numericalsgps– a package for numerical semigroups

Version 1.1.8 dev

Manuel Delgado
Pedro A. García-Sánchez
José João Morais

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

Pedro A. García-Sánchez Email: pedro@ugr.es
Homepage: http://www.ugr.es/~pedro

Copyright

Acknowledgements

The first author's work was (partially) supported by the *Centro de Matemática da Universidade do Porto* (CMUP), financed by FCT (Portugal) through the programs POCTI (Programa Operacional "Ciência, Tecnologia, Inovação") and POSI (Programa Operacional Sociedade da Informação), with national and European Community structural funds and a sabbatical grant of FCT.

The second author was supported by the projects MTM2004-01446, MTM2007-62346 and MTM2010-15595, the Junta de Andalucía group FQM-343, and FEDER founds.

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

The authors wish to thank J. I. García-García and Alfredo Sánchez-R. Navarro for many helpful discussions and for helping in the programming of preliminary versions of some functions, and also to C. O'Neill, A. Sammartano, I. Ojeda, C. M. Moreno Ávila, A. Herrera-Poyatos and K. Stokes for their contributions (see Contributions Chapter). We are also in debt with S. Gutsche, M. Horn, H. Schönemann, C. Söeger and M. Barakat for their fruitful advices concerning 4ti2Interface, SingularInterface, Singular, Normaliz, NormalizInterface and GradedModules.

The first and second authors warmly thank María Burgos for her support and help.

CONCERNING THE MANTAINMENT:

The first author was (partially) supported by the FCT project PTDC/MAT/65481/2006 and also by the *Centro de Matemática da Universidade do Porto* (CMUP), funded by the European Regional Development Fund through the programme COMPETE and by the Portuguese Government through the FCT - Fundação para a Ciência e a Tecnologia under the project PEst-C/MAT/UI0144/2011.

The second author was/is supported by the project MTM2014-55367-P, which is funded by Ministerio de Economía y Competitividad and the Fondo Europeo de Desarrollo Regional FEDER.

Both maintainers want to acknowledge partial support by CMUP (UID/MAT/00144/2013), which is funded by FCT (Portugal) with national (MEC) and European structural funds through the programs FEDER, under the partnership agreement PT2020.

Both maintainers are also supported by the project MTM2014-55367-P, which is funded by Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional FEDER.

The maintainers want to thank the organizers of `GAPDays` in their several editions.

The authors also thank the Centro de Servicios de Informática y Redes de Comunicaciones (CSIRC), Universidad de Granada, for providing the computing time, specially Rafael Arco Arredondo for installing this package and the extra software needed in alhambra.ugr.es.

Colophon

This work started when (in 2004) the first author visited the University of Granada in part of a sabbatical year. Since Version 0.96 (released in 2008), the package is maintained by the first two authors. Bug reports, suggestions and comments are, of course, welcome. Please use our email addresses to this effect.

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

If you have predominantly used the functions in the Appendix, contributed by other authors, please cite in addition these authors, referring "software implementations available in the GAP package NumericalSgps".

Contents

Chapter 1

Introduction

A *numerical semigroup* is a subset of the set $\mathbb{N}$ of nonnegative integers that is closed under addition, contains 0 and whose complement in $\mathbb{N}$ is finite. The smallest positive integer belonging to a numerical semigroup is its *multiplicity*.

Let S be a numerical semigroup and A be a subset of S. We say that A is a *system of generators* of S if $S = \{k_1a_1 + \cdots + k_na_n \mid n, k_1, \ldots, k_n \in \mathbb{N}, a_1, \ldots, a_n \in A\}$. The set A is a *minimal system of generators* of S if no proper subset of A is a system of generators of S.

Every numerical semigroup has a unique minimal system of generators. This is a data that can be used in order to uniquely define a numerical semigroup. Observe that since the complement of a numerical semigroup in the set of nonnegative integers is finite, this implies that the greatest common divisor of the elements of a numerical semigroup is 1, and the same condition must be fulfilled by its minimal system of generators (or by any of its systems of generators).

Given a numerical semigroup S and a nonzero element s in it, one can consider for every integer i ranging from 0 to $s-1$, the smallest element in S congruent with i modulo s, say $w(i)$ (this element exists since the complement of S in $\mathbb{N}$ is finite). Clearly $w(0) = 0$. The set $\mathrm{Ap}(S,s) = \{w(0), w(1), \ldots, w(s-1)\}$ is called the *Apéry set* of S with respect to s. Note that a nonnegative integer x congruent with i modulo s belongs to S if and only if $w(i) \leq x$. Thus the pair $(s, \mathrm{Ap}(S,s))$ fully determines the numerical semigroup S (and can be used to easily solve the membership problem to S). This set is in fact one of the most powerfull tools known for numerical semigroups, and it is used almost everywhere in the computation of components and invariants associated to a numerical semigroup. Usually the element s is taken to be the multiplicity, since in this way the resulting Apéry set is the smallest possible.

A *gap* of a numerical semigroup S is a nonnegative integer not belonging to S. The set of gaps of S is usually denoted by $\mathrm{H}(S)$, and clearly determines uniquely S. Note that if x is a gap of S, then so are all the nonnegative integers dividing it. Thus in order to describe S we do not need to know all its gaps, but only those that are maximal with respect to the partial order induced by division in $\mathbb{N}$. These gaps are called *fundamental gaps*.

The largest nonnegative integer not belonging to a numerical semigroup S is the *Frobenius number* of S. If S is the set of nonnegative integers, then clearly its Frobenius number is -1, otherwise its Frobenius number coincides with the maximum of the gaps (or fundamental gaps) of S. The Frobenius number plus one is known as the *conductor* of the semigroup. In this package we refer to the elements in the semigroup that are less than or equal to the conductor as *small elements* of the semigroup. Observe that from the definition, if S is a numerical semigroup with Frobenius number f, then $f + \mathbb{N} \setminus \{0\} \subseteq S$. An integer z is a *pseudo-Frobenius number* of S if $z + S \setminus \{0\} \subseteq S$. Thus the

Frobenius number of S is one of its pseudo-Frobenius numbers. The *type* of a numerical semigroup is the cardinality of the set of its pseudo-Frobenius numbers.

The number of numerical semigroups having a given Frobenius number is finite. The elements in this set of numerical semigroups that are maximal with respect to set inclusion are precisely those numerical semigroups that cannot be expressed as intersection of two other numerical semigroups containing them properly, and thus they are known as *irreducible* numerical semigroups. Clearly, every numerical semigroup is the intersection of (finitely many) irreducible numerical semigroups.

A numerical semigroup S with Frobenius number f is *symmetric* if for every integer x, either $x \in S$ or $f - x \in S$. The set of irreducible numerical semigroups with odd Frobenius number coincides with the set of symmetric numerical semigroups. The numerical semigroup S is *pseudo-symmetric* if f is even and for every integer x not equal to $f/2$ either $x \in S$ or $f - x \in S$. The set of irreducible numerical semigroups with even Frobenius number is precisely the set of pseudo-symmetric numerical semigroups. These two classes of numerical semigroups have been widely studied in the literature due to their nice applications in Algebraic Geometry. This is probably one of the main reasons that made people turn their attention on numerical semigroups again in the last decades. Symmetric numerical semigroups can be also characterized as those with type one, and pseudo-symmetric numerical semigroups are those numerical semigroups with type two and such that its pseudo-Frobenius numbers are its Frobenius number and its Frobenius number divided by two.

Another class of numerical semigroups that catched the attention of researchers working on Algebraic Geometry and Commutative Ring Theory is the class of numerical semigroups with maximal embedding dimension. The *embedding dimension* of a numerical semigroup is the cardinality of its minimal system of generators. It can be shown that the embedding dimension is at most the multiplicity of the numerical semigroup. Thus *maximal embedding dimension* numerical semigroups are those numerical semigroups for which their embedding dimension and multiplicity coincide. These numerical semigroups have nice maximal properties, not only (of course) related to their embedding dimension, but also by means of their presentations. Among maximal embedding dimension there are two classes of numerical semigroups that have been studied due to the connections with the equivalence of algebroid branches. A numerical semigroup S is Arf if for every $x \geq y \geq z \in S$, then $x + y - z \in S$; and it is *saturated* if the following condition holds: if $s, s_1, \ldots, s_r \in S$ are such that $s_i \leq s$ for all $i \in \{1, \ldots, r\}$ and $z_1, \ldots, z_r \in \mathbb{Z}$ are such that $z_1 s_1 + \cdots + z_r s_r \geq 0$, then $s + z_1 s_1 + \cdots + z_r s_r \in S$.

