $u \in U$ and the multiset of the additive inverses of the elements of U. Further there are methods for * and / available for computing the multiset of products $x \cdot u$, $u \in U$ and the multiset of quotients u/x, $u \in U$. The division method requires all elements of U to be divisible by x. If the underlying ring is the ring of integers, scalar multiplication and division leave δ invariant ($\rightarrow$ Delta (2.3.1)).

Example

```
gap> U := UnionOfResidueClassesWithFixedReps(Integers,[[2,0],[3,0]]);
[0/2] U [0/3]
gap> U + 7;
[7/2] U [7/3]
gap> U - 7; 7 - U; -U;
[-7/2] U [-7/3]
[7/-3] U [7/-2]
[0/-3] U [0/-2]
gap> V := 2 * U;
[0/4] U [0/6]
gap> V/2;
[0/2] U [0/3]
```

2.2.2 Union (for unions of residue classes with fixed representatives)

▷ `Union(U1, U2)` (method)

Returns: the union of *U1* and *U2*.

The multiplicity of any ring element or residue class in the union is the sum of its multiplicities in the arguments. It holds that `Delta(Union(U1,U2)) = Delta(U1) + Delta(U2)`. (→ `Delta` (2.3.1)).

Example

```
gap> U := UnionOfResidueClassesWithFixedReps(Integers,[[2,0],[3,0]]);
[0/2] U [0/3]
gap> Union(U,U);
[0/2] U [0/2] U [0/3] U [0/3]
```

2.2.3 Intersection (for unions of residue classes with fixed representatives)

▷ `Intersection(U1, U2)` (method)

Returns: the intersection of *U1* and *U2*.

The multiplicity of any residue class in the intersection is the minimum of its multiplicities in the arguments.

Example

```
gap> U := UnionOfResidueClassesWithFixedReps(Integers,[[2,0],[3,0]]);
[0/2] U [0/3]
gap> Intersection(U,ResidueClassWithFixedRep(2,0));
[0/2]
gap> Intersection(U,ResidueClassWithFixedRep(6,0));
[]
```

2.2.4 Difference (for unions of residue classes with fixed representatives)

▷ `Difference(U1, U2)` (method)

Returns: the difference of *U1* and *U2*.

The multiplicity of any residue class in the difference is its multiplicity in *U1* minus its multiplicity in *U2*, if this value is nonnegative. The difference of the empty residue class union with fixed representatives and some residue class $[r/m]$ is set equal to $[(m-r)/m]$. It holds that `Delta(Difference(U1,U2)) = Delta(U1) - Delta(U2)`. (→ `Delta` (2.3.1)).

Example

```
gap> U := UnionOfResidueClassesWithFixedReps(Integers,[[2,0],[3,0]]);
[0/2] U [0/3]
gap> V := UnionOfResidueClassesWithFixedReps(Integers,[[3,0],[5,2]]);
[0/3] U [2/5]
gap> Difference(U,V);
[0/2] U [3/5]
```

2.3 The invariant Delta

2.3.1 Delta (for a union of residue classes with fixed representatives)

▷ `Delta(U)` (attribute)

Returns: the value of the invariant δ of the residue class union `U`.

For a residue class $[r/m]$ with fixed representative we set $\delta([r/m]) := r/m - 1/2$, and extend this definition additively to unions of such residue classes. If no representatives are fixed, this definition is still unique (mod 1). There is a related invariant ρ which is defined by $e^{\delta(U)\pi i}$. The corresponding attribute is called `Rho`.

Example

```
gap> U := UnionOfResidueClassesWithFixedReps(Integers,[[2,3],[3,4]]);
[3/2] U [4/3]
gap> Delta(U) = (3/2-1/2) + (4/3-1/2);
true
gap> V := RepresentativeStabilizingRefinement(U,3);
[3/6] U [5/6] U [7/6] U [4/9] U [7/9] U [10/9]
gap> Delta(V) = Delta(U);
true
gap> Rho(V);
E(12)^11
```

2.3.2 RepresentativeStabilizingRefinement (of a union of res.-classes with fixed rep's)

▷ `RepresentativeStabilizingRefinement(U, k)` (method)

Returns: the representative stabilizing refinement of `U` into `k` parts.