If we look carefully inside the set of fundamental gaps of a numerical semigroup, we see that there are some fulfilling the condition that if they are added to the given numerical semigroup, then the resulting set is again a numerical semigroup. These elements are called *special gaps* of the numerical semigroup. A numerical semigroup other than the set of nonnegative integers is irreducible if and only if it has only a special gap.

The inverse operation to the one described in the above paragraph is that of removing an element of a numerical semigroup. If we want the resulting set to be a numerical semigroup, then the only thing we can remove is a minimal generator.

Let a, b, c, d be positive integers such that $a/b < c/d$, and let $I = [a/b, c/d]$. Then the set $\mathrm{S}(I) = \mathbb{N} \cap \bigcup_{n \geq 0} nI$ is a numerical semigroup. This class of numerical semigroups coincides with that of sets of solutions to equations of the form $Ax \; mod \; B \leq Cx$ with A, B, C positive integers. A numerical semigroup in this class is said to be *proportionally modular*.

A sequence of positive rational numbers $a_1/b_1 < \cdots < a_n/b_n$ with a_i, b_i positive integers is a *Bézout sequence* if $a_{i+1}b_i - a_i b_{i+1} = 1$ for all $i \in \{1, \ldots, n-1\}$. If $a/b = a_1/b_1 < \cdots < a_n/b_n = c/d$, then $\mathrm{S}([a/b, c/d]) = \langle a_1, \ldots, a_n \rangle$. Bézout sequences are not only interesting for this fact, they have shown to be a major tool in the study of proportionally modular numerical semigroups.

If S is a numerical semigroup and k is a positive integer, then the set $S/k = \{x \in \mathbb{N} \mid kx \in S\}$ is a numerical semigroup, known as the *quotient* S by k.

Let m be a positive integer. A *subadditive* function with period m is a map $f:\mathbb{N}\to\mathbb{N}$ such that $f(0)=0$, $f(x+y)\leq f(x)+f(y)$ and $f(x+m)=f(x)$. If f is a subadditive function with period m, then the set $\mathrm{M}_f=\{x\in\mathbb{N}\mid f(x)\leq x\}$ is a numerical semigroup. Moreover, every numerical semigroup is of this form. Thus a numerical semigroup can be given by a subadditive function with a given period. If S is a numerical semigroup and $s\in S, s\neq 0$, and $\mathrm{Ap}(S,s)=\{w(0),w(1),\ldots,w(s-1)\}$, then $f(x)=w(x\ mod\ s)$ is a subadditive function with period s such that $\mathrm{M}_f=S$.

Let S be a numerical semigroup generated by $\{n_1,\ldots,n_k\}$. Then we can define the following morphism (called sometimes the factorization morphism) by $\varphi:\mathbb{N}^k\to S$, $\varphi(a_1,\ldots,a_k)=a_1n_1+\cdots+a_kn_k$. If σ is the kernel congruence of φ (that is, $a\sigma b$ if $\varphi(a)=\varphi(b)$), then S is isomorphic to $\mathbb{N}^k/\sigma$. A *presentation* for S is a system of generators (as a congruence) of σ. If $\{n_1,\ldots,n_p\}$ is a minimal system of generators, then a *minimal presentation* is a presentation such that none of its proper subsets is a presentation. Minimal presentations of numerical semigroups coincide with presentations with minimal cardinality, though in general these two concepts are not the same for an arbitrary commutative semigroup.

A set I of integers is an *ideal relative to a numerical semigroup* S provided that $I+S\subseteq I$ and that there exists $d\in S$ such that $d+I\subseteq S$. If $I\subseteq S$, we simply say that I is an *ideal* of S. If I and J are relative ideals of S, then so is $I-J=\{z\in\mathbb{Z}\mid z+J\subseteq I\}$, and it is tightly related to the operation ":" of ideals in a commutative ring.

In this package we have implemented the functions needed to deal with the elements exposed in this introduction.

Many of the algorithms, and the necessary background to understand them, can be found in the monograph [RGS09]. Some examples in this book have been illustrated with the help of this package. So the reader can also find there more examples on the usage of the functions implemented here.

This package was presented in [DGSM06]. For a survey of the features of this package, see [DGS16].

Chapter 2

Numerical Semigroups

This chapter describes how to create numerical semigroups in GAP and perform some basic tests.

2.1 Generating Numerical Semigroups

We recall some definitions from Chapter 1.

A numerical semigroup is a subset of the set $\mathbb{N}$ of nonnegative integers that is closed under addition, contains 0 and whose complement in $\mathbb{N}$ is finite.

We refer to the elements in a numerical semigroup that are less than or equal to the conductor as *small elements* of the semigroup.

A *gap* of a numerical semigroup S is a nonnegative integer not belonging to S. The *fundamental gaps* of S are those gaps that are maximal with respect to the partial order induced by division in $\mathbb{N}$.

Given a numerical semigroup S and a nonzero element s in it, one can consider for every integer i ranging from 0 to $s-1$, the smallest element in S congruent with i modulo s, say $w(i)$ (this element exists since the complement of S in $\mathbb{N}$ is finite). Clearly $w(0)=0$. The set $\mathrm{Ap}(S,s) = \{w(0), w(1), \ldots, w(s-1)\}$ is called the *Apéry set* of S with respect to s.

Let a,b,c,d be positive integers such that $a/b < c/d$, and let $I = [a/b, c/d]$. Then the set $\mathrm{S}(I) = \mathbb{N} \cap \bigcup_{n \geq 0} nI$ is a numerical semigroup. This class of numerical semigroups coincides with that of sets of solutions to equations of the form $Ax \bmod B \leq Cx$ with A,B,C positive integers. A numerical semigroup in this class is said to be *proportionally modular*. If $C=1$, then it is said to be *modular*.

There are different ways to specify a numerical semigroup S, namely, by its generators; by its gaps, its fundamental or special gaps by its Apéry set, just to name some. In this section we describe functions that may be used to specify, in one of these ways, a numerical semigroup in GAP.

2.1.1 NumericalSemigroupByGenerators

▷ `NumericalSemigroupByGenerators(`*`List`*`)` (function)

▷ `NumericalSemigroup(`*`String, List`*`)` (function)

`List` is a list of nonnegative integers with greatest common divisor equal to one. These integers may be given as a list or by a sequence of individual elements. The output is the numerical semigroup spanned by `List`.

`String` does not need to be present. When it is present, it must be `"generators"`.

Example
```
gap> s1 := NumericalSemigroupByGenerators(3,5,7);
<Numerical semigroup with 3 generators>
gap> s2 := NumericalSemigroupByGenerators([3,5,7]);
<Numerical semigroup with 3 generators>
gap> s3 := NumericalSemigroup("generators",3,5,7);
<Numerical semigroup with 3 generators>
gap> s4 := NumericalSemigroup("generators",[3,5,7]);
<Numerical semigroup with 3 generators>
gap> s5 := NumericalSemigroup(3,5,7);
<Numerical semigroup with 3 generators>
gap> s6 := NumericalSemigroup([3,5,7]);
<Numerical semigroup with 3 generators>
gap> s1=s2;s2=s3;s3=s4;s4=s5;s5=s6;
true
true
true
true
true
```

2.1.2 NumericalSemigroupBySubAdditiveFunction

▷ NumericalSemigroupBySubAdditiveFunction(*List*) (function)
▷ NumericalSemigroup(*String*, *List*) (function)

A periodic subadditive function with period m is given through the list of images of the integers from 1 to m. The image of m has to be 0. The output is the numerical semigroup determined by this subadditive function.

In the second form, String must be "subadditive".

Example
```
gap> s := NumericalSemigroupBySubAdditiveFunction([5,4,2,0]);
<Numerical semigroup>
gap> t := NumericalSemigroup("subadditive",[5,4,2,0]);;
gap> s=t;
true
```

2.1.3 NumericalSemigroupByAperyList

▷ NumericalSemigroupByAperyList(*List*) (function)
▷ NumericalSemigroup(*String*, *List*) (function)

List is an Apéry list. The output is the numerical semigroup whose Apéry set with respect to the length of given list is List.

In the second form, String must be "apery".

Example
```
gap> s:=NumericalSemigroup(3,11);;
gap> ap := AperyListOfNumericalSemigroupWRTElement(s,20);
[ 0, 21, 22, 3, 24, 25, 6, 27, 28, 9, 30, 11, 12, 33, 14, 15, 36, 17, 18, 39 ]
gap> t:=NumericalSemigroupByAperyList(ap);;
gap> r := NumericalSemigroup("apery",ap);;
```

```
gap> s=t;t=r;
true
true
```

2.1.4 NumericalSemigroupBySmallElements

▷ NumericalSemigroupBySmallElements(*List*) (function)
▷ NumericalSemigroup(*String, List*) (function)

`List` is the set of small elements of a numerical semigroup, that is, the set of all elements not greater than the conductor. The output is the numerical semigroup with this set of small elements. When no such semigroup exists, an error is returned.

In the second form, `String` must be `"elements"`.

Example
```
gap> s:=NumericalSemigroup(3,11);;
gap> se := SmallElements(s);
[ 0, 3, 6, 9, 11, 12, 14, 15, 17, 18, 20 ]
gap> t := NumericalSemigroupBySmallElements(se);;
gap> r := NumericalSemigroup("elements",se);;
gap> s=t;t=r;
true
true
gap> e := [ 0, 3, 6, 9, 11, 14, 15, 17, 18, 20 ];
[ 0, 3, 6, 9, 11, 14, 15, 17, 18, 20 ]
gap> NumericalSemigroupBySmallElements(e);
Error, The argument does not represent a numerical semigroup called from
<function "NumericalSemigroupBySmallElements">( <arguments> )
 called from read-eval loop at line 35 of *stdin*
you can 'quit;' to quit to outer loop, or
you can 'return;' to continue
brk>
```

2.1.5 NumericalSemigroupByGaps

▷ NumericalSemigroupByGaps(*List*) (function)
▷ NumericalSemigroup(*String, List*) (function)

`List` is the set of gaps of a numerical semigroup. The output is the numerical semigroup with this set of gaps. When no semigroup exists with the given set as set of gaps, an error is returned.

In the second form, `String` must be `"gaps"`.

Example
```
gap> g := [ 1, 2, 4, 5, 7, 8, 10, 13, 16 ];;
gap> s := NumericalSemigroupByGaps(g);;
gap> t := NumericalSemigroup("gaps",g);;
gap> s=t;
true
gap> h := [ 1, 2, 5, 7, 8, 10, 13, 16 ];;
gap> NumericalSemigroupByGaps(h);
Error, The argument does not represent the gaps of a numerical semigroup called
 from
```

```
<function "NumericalSemigroupByGaps">( <arguments> )
 called from read-eval loop at line 34 of *stdin*
you can 'quit;' to quit to outer loop, or
you can 'return;' to continue
brk>
```

2.1.6 NumericalSemigroupByFundamentalGaps

▷ NumericalSemigroupByFundamentalGaps(*List*) (function)
▷ NumericalSemigroup(*String*, *List*) (function)

List is the set of fundamental gaps of a numerical semigroup. The output is the numerical semigroup determined by these gaps. When the given set contains elements (which will be gaps) that are not fundamental gaps, they are silently removed.

In the second form, String must be "fundamentalgaps".

Example
```
gap> fg := [ 11, 14, 17, 20, 23, 26, 29, 32, 35 ];;
gap> NumericalSemigroupByFundamentalGaps(fg);
<Numerical semigroup>
gap> NumericalSemigroup("fundamentalgaps",fg);
<Numerical semigroup>
gap> last=last2;
true
gap> gg := [ 11, 17, 20, 22, 23, 26, 29, 32, 35 ];; #22 is not fundamental
gap> NumericalSemigroup("fundamentalgaps",fg);
<Numerical semigroup>
```

2.1.7 NumericalSemigroupByAffineMap

▷ NumericalSemigroupByAffineMap(a, b, c) (function)
▷ NumericalSemigroup(*String*, a, b) (function)

Given three nonnegative integers a, b and c, with $a,c > 0$ and $\gcd(b,c) = 1$, this function returns the least (with restpect to set order inclusion) numerical semigroup containing c and closed under the map $x \mapsto ax+b$. The procedure is explained in [Ugo16].

In the second form, String must be "affinemap".

Example
```
gap> s:=NumericalSemigroupByAffineMap(3,1,3);
<Numerical semigroup with 3 generators>
gap> SmallElements(s);
[ 0, 3, 6, 9, 10, 12, 13, 15, 16, 18 ]
gap> t:=NumericalSemigroup("affinemap",3,1,3);;
gap> s=t;
true
```

2.1.8 ModularNumericalSemigroup

▷ ModularNumericalSemigroup(a, b) (function)
▷ NumericalSemigroup(*String*, a, b) (function)

Given two positive integers `a` and `b`, this function returns a modular numerical semigroup satisfying $ax \bmod b \leq x$.

In the second form, `String` must be `"modular"`.

Example

```
gap> ModularNumericalSemigroup(3,7);
<Modular numerical semigroup satisfying 3x mod 7 <= x >
gap> NumericalSemigroup("modular",3,7);
<Modular numerical semigroup satisfying 3x mod 7 <= x >
```

2.1.9 ProportionallyModularNumericalSemigroup

▷ `ProportionallyModularNumericalSemigroup(a, b, c)` (function)

▷ `NumericalSemigroup(String, a, b)` (function)

Given three positive integers `a`, `b` and `c`, this function returns a proportionally modular numerical semigroup satisfying $ax \bmod b \leq cx$.

In the second form, `String` must be `"propmodular"`.

Example

```
gap> ProportionallyModularNumericalSemigroup(3,7,12);
<Proportionally modular numerical semigroup satisfying 3x mod 7 <= 12x >
gap> NumericalSemigroup("propmodular",3,7,12);
<Proportionally modular numerical semigroup satisfying 3x mod 7 <= 12x >
```

When $c = 1$, the semigroup is seen as a modular numerical semigroup.

Example

```
gap> NumericalSemigroup("propmodular",67,98,1);
<Modular numerical semigroup satisfying 67x mod 98 <= x >
```

Numerical semigroups generated by an interval of positive integers are known to be proportionally modular, and thus they are treated as such, since membership and other problems can be solved efficiently for these semigroups.

2.1.10 NumericalSemigroupByInterval

▷ `NumericalSemigroupByInterval(List)` (function)

▷ `NumericalSemigroup(String, List)` (function)

The input is a list of rational numbers defining a closed interval. The output is the semigroup of numerators of all rational numbers in this interval.

`String` does not need to be present. When it is present, it must be `"interval"`.

Example

```
gap> NumericalSemigroupByInterval(7/5,5/3);
<Proportionally modular numerical semigroup satisfying 25x mod 35 <= 4x >
gap> NumericalSemigroup("interval",[7/5,5/3]);
<Proportionally modular numerical semigroup satisfying 25x mod 35 <= 4x >
gap> SmallElements(last);
[ 0, 3, 5 ]
```

2.1.11 NumericalSemigroupByOpenInterval

▷ `NumericalSemigroupByOpenInterval(List)` (function)
▷ `NumericalSemigroup(String, List)` (function)

The input is a list of rational numbers defining a open interval. The output is the semigroup of numerators of all rational numbers in this interval.

`String` does not need to be present. When it is present, it must be `"openinterval"`.