The *representative stabilizing refinement* of a residue class $[r/m]$ of $\mathbb{Z}$ into k parts is defined by $[r/km] \cup [(r+m)/km] \cup \ldots \cup [(r+(k-1)m)/km]$. This definition is extended in the obvious way to unions of residue classes.

If the argument `k` is zero, the method performs a simplification of `U` by joining appropriate residue classes, if this is possible.

In any case the value of `Delta(U)` is invariant under this operation (→ `Delta` (2.3.1)).

Example

```
gap> U := UnionOfResidueClassesWithFixedReps(Integers,[[2,0],[3,0]]);
[0/2] U [0/3]
gap> RepresentativeStabilizingRefinement(U,4);
[0/8] U [2/8] U [4/8] U [6/8] U [0/12] U [3/12] U [6/12] U [9/12]
gap> RepresentativeStabilizingRefinement(last,0);
[0/2] U [0/3]
```

2.4 The categories of unions of residue classes with fixed rep's

The names of the categories of unions of residue classes with fixed representatives are `IsUnionOfResidueClassesOf[Z|Z_pi|ZorZ_pi|GFqx]WithFixedRepresentatives`.

Chapter 3

Semilocalizations of the Integers

This package implements residue class unions of the semilocalizations $\mathbb{Z}_{(\pi)}$ of the ring of integers. It also provides the underlying GAP implementation of these rings themselves.

3.1 Entering semilocalizations of the integers

3.1.1 Z_pi (by set of non-invertible primes)

▷ Z_pi(*pi*) (function)
▷ Z_pi(*p*) (function)

Returns: the ring $\mathbb{Z}_{(\pi)}$ or the ring $\mathbb{Z}_{(p)}$, respectively.

The returned ring has the property IsZ_pi. The set *pi* of non-invertible primes can be retrieved by the operation NoninvertiblePrimes.

Example

```
gap> R := Z_pi(2);
Z_( 2 )
gap> S := Z_pi([2,5,7]);
Z_( 2, 5, 7 )
```

3.2 Methods for semilocalizations of the integers

There are methods for the operations in, Intersection, IsSubset, StandardAssociate, Gcd, Lcm, Factors and IsUnit available for semilocalizations of the integers. For the documentation of these operations, see the GAP reference manual. The standard associate of an element of a ring $\mathbb{Z}_{(\pi)}$ is defined by the product of the non-invertible prime factors of its numerator.

Example

```
gap> 4/7 in R; 3/2 in R;
true
false
gap> Intersection(R,Z_pi([3,11])); IsSubset(R,S);
Z_( 2, 3, 11 )
true
```

Example

```
gap> StandardAssociate(R,-6/7);
2
gap> Gcd(S,90/3,60/17,120/33);
10
gap> Lcm(S,90/3,60/17,120/33);
40
gap> Factors(R,840);
[ 105, 2, 2, 2 ]
gap> Factors(R,-2/3);
[ -1/3, 2 ]
gap> IsUnit(S,3/11);
true
```

Chapter 4

Installation and Auxiliary Functions

4.1 Requirements

This version of ResClasses needs at least GAP 4.9.0, Polycyclic 2.11 [EHN13], GAPDoc 1.5.1 [LN12] and Utils 0.40 [GKW16]. It can be used on all platforms for which GAP is available. ResClasses is completely written in the GAP language and does neither contain nor require external binaries.

4.2 Installation

Like any other GAP package, ResClasses is usually installed in the `pkg` subdirectory of the GAP distribution. This is accomplished by extracting the distribution file in this directory. By default, the package ResClasses is autoloaded. If you have switched autoloading of packages off, you can load ResClasses via `LoadPackage( "resclasses" );`.

4.3 The testing routines

4.3.1 ResClassesTest

▷ `ResClassesTest()` (function)

Returns: `true` if no errors were found, and `false` otherwise.

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

4.3.2 ResClassesTestExamples

▷ `ResClassesTestExamples()` (function)

Returns: nothing.

Runs all examples in the manual of the ResClasses package, and reports any differences between the actual output and the output printed in the manual.

4.4 Creating timestamped logfiles

4.4.1 LogToDatedFile

▷ `LogToDatedFile(directory)` (function)

Returns: the full pathname of the created logfile.

This function opens a logfile in the specified directory; the name of the logfile has the form of a timestamp, i.e. `year-month-day hour-minute-second.log`. If GAP is already in logging mode, the old logfile is closed before the new one is opened.

The availability of this function depends on that the package IO [HN16] is installed and compiled.

4.5 DownloadFile, SendEmail and EmailLogFile

4.5.1 DownloadFile

▷ `DownloadFile(url)` (function)

Returns: the contents of the file with URL *url* in the form of a string if that file exists and the download was successful, and `fail` otherwise.

As most system-related functions, `DownloadFile` works only under UNIX / Linux. Also the computer must of course be connected to the Internet.

4.5.2 SendEmail

▷ `SendEmail(sendto, copyto, subject, text)` (function)

Returns: zero if everything worked correctly, and a system error number otherwise.

Sends an e-mail with subject *subject* and body *text* to the addresses in the list *sendto*, and copies it to those in the list *copyto*. The first two arguments must be lists of strings, and the latter two must be strings.

As most system-related functions, `SendEmail` works only under UNIX / Linux. Also the computer must of course be connected to the Internet.

4.5.3 EmailLogFile

▷ `EmailLogFile(addresses)` (function)

Returns: zero if everything worked correctly, and a system error number otherwise.

Sends the current log file by e-mail to *addresses*, if GAP is in logging mode and one is working under UNIX / Linux, and does nothing otherwise. The argument *addresses* must be either a list of e-mail addresses or a single e-mail address. Long log files are abbreviated, i.e. if the log file is larger than 64KB, then any output is truncated at 1KB, and if the log file is still longer than 64KB afterwards, it is truncated at 64KB.

4.6 Creating bitmap pictures

ResClasses provides functions to generate bitmap picture files from suitable pixel matrices and vice versa. The author has successfully tested this feature both under Linux and under Windows, and the generated pictures can be processed further with many common graphics programs:

4.6.1 SaveAsBitmapPicture (picture, filename)

▷ `SaveAsBitmapPicture(picture, filename)` (function)

Returns: nothing.

Writes the pixel matrix *picture* to a bitmap- (bmp-) picture file named *filename*. The filename should include the entire pathname. The argument *picture* can be a GF(2) matrix, in which case a monochrome picture file is generated. In this case, zeros stand for black pixels and ones stand for white pixels. The argument *picture* can also be an integer matrix, in which case a 24-bit true color picture file is generated. In this case, the entries of the matrix are supposed to be integers $n = 65536 \cdot red + 256 \cdot green + blue$ in the range $0, \dots, 2^{24} - 1$ specifying the RGB values of the colors of the pixels.

The picture can be read back into GAP by the function `LoadBitmapPicture(filename)`.

Example

```
gap> color   := n->32*(n mod 8)+256*32*(Int(n/8) mod 8)+65536*32*Int(n/64);;
gap> picture := List([1..512],y->List([1..512],x->color(Gcd(x,y)-1)));;
gap> SaveAsBitmapPicture(picture,Filename(DirectoryTemporary(),"gcd.bmp"));
```

4.6.2 DrawLineNC (pic, x1, y1, x2, y2, color, width)

▷ `DrawLineNC(pic, x1, y1, x2, y2, color, width)` (function)

Returns: nothing.

Draws a line on picture *pic* from (*x1*,*y1*) to (*x2*,*y2*), with color *color* and of width *width*.

Example

```
gap> picture := NullMat(100,100)+2^24-1;;
gap> DrawLineNC(picture,30,20,70,80,255,8);
gap> SaveAsBitmapPicture(picture,Filename(DirectoryTemporary(),
>                        "example.bmp"));
```

4.7 Some general utility functions

ResClasses provides a few small utility functions and -operations which can be used in a more general context. They are described in this section.

There is an operation `PositionsSublist(list,sub)` which returns the list of positions at which *sub* occurs as a sublist of *list*.

Example

```
gap> PositionsSublist([1,2,6,2,7,2,7,2,3,1,6,2,7,2,8],[2,7,2]);
[ 4, 6, 12 ]
gap> PositionsSublist([1,2,3,4,3,2,1],[1,3,5]);
[  ]
gap> PositionsSublist("This is an example, isn't it?","is");
[ 3, 6, 21 ]
```

Also there are methods `EquivalenceClasses(l,inv)` and `EquivalenceClasses(l,rel)` which decompose a list `l` into equivalence classes under an equivalence relation. The equivalence relation is given either as a function `inv` computing a class invariant of a given list entry or as a function `rel` which takes as arguments two list entries and returns either `true` or `false` depending on whether the arguments belong to the same equivalence class or not.