Example
```
gap> NumericalSemigroupByOpenInterval(7/5,5/3);
<Numerical semigroup>
gap> NumericalSemigroup("openinterval",[7/5,5/3]);
<Numerical semigroup>
gap> SmallElements(last);
[ 0, 3, 6, 8 ]
```

2.2 Some basic tests

This section describes some basic tests on numerical semigroups. The first described tests refer to what the semigroup is currently known to be (not necessarily the way it was created). Then are presented functions to test if a given list represents the small elements, gaps or the Apéry set (see 1) of a numerical semigroup; to test if an integer belongs to a numerical semigroup and if a numerical semigroup is a subsemigroup of another one.

2.2.1 IsNumericalSemigroup

▷ `IsNumericalSemigroup(NS)` (attribute)
▷ `IsNumericalSemigroupByGenerators(NS)` (attribute)
▷ `IsNumericalSemigroupByInterval(NS)` (attribute)
▷ `IsNumericalSemigroupByOpenInterval(NS)` (attribute)
▷ `IsNumericalSemigroupBySubAdditiveFunction(NS)` (attribute)
▷ `IsNumericalSemigroupByAperyList(NS)` (attribute)
▷ `IsNumericalSemigroupBySmallElements(NS)` (attribute)
▷ `IsNumericalSemigroupByGaps(NS)` (attribute)
▷ `IsNumericalSemigroupByFundamentalGaps(NS)` (attribute)
▷ `IsProportionallyModularNumericalSemigroup(NS)` (attribute)
▷ `IsModularNumericalSemigroup(NS)` (attribute)

NS is a numerical semigroup and these attributes are available (their names should be self explanatory). The attribute `IsNumericalSemigroupByMinimalGenerators` is obsolet and is to be removed in a further version.

Example
```
gap> s:=NumericalSemigroup(3,7);
<Numerical semigroup with 2 generators>
gap> AperyListOfNumericalSemigroupWRTElement(s,30);;
gap> t:=NumericalSemigroupByAperyList(last);
<Numerical semigroup>
gap> IsNumericalSemigroupByGenerators(s);
true
```

```
gap> IsNumericalSemigroupByGenerators(t);
false
gap> IsNumericalSemigroupByAperyList(s);
false
gap> IsNumericalSemigroupByAperyList(t);
true
```

2.2.2 RepresentsSmallElementsOfNumericalSemigroup

▷ RepresentsSmallElementsOfNumericalSemigroup(*L*) (attribute)

Tests if the list *L* (which has to be a set) may represent the “small" elements of a numerical semigroup.

Example
```
gap> L:=[ 0, 3, 6, 9, 11, 12, 14, 15, 17, 18, 20 ];
[ 0, 3, 6, 9, 11, 12, 14, 15, 17, 18, 20 ]
gap> RepresentsSmallElementsOfNumericalSemigroup(L);
true
gap> L:=[ 6, 9, 11, 12, 14, 15, 17, 18, 20 ];
[ 6, 9, 11, 12, 14, 15, 17, 18, 20 ]
gap> RepresentsSmallElementsOfNumericalSemigroup(L);
false
```

2.2.3 RepresentsGapsOfNumericalSemigroup

▷ RepresentsGapsOfNumericalSemigroup(*L*) (attribute)

Tests if the list *L* may represent the gaps (see 1) of a numerical semigroup.

Example
```
gap> s:=NumericalSemigroup(3,7);
<Numerical semigroup with 2 generators>
gap> L:=GapsOfNumericalSemigroup(s);
[ 1, 2, 4, 5, 8, 11 ]
gap> RepresentsGapsOfNumericalSemigroup(L);
true
gap> L:=Set(List([1..21],i->RandomList([1..50])));
[ 2, 6, 7, 8, 10, 12, 14, 19, 24, 28, 31, 35, 42, 50 ]
gap> RepresentsGapsOfNumericalSemigroup(L);
false
```

2.2.4 IsAperyListOfNumericalSemigroup

▷ IsAperyListOfNumericalSemigroup(*L*) (function)

Tests whether a list *L* of integers may represent the Apéry list of a numerical semigroup. It returns `true` when the periodic function represented by *L* is subadditive (see `RepresentsPeriodicSubAdditiveFunction` (A.2.1)) and the remainder of the division of `L[i]` by the length of *L* is `i` and returns `false` otherwise (the criterium used is the one explained in [Ros96b]).

Example
```
gap> IsAperyListOfNumericalSemigroup([0,21,7,28,14]);
true
```

2.2.5 IsSubsemigroupOfNumericalSemigroup

▷ IsSubsemigroupOfNumericalSemigroup(*S*, *T*) (function)

S and T are numerical semigroups. Tests whether T is contained in S.

Example
```
gap> S := NumericalSemigroup("modular", 5,53);
<Modular numerical semigroup satisfying 5x mod 53 <= x >
gap> T:=NumericalSemigroup(2,3);
<Numerical semigroup with 2 generators>
gap> IsSubsemigroupOfNumericalSemigroup(T,S);
true
gap> IsSubsemigroupOfNumericalSemigroup(S,T);
false
```

2.2.6 IsSubset

▷ IsSubset(*S*, *T*) (attribute)

S is a numerical semigroup. T can be a numerical semigroup, in which case the function is just a synonym of `IsSubsemigroupOfNumericalSemigroup` (2.2.5), or a list of integers, in which case tests whether all elements of the list belong to S.

Example
```
gap> ns1 := NumericalSemigroup(5,7);;
gap> ns2 := NumericalSemigroup(5,7,11);;
gap> IsSubset(ns1,ns2);
false
gap> IsSubset(ns2,[5,15]);
true
gap> IsSubset(ns1,[5,11]);
false
gap> IsSubset(ns2,ns1);
true
```

2.2.7 BelongsToNumericalSemigroup

▷ BelongsToNumericalSemigroup(*n*, *S*) (operation)
▷ \in(*n*, *S*) (operation)

n is an integer and S is a numerical semigroup. Tests whether n belongs to S. `\in(n,S)` calls the infix variant `n in S`, and both can be seen as a short for `BelongsToNumericalSemigroup(n,S)`.

Example
```
gap> S := NumericalSemigroup("modular", 5,53);
<Modular numerical semigroup satisfying 5x mod 53 <= x >
gap> BelongsToNumericalSemigroup(15,S);
```

```
false
gap> 15 in S;
false
gap> SmallElementsOfNumericalSemigroup(S);
[ 0, 11, 12, 13, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 37, 38, 39, 43 ]
gap> BelongsToNumericalSemigroup(13,S);
true
gap> 13 in S;
true
```

Chapter 3

Basic operations with numerical semigroups

This chapter describes some basic functions to deal with notable elements in a numerical semigroup. A section including functions to test Wilf's conjecture is also included in this chapter. We provide some functions that allow to treat a numerical semigroup as a list, and thus easy the task to access to its elements.

3.1 Invariants

In this section we present formulas to compute invariants and notable elements of a numerical semigroup. Some tests depending on these invariants are provided heres, like being an acute or an ordinary numerical semigroup. We also present procedures to construct iterators from a numerical semigroup, or to retrieve several elemements from a numerical semigroup as if it where a list (with infinitely many elements).

3.1.1 Multiplicity (for numerical semigroup)

▷ `Multiplicity(NS)` (attribute)
▷ `MultiplicityOfNumericalSemigroup(NS)` (attribute)

NS is a numerical semigroup. Returns the multiplicity of *NS*, which is the smallest positive integer belonging to *NS*.

Example

```
gap> S := NumericalSemigroup("modular", 7,53);
<Modular numerical semigroup satisfying 7x mod 53 <= x >
gap> MultiplicityOfNumericalSemigroup(S);
8
gap> NumericalSemigroup(3,5);
<Numerical semigroup with 2 generators>
gap> Multiplicity(last);
3
```

3.1.2 GeneratorsOfNumericalSemigroup

▷ `GeneratorsOfNumericalSemigroup(S)` (attribute)
▷ `Generators(S)` (attribute)
▷ `MinimalGeneratingSystemOfNumericalSemigroup(S)` (attribute)
▷ `MinimalGeneratingSystem(S)` (attribute)
▷ `MinimalGenerators(S)` (attribute)

S is a numerical semigroup. `GeneratorsOfNumericalSemigroup` returns a set of generators of `S`, which may not be minimal. The shorter name `Generators` may be used. `MinimalGeneratingSystemOfNumericalSemigroup` returns the minimal set of generators of `S`. The shorter names `MinimalGenerators` or `MinimalGeneratingSystem` may be used.