Example

```
gap> EquivalenceClasses([2..50],n->Length(Factors(n)));
[ [ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47 ],
  [ 4, 6, 9, 10, 14, 15, 21, 22, 25, 26, 33, 34, 35, 38, 39, 46, 49 ],
  [ 8, 12, 18, 20, 27, 28, 30, 42, 44, 45, 50 ], [ 16, 24, 36, 40 ],
  [ 32, 48 ] ]
gap> EquivalenceClasses(AsList(AlternatingGroup(4)),
>                       function ( g, h )
>                         return IsConjugate(SymmetricGroup(4),g,h);
>                       end);
[ [ (2,3,4), (2,4,3), (1,2,3), (1,2,4), (1,3,2), (1,3,4), (1,4,2),
      (1,4,3) ], [ (1,2)(3,4), (1,3)(2,4), (1,4)(2,3) ], [ () ] ]
```

Further, there is an operation `GraphClasses(n)` which returns a list of isomorphism classes of graphs with vertices $1,2,\dots,n$, and an operation `AllGraphs(n)` which returns a list of representatives of these classes. The graphs are represented as lists of edges, where each edge is a list of the two vertices it connects, and they are ordered by ascending number of edges. Given a graph *graph* with *n* vertices, the operation `IdGraphNC(graph,GraphClasses(n))` returns the index `i` such that *graph* lies in `GraphClasses(n)[i]`. For reasons of efficiency, `IdGraphNC` performs no argument checks.

Example

```
gap> GraphClasses(3);
[ [ [  ] ], [ [ [ 1, 2 ] ], [ [ 2, 3 ] ], [ [ 1, 3 ] ] ],
  [ [ [ 1, 2 ], [ 1, 3 ] ], [ [ 1, 2 ], [ 2, 3 ] ],
      [ [ 1, 3 ], [ 2, 3 ] ] ], [ [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ] ] ] ]
gap> List(last,Length); # sizes of classes
[ 1, 3, 3, 1 ]
gap> AllGraphs(4);
[ [  ], [ [ 1, 2 ] ], [ [ 1, 2 ], [ 1, 3 ] ], [ [ 1, 2 ], [ 3, 4 ] ],
  [ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ] ], [ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ] ],
  [ [ 1, 2 ], [ 1, 3 ], [ 2, 4 ] ],
  [ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 2, 3 ] ],
  [ [ 1, 2 ], [ 1, 3 ], [ 2, 4 ], [ 3, 4 ] ],
  [ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 2, 3 ], [ 2, 4 ] ],
  [ [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 2, 3 ], [ 2, 4 ], [ 3, 4 ] ] ]
gap> List(last,Length); # numbers of edges
[ 0, 1, 2, 2, 3, 3, 3, 4, 4, 5, 6 ]
gap> IdGraphNC([[1,3],[1,8],[3,8]],GraphClasses(4)); # a triangle graph
6
gap> AllGraphs(4)[last];
[ [ 1, 2 ], [ 1, 3 ], [ 2, 3 ] ]
```

References

[EHN13] B. Eick, M. Horn, and W. Nickel. *Polycyclic – Computation with polycyclic groups (Version 2.11)*, 2013. GAP package, https://www.gap-system.org/Packages/polycyclic.html. 19

[GKW16] S. Gutsche, S. Kohl, and C. Wensley. *Utils - Utility functions in GAP (Version 0.38)*, 2016. GAP package, https://www.gap-system.org/Packages/utils.html. 19

[HN16] M. Horn and M. Neunhöffer. *IO – Bindings for low level C library I/O routines (Version 4.4.5)*, 2016. GAP package, https://www.gap-system.org/Packages/io.html. 20

[Koh16] S. Kohl. *RCWA - Residue-Class-Wise Affine Groups (Version 4.4.0)*, 2016. GAP package, https://www.gap-system.org/Packages/rcwa.html. 2

[LN12] F. Lübeck and M. Neunhöffer. *GAPDoc (Version 1.5.1)*. RWTH Aachen, 2012. GAP package, https://www.gap-system.org/Packages/gapdoc.html. 19

Index