From Version 0.980, `ReducedSetOfGeneratorsOfNumericalSemigroup` is a synonym of `MinimalGeneratingSystemOfNumericalSemigroup`; `GeneratorsOfNumericalSemigroupNC` is a synonym of `GeneratorsOfNumericalSemigroup`. The names are kept for compatibility with code produced for previous versions, but will be removed in the future.

Example
```
gap> S := NumericalSemigroup("modular", 5,53);
<Modular numerical semigroup satisfying 5x mod 53 <= x >
gap> GeneratorsOfNumericalSemigroup(S);
[ 11, 12, 13, 32, 53 ]
gap> S := NumericalSemigroup(3, 5, 53);
<Numerical semigroup with 3 generators>
gap> GeneratorsOfNumericalSemigroup(S);
[ 3, 5, 53 ]
gap> MinimalGeneratingSystemOfNumericalSemigroup(S);
[ 3, 5 ]
gap> MinimalGeneratingSystem(S)=MinimalGeneratingSystemOfNumericalSemigroup(S);
true
gap> s := NumericalSemigroup(3,5,7,15);
<Numerical semigroup with 4 generators>
gap> HasGenerators(s);
true
gap> HasMinimalGenerators(s);
false
gap> MinimalGenerators(s);
[ 3, 5, 7 ]
gap> Generators(s);
[ 3, 5, 7, 15 ]
```

3.1.3 EmbeddingDimension (for numerical semigroup)

▷ `EmbeddingDimension(NS)` (attribute)
▷ `EmbeddingDimensionOfNumericalSemigroup(NS)` (attribute)

`NS` is a numerical semigroup. It returns the cardinality of its minimal generating system.

Example
```
gap> s := NumericalSemigroup(3,5,7,15);
<Numerical semigroup with 4 generators>
gap> EmbeddingDimension(s);
```

```
3
gap> EmbeddingDimensionOfNumericalSemigroup(s);
3
```

3.1.4 SmallElements (for numerical semigroup)

▷ SmallElements(*NS*) (attribute)
▷ SmallElementsOfNumericalSemigroup(*NS*) (attribute)

NS is a numerical semigroup. It returns the list of small elements of NS. Of course, the time consumed to return a result may depend on the way the semigroup is given.

Example
```
gap> SmallElementsOfNumericalSemigroup(NumericalSemigroup(3,5,7));
[ 0, 3, 5 ]
gap> SmallElements(NumericalSemigroup(3,5,7));
[ 0, 3, 5 ]
```

3.1.5 FirstElementsOfNumericalSemigroup

▷ FirstElementsOfNumericalSemigroup(*n*, *NS*) (function)

NS is a numerical semigroup. It returns the list with the first *n* elements of NS.

Example
```
gap> FirstElementsOfNumericalSemigroup(2,NumericalSemigroup(3,5,7));
[ 0, 3 ]
gap> FirstElementsOfNumericalSemigroup(10,NumericalSemigroup(3,5,7));
[ 0, 3, 5, 6, 7, 8, 9, 10, 11, 12 ]
```

3.1.6 \[\] (for numerical semigroups)

▷ \[\](*S*, *r*) (operation)

S is a numerical semigroup and *r* is an integer. It returns the *r*-th element of *S*.

Example
```
gap> S := NumericalSemigroup(7,8,17);;
gap> S[53];
68
```

3.1.7 \{ \} (for numerical semigroups)

▷ \{ \}(*S*, *ls*) (operation)

S is a numerical semigroup and *ls* is a list of integers. It returns the list *[S[r] : r in ls]*.

Example
```
gap> S := NumericalSemigroup(7,8,17);;
gap> S{[1..5]};
[ 0, 7, 8, 14, 15 ]
```

3.1.8 NextElementOfNumericalSemigroup

▷ `NextElementOfNumericalSemigroup(S, r)` (operation)

S is a numerical semigroup and *r* is an integer. It returns the returns the least integer greater than *r* belonging to *S*.

Example
```
gap> S := NumericalSemigroup(7,8,17);;
gap> NextElementOfNumericalSemigroup(S,9);
14
gap> NextElementOfNumericalSemigroup(16,S);
17
gap> NextElementOfNumericalSemigroup(S,FrobeniusNumber(S))=Conductor(S);
true
```

3.1.9 ElementNumber_NumericalSemigroup

▷ `ElementNumber_NumericalSemigroup(S, r)` (function)

S is a numerical semigroup and *r* is an integer. Both functions (which are like synonyms) return the *r*-th element of *S*.

Example
```
gap> S := NumericalSemigroup(7,8,17);;
gap> ElementNumber_NumericalSemigroup(S,53);
68
gap> RthElementOfNumericalSemigroup(S,53);
68
```

3.1.10 RthElementOfNumericalSemigroup

▷ `RthElementOfNumericalSemigroup(S, r)` (operation)

This operation works as a synonym of `ElementNumber_NumericalSemigroup` (3.1.9).

Example
```
gap> S := NumericalSemigroup(7,8,17);;
gap> RthElementOfNumericalSemigroup(S,53);
68
```

3.1.11 NumberElement_NumericalSemigroup

▷ `NumberElement_NumericalSemigroup(S, r)` (function)

S is a numerical semigroup and *r* is an integer. It returns the position of *r* in *S* (and `fail` if the integer is not in the semigroup).

Example
```
gap> S := NumericalSemigroup(7,8,17);;
gap> NumberElement_NumericalSemigroup(S,68);
53
```

3.1.12 Iterator (for numerical semigroups)

▷ `Iterator(S)` (operation)

S is a numerical semigroup. It returns an iterator over S.

Example
```
gap> S := NumericalSemigroup(7,8,17);;
gap> iter:=Iterator(S);
<iterator>
gap> NextIterator(iter);
0
gap> NextIterator(iter);
7
gap> NextIterator(iter);
8
```

3.1.13 AperyList (for numerical semigroup with respect to element)

▷ `AperyList(S, n)` (attribute)
▷ `AperyListOfNumericalSemigroupWRTElement(S, n)` (operation)

S is a numerical semigroup and n is a positive element of S. Computes the Apéry list of S with respect to n. It contains for every $i \in \{0,\dots,n-1\}$, in the $i+1$th position, the smallest element in the semigroup congruent with i modulo n.

Example
```
gap> S := NumericalSemigroup("modular", 5,53);;
gap> AperyListOfNumericalSemigroupWRTElement(S,12);
[ 0, 13, 26, 39, 52, 53, 54, 43, 32, 33, 22, 11 ]
gap> AperyList(S,12);
[ 0, 13, 26, 39, 52, 53, 54, 43, 32, 33, 22, 11 ]
```

3.1.14 AperyList (for numerical semigroup with respect to multiplicity)

▷ `AperyList(S)` (attribute)
▷ `AperyListOfNumericalSemigroup(S)` (attribute)

S is a numerical semigroup. It computes the Apéry list of S with respect to the multiplicity of S.

Example
```
gap> S := NumericalSemigroup("modular", 5,53);;
gap> AperyListOfNumericalSemigroup(S);
[ 0, 12, 13, 25, 26, 38, 39, 51, 52, 53, 32 ]
gap> AperyList(NumericalSemigroup(5,7,11));
[ 0, 11, 7, 18, 14 ]
```

3.1.15 AperyList (for numerical semigroup with respect to integer)

▷ `AperyList(S, n)` (attribute)
▷ `AperyListOfNumericalSemigroupWRTInteger(S, m)` (function)

`S` is a numerical semigroup and `m` is an integer. Computes the Apéry list of `S` with respect to `m`, that is, the set of elements x in `S` such that x−`m` is not in `S`. If `m` is an element in `S`, then the output of `AperyListOfNumericalSemigroupWRTInteger`, as sets, is the same as `AperyListOfNumericalSemigroupWRTElement`, though without side effects, in the sense that this information is no longer used by the package. The output of `AperyList` is the same as `AperyListOfNumericalSemigroupWRTElement`.

Example

```
gap>  s:=NumericalSemigroup(10,13,19,27);;
gap> AperyListOfNumericalSemigroupWRTInteger(s,11);
[ 0, 10, 13, 19, 20, 23, 26, 27, 29, 32, 33, 36, 39, 42, 45, 46, 52, 55 ]
gap> AperyList(s,11);
[ 0, 10, 13, 19, 20, 23, 26, 27, 29, 32, 33, 36, 39, 42, 45, 46, 52, 55 ]
gap> Length(last);
18
gap> AperyListOfNumericalSemigroupWRTInteger(s,10);
[ 0, 13, 19, 26, 27, 32, 38, 45, 51, 54 ]
gap> AperyListOfNumericalSemigroupWRTElement(s,10);
[ 0, 51, 32, 13, 54, 45, 26, 27, 38, 19 ]
gap> Length(last);
10
gap> AperyList(s,10);
[ 0, 51, 32, 13, 54, 45, 26, 27, 38, 19 ]
```

3.1.16 AperyListOfNumericalSemigroupAsGraph

▷ `AperyListOfNumericalSemigroupAsGraph(ap)` (function)

ap is the Apéry list of a numerical semigroup. This function returns the adjacency list of the graph (ap,E) where the edge $u->v$ is in E iff $v-u$ is in ap. The 0 is ignored.

Example

```
gap> s:=NumericalSemigroup(3,7);;
gap> AperyListOfNumericalSemigroupWRTElement(s,10);
[ 0, 21, 12, 3, 14, 15, 6, 7, 18, 9 ]
gap> AperyListOfNumericalSemigroupAsGraph(last);
[ ,, [ 3, 6, 9, 12, 15, 18, 21 ],,, [ 6, 9, 12, 15, 18, 21 ],
[ 7, 14, 21 ],, [ 9, 12, 15, 18, 21 ],,, [ 12, 15, 18, 21 ],,
[ 14, 21 ], [ 15, 18, 21 ],,, [ 18, 21 ],,, [ 21 ] ]
```

3.1.17 KunzCoordinatesOfNumericalSemigroup

▷ `KunzCoordinatesOfNumericalSemigroup(S, m)` (function)

`S` is a numerical semigroup, and `m` is a nonzero element of `S`. The second argument is optional, and if missing it is assumed to be the multiplicity of `S`.

Then the Apéry set of `m` in `S` has the form $[0,k_1m+1,\dots,k_{m-1}m+m-1]$, and the output is the $(m-1)$-uple $[k_1,k_2,\dots,k_{m-1}]$

Example

```
gap> s:=NumericalSemigroup(3,5,7);
<Numerical semigroup with 3 generators>
gap> KunzCoordinatesOfNumericalSemigroup(s);
```

```
[ 2, 1 ]
gap> KunzCoordinatesOfNumericalSemigroup(s,5);
[ 1, 1, 0, 1 ]
```

3.1.18 KunzPolytope

▷ `KunzPolytope(m)` (function)

`m` is a positive integer.

The Kunz coordinates of the semigroups with multiplicity `m` are solutions of a system of inequalities $Ax \geq b$ (see [RGSB02]). The output is the matrix $(A| - b)$.

Example
```
gap> KunzPolytope(3);
[ [ 1, 0, -1 ], [ 0, 1, -1 ], [ 2, -1, 0 ], [ -1, 2, 1 ] ]
```

3.1.19 CocycleOfNumericalSemigroupWRTElement

▷ `CocycleOfNumericalSemigroupWRTElement(S, m)` (function)

`S` is a numerical semigroup, and `m` is a nonzero element of `S`. The output is the matrix $h(i, j) = (w(i)+w(j)-w((i+j) \bmod m))/m$, where $w(i)$ is the smallest element in `S` congruent with i modulo m (and thus it is in the Apéry set of m), [GSHKR17].

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> CocycleOfNumericalSemigroupWRTElement(s,3);
[ [ 0, 0, 0 ], [ 0, 3, 4 ], [ 0, 4, 1 ] ]
```

3.1.20 FrobeniusNumber (for numerical semigroup)

▷ `FrobeniusNumber(NS)` (attribute)
▷ `FrobeniusNumberOfNumericalSemigroup(NS)` (attribute)

The largest nonnegative integer not belonging to a numerical semigroup S is the *Frobenius number* of S. If S is the set of nonnegative integers, then clearly its Frobenius number is -1, otherwise its Frobenius number coincides with the maximum of the gaps (or fundamental gaps) of S.

`NS` is a numerical semigroup. It returns the Frobenius number of `NS`. Of course, the time consumed to return a result may depend on the way the semigroup is given or on the knowledge already produced on the semigroup.

Example
```
gap> FrobeniusNumberOfNumericalSemigroup(NumericalSemigroup(3,5,7));
4
gap> FrobeniusNumber(NumericalSemigroup(3,5,7));
4
```

3.1.21 Conductor (for numerical Semigroup)

▷ Conductor(*NS*) (attribute)
▷ ConductorOfNumericalSemigroup(*NS*) (attribute)

This is just a synonym of FrobeniusNumberOfNumericalSemigroup (NS)+1.

Example
```
gap> ConductorOfNumericalSemigroup(NumericalSemigroup(3,5,7));
5
gap> Conductor(NumericalSemigroup(3,5,7));
5
```

3.1.22 PseudoFrobeniusOfNumericalSemigroup

▷ PseudoFrobeniusOfNumericalSemigroup(*S*) (attribute)

An integer z is a *pseudo-Frobenius number* of S if $z+S\setminus\{0\}\subseteq S$.
S is a numerical semigroup. It returns the set of pseudo-Frobenius numbers of *S*.

Example
```
gap> S := NumericalSemigroup("modular", 5,53);
<Modular numerical semigroup satisfying 5x mod 53 <= x >
gap> PseudoFrobeniusOfNumericalSemigroup(S);
[ 21, 40, 41, 42 ]
```

3.1.23 TypeOfNumericalSemigroup

▷ TypeOfNumericalSemigroup(*NS*) (attribute)

Stands for Length(PseudoFrobeniusOfNumericalSemigroup (NS)).

Example
```
gap> S := NumericalSemigroup("modular", 5,53);
<Modular numerical semigroup satisfying 5x mod 53 <= x >
gap> Type(S);
4
gap> TypeOfNumericalSemigroup(S);
4
```

3.1.24 Gaps (for numerical semigroup)

▷ Gaps(*NS*) (attribute)
▷ GapsOfNumericalSemigroup(*NS*) (attribute)

A *gap* of a numerical semigroup S is a nonnegative integer not belonging to S. NS is a numerical semigroup. Both return the set of gaps of NS.

Example
```
gap> GapsOfNumericalSemigroup(NumericalSemigroup(3,5,7));
[ 1, 2, 4 ]
gap> Gaps(NumericalSemigroup(5,7,11));
[ 1, 2, 3, 4, 6, 8, 9, 13 ]
```

3.1.25 Weight (for numerical semigroup)

▷ `Weight(NS)` (attribute)

If $l_1 < \cdots < l_g$ are the gaps of `NS`, then its (Weierstrass) weight is $\sum_{i=1}^{g}(l_i - i)$.

Example
```
gap> Weight(NumericalSemigroup(4,5,6,7));
0
gap> Weight(NumericalSemigroup(4,5));
9
```

3.1.26 DesertsOfNumericalSemigroup

▷ `DesertsOfNumericalSemigroup(NS)` (function)

NS is a numerical semigroup. The output is the list with the runs of gaps of *NS*.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> DesertsOfNumericalSemigroup(s);
[ [ 1, 2 ], [ 4 ] ]
```

3.1.27 IsOrdinaryNumericalSemigroup

▷ `IsOrdinaryNumericalSemigroup(NS)` (property)
▷ `IsOrdinary(NS)` (property)

NS is a numerical semigroup. Dectects if the semigroup is ordinary, that is, with less than two deserts.

This filter implies `IsAcuteNumericalSemigroup` (3.1.28).

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> IsOrdinary(s);
false
```

3.1.28 IsAcuteNumericalSemigroup

▷ `IsAcuteNumericalSemigroup(NS)` (property)
▷ `IsAcute(NS)` (property)

NS is a numerical semigroup. Dectects if the semigroup is acute, that is, it is either ordinary or its last desert (the one with the Frobenius number) has less elements than the preceding one ([BA04]).

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> IsAcute(s);
true
```

3.1.29 Holes (for numerical semigroup)

▷ `Holes(NS)` (attribute)
▷ `HolesOfNumericalSemigroup(S)` (attribute)

S is a numerical semigroup. Returns the set of gaps x of S such that $F(S)-x$ is also a gap, where $F(S)$ stands for the Frobenius number of S.

Example

```
gap> s:=NumericalSemigroup(3,5);;
gap> Holes(s);
[  ]
gap> s:=NumericalSemigroup(3,5,7);;
gap> HolesOfNumericalSemigroup(s);
[ 2 ]
```

3.1.30 LatticePathAssociatedToNumericalSemigroup

▷ `LatticePathAssociatedToNumericalSemigroup(S, p, q)` (attribute)

S is a numerical semigroup and p,q are two coprime elements in S.

In this setting S is an oversemigroup of $\langle p,q\rangle$, and consequently every gap of S is a gap of $\langle p,q\rangle$. If c is the conductor of $\langle p,q\rangle$, then every gap g of $\langle p,q\rangle$ can be written uniquely as $g=c-1-(ap+bp)$ for some nonnegative integers a,b. We say that (a,b) are the coordinates associated to g.

The output is a path in $\mathbb{N}^2$ such that the coordinates of the gaps of S correspond exactly with the points in $\mathbb{N}^2$ that are between the path and the line $ax+by=c-1$. See [KW14].

Example

```
gap> s:=NumericalSemigroup(16,17,71,72);;
gap> LatticePathAssociatedToNumericalSemigroup(s,16,17);
[ [ 0, 14 ], [ 1, 13 ], [ 2, 12 ], [ 3, 11 ], [ 4, 10 ], [ 5, 9 ], [ 6, 8 ],
  [ 7, 7 ], [ 8, 6 ], [ 9, 5 ], [ 10, 4 ], [ 11, 3 ], [ 12, 2 ], [ 13, 1 ],
  [ 14, 0 ] ]
```

3.1.31 Genus (for numerical semigroup)

▷ `Genus(NS)` (attribute)
▷ `GenusOfNumericalSemigroup(NS)` (attribute)

NS is a numerical semigroup. It returns the number of gaps of NS.

Example

```
gap> s:=NumericalSemigroup(16,17,71,72);;
gap> GenusOfNumericalSemigroup(s);
80
gap> S := NumericalSemigroup("modular", 5,53);
<Modular numerical semigroup satisfying 5x mod 53 <= x >
gap> Genus(S);
26
```

3.1.32 FundamentalGaps (for numerical semigroup)

▷ `FundamentalGaps(S)` (attribute)
▷ `FundamentalGapsOfNumericalSemigroup(S)` (attribute)

S The *fundamental gaps* of S are those gaps that are maximal with respect to the partial order induced by division in $\mathbb{N}$. It returns the set of fundamental gaps of S.

Example

```
gap> S := NumericalSemigroup("modular", 5,53);
<Modular numerical semigroup satisfying 5x mod 53 <= x >
gap> FundamentalGapsOfNumericalSemigroup(S);
[ 16, 17, 18, 19, 27, 28, 29, 30, 31, 40, 41, 42 ]
gap> GapsOfNumericalSemigroup(S);
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14, 15, 16, 17, 18, 19, 20, 21, 27, 28, 29,
  30, 31, 40, 41, 42 ]
gap> Gaps(NumericalSemigroup(5,7,11));
[ 1, 2, 3, 4, 6, 8, 9, 13 ]
gap> FundamentalGaps(NumericalSemigroup(5,7,11));
[ 6, 8, 9, 13 ]
```

3.1.33 SpecialGaps (for numerical semigroup)

▷ `SpecialGaps(S)` (attribute)
▷ `SpecialGapsOfNumericalSemigroup(S)` (attribute)

The *special gaps* of a numerical semigroup S are those fundamental gaps such that if they are added to the given numerical semigroup, then the resulting set is again a numerical semigroup. S is a numerical semigroup. It returns the special gaps of S.

Example

```
gap> S := NumericalSemigroup("modular", 5,53);
<Modular numerical semigroup satisfying 5x mod 53 <= x >
gap> SpecialGaps(S);
[ 40, 41, 42 ]
gap> SpecialGapsOfNumericalSemigroup(S);
[ 40, 41, 42 ]
```

3.2 Wilf's conjecture

Let S be a numerical semigroup, with conductor c and embedding dimension e. Denote by l the cardinality of the set of elements in S smaller than c. Wilf in [Wil78] asked whether or not $l/c \geq 1/e$ for all numerical semigroups. In this section we give some functions to experiment with this conjecture, as defined in [Eli15].

3.2.1 WilfNumber (for numerical semigroup)

▷ `WilfNumber(S)` (attribute)
▷ `WilfNumberOfNumericalSemigroup(S)` (attribute)

S is a numerical semigroup. Let c, e and l be the conductor, embedding dimension and number of elements smaller than c in S. Returns $el-c$, which was conjetured by Wilf to be nonnegative.

Example

```
gap> l:=NumericalSemigroupsWithGenus(10);;
gap> Filtered(l, s->WilfNumberOfNumericalSemigroup(s)<0);
[  ]
gap> Maximum(Set(l, s->WilfNumberOfNumericalSemigroup(s)));
70
gap> s := NumericalSemigroup(13,25,37);;
gap> WilfNumber(s);
96
```

3.2.2 EliahouNumber (for numerical semigroup)

▷ EliahouNumber(*S*) (attribute)
▷ TruncatedWilfNumberOfNumericalSemigroup(*S*) (attribute)

S is a numerical semigroup. Let c, m, s and l be the conductor, multiplicity, number of generators smaller than c, and number of elements smaller than c in S, respectively. Let q and r be the quotient and nonpositive remainder of the division of c by m, that is, $c=qm-r$. Returns $sl-qd_q+r$, where d_q corresponds with the number of integers in $[c,c+m[$ that are not minimal generators of S.

Example

```
gap> s:=NumericalSemigroup(5,7,9);;
gap> TruncatedWilfNumberOfNumericalSemigroup(s);
4
gap> s:=NumericalSemigroupWithGivenElementsAndFrobenius([14,22,23],55);;
gap> EliahouNumber(s);
-1
```

3.2.3 ProfileOfNumericalSemigroup

▷ ProfileOfNumericalSemigroup(*S*) (attribute)

S is a numerical semigroup. Let c and m be the conductor and multiplicity of S, respectively. Let q and r be the quotient and nonpositive remainder of the division of c by m, that is, $c=qm-r$. Returns a list of lists of integers, each list is the cardinality of $S\cap[jm-r,(j+1)m-r[$ with j in [1..q-1].

Example

```
gap> s:=NumericalSemigroup(5,7,9);;
gap> ProfileOfNumericalSemigroup(s);
[ 2, 1 ]
gap> s:=NumericalSemigroupWithGivenElementsAndFrobenius([14,22,23],55);;
gap> ProfileOfNumericalSemigroup(s);
[ 3, 0, 0 ]
```

3.2.4 EliahouSlicesOfNumericalSemigroup

▷ EliahouSlicesOfNumericalSemigroup(*S*) (attribute)

S is a numerical semigroup. Let c and m be the conductor and multiplicity of S, respectively. Let q and r be the quotient and nonpositive remainder of the division of c by m, that is, $c = qm - r$. Returns a list of lists of integers, each list is the set $S \cap [jm - r, (j+1)m - r[$ with j in [1..q]. So this is a partition of the set of small elements of S (without 0 and c).

Example

```
gap> s:=NumericalSemigroup(5,7,9);;
gap> EliahouSlicesOfNumericalSemigroup(s);
[ [ 5, 7 ], [ 9, 10, 12 ] ]
gap> SmallElements(s);
[ 0, 5, 7, 9, 10, 12, 14 ]
```

Chapter 4

Presentations of Numerical Semigroups

In this chapter we explain how to compute a minimal presentation of a numerical semigroup. Recall that a minimal presentation is a minimal generating system of the kernel congruence of the factorization map of the numerical semigroup. If S is a numerical semigroup minimally generated by $\{n_1,\dots,n_e\}$, then the factorization map is the epimorphism $\varphi : \mathbb{N}^e \to S$, $(x_1,\dots,x_e) \mapsto x_1n_1+\dots+x_en_e$; its kernel is the congruence $\{(a,b) \mid \varphi(a)=\varphi(b)\}$.

The set of minimal generators is stored in a set, and so it may not be arranged as the user gave them. This may affect the arrangement of the coordinates of the pairs in a minimal presentation, since every coordinate is associated to a minimal generator.

4.1 Presentations of Numerical Semigroups

In this section we provide a way to compute minimal presentations of a numerical semigroup. These presentations are constructed from some special elelements in the semigroup (Betti elemenents) whose associated graphs are nonconnected. A generalization of these graphs are the simplicial complexes called shaded sets of an element.

4.1.1 MinimalPresentationOfNumericalSemigroup

▷ `MinimalPresentationOfNumericalSemigroup(S)` (function)
▷ `MinimalPresentation(S)` (operation)

S is a numerical semigroup. The output is a list of lists with two elements. Each list of two elements represents a relation between the minimal generators of the numerical semigroup. If $\{\{x_1,y_1\},\dots,\{x_k,y_k\}\}$ is the output and $\{m_1,\dots,m_n\}$ is the minimal system of generators of the numerical semigroup, then $\{x_i,y_i\} = \{\{a_{i_1},\dots,a_{i_n}\},\{b_{i_1},\dots,b_{i_n}\}\}$ and $a_{i_1}m_1+\cdots+a_{i_n}m_n = b_{i_1}m_1+\cdots+b_{i_n}m_n$.

Any other relation among the minimal generators of the semigroup can be deduced from the ones given in the output.

The algorithm implemented is described in [Ros96a] (see also [RGS99a]).

Example
```
gap> s:=NumericalSemigroup(3,5,7);
<Numerical semigroup with 3 generators>
gap> MinimalPresentationOfNumericalSemigroup(s);
```

```
[ [ [ 0, 2, 0 ], [ 1, 0, 1 ] ], [ [ 3, 1, 0 ], [ 0, 0, 2 ] ],
[ [ 4, 0, 0 ], [ 0, 1, 1 ] ] ]
```

The first element in the list means that $1 \times 3 + 1 \times 7 = 2 \times 5$, and the others have similar meanings.

4.1.2 GraphAssociatedToElementInNumericalSemigroup

▷ `GraphAssociatedToElementInNumericalSemigroup(n, S)` (function)

S is a numerical semigroup and n is an element in S.

The output is a pair. If $\{m_1, \ldots, m_n\}$ is the set of minimal generators of S, then the first component is the set of vertices of the graph associated to n in S, that is, the set $\{m_i \mid n - m_i \in S\}$, and the second component is the set of edges of this graph, that is, $\{\{m_i, m_j\} \mid n - (m_i + m_j) \in S\}$.

This function is used to compute a minimal presentation of the numerical semigroup S, as explained in [Ros96a].

Example

```
gap> s:=NumericalSemigroup(3,5,7);;
gap> GraphAssociatedToElementInNumericalSemigroup(10,s);
[ [ 3, 5, 7 ], [ [ 3, 7 ] ] ]
```

4.1.3 BettiElementsOfNumericalSemigroup

▷ `BettiElementsOfNumericalSemigroup(S)` (function)
▷ `BettiElements(S)` (operation)

S is a numerical semigroup.

The output is the set of elements in S whose associated graph is nonconnected [GSO10].

Example

```
gap> s:=NumericalSemigroup(3,5,7);;
gap> BettiElementsOfNumericalSemigroup(s);
[ 10, 12, 14 ]
```

4.1.4 DegreesOfPrimitiveElementsOfNumericalSemigroup

▷ `DegreesOfPrimitiveElementsOfNumericalSemigroup(S)` (function)

S is a numerical semigroup.

The output is the set of elements s in S such that there exists a minimal solution to $msg \cdot x - msg \cdot y = 0$, such that x, y are factorizations of s, and msg is the minimal generating system of S. Betti elements are primitive, but not the way around in general.

Example

```
gap> s:=NumericalSemigroup(3,5,7);;
gap> DegreesOfPrimitiveElementsOfNumericalSemigroup(s);
[ 3, 5, 7, 10, 12, 14, 15, 21, 28, 35 ]
```

4.1.5 ShadedSetOfElementInNumericalSemigroup

▷ ShadedSetOfElementInNumericalSemigroup(*n*, *S*) (function)

S is a numerical semigroup and *n* is an element in *S*.

The output is a simplicial complex C. If $\{m_1,\ldots,m_n\}$ is the set of minimal generators of *S*, then $L\in C$ if $n-\sum_{i\in L}m_i\in S$ ([SW86]).

This function is a generalization of the graph associated to *n*.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> ShadedSetOfElementInNumericalSemigroup(10,s);
[ [  ], [ 3 ], [ 3, 7 ], [ 5 ], [ 7 ] ]
```

4.2 Uniquely Presented Numerical Semigroups

A numerical semigroup S is uniquely presented if for any two minimal presentations σ and τ and any $(a,b)\in\sigma$, either $(a,b)\in\tau$ or $(b,a)\in\tau$, that is, there is essentially a unique minimal presentation (up to arrangement of the components of the pairs in it).

4.2.1 IsUniquelyPresented

▷ IsUniquelyPresented(*S*) (property)

▷ IsUniquelyPresentedNumericalSemigroup(*S*) (property)

S is a numerical semigroup.

The output is true if *S* has uniquely presented. The implementation is based on [GSO10].

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> IsUniquelyPresentedNumericalSemigroup(s);
true
```

4.2.2 IsGeneric (for numerical semigroups)

▷ IsGeneric(*S*) (property)

▷ IsGenericNumericalSemigroup(*S*) (property)

S is a numerical semigroup.

The output is true if *S* has a generic presentation, that is, in every minimal relation all generators occur. These semigroups are uniquely presented (see [BGSG11]).

This filter implies IsUniquelyPresentedNumericalSemigroup (4.2.1).

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> IsGenericNumericalSemigroup(s);
true
```

Chapter 5

Constructing numerical semigroups from others

This chapter provides several functions to construct numerical semigroups from others (via intersections, quotients by an integer, removing or adding integers, etc.).

5.1 Adding and removing elements of a numerical semigroup

In this section we show how to construct new numerical semigroups from a given numerical semigroup. Two dual operations are presented. The first one removes a minimal generator from a numerical semigroup. The second adds a special gap to a semigroup (see [RGSGGJM03]).

5.1.1 RemoveMinimalGeneratorFromNumericalSemigroup

▷ RemoveMinimalGeneratorFromNumericalSemigroup(*n*, *S*) (function)

S is a numerical semigroup and *n* is one if its minimal generators.

The output is the numerical semigroup $S \setminus \{n\}$ (see [RGSGGJM03]; $S \setminus \{n\}$ is a numerical semigroup if and only if n is a minimal generator of S).

Example

```
gap> s:=NumericalSemigroup(3,5,7);
<Numerical semigroup with 3 generators>
gap> RemoveMinimalGeneratorFromNumericalSemigroup(7,s);
<Numerical semigroup with 3 generators>
gap> MinimalGeneratingSystemOfNumericalSemigroup(last);
[ 3, 5 ]
```

5.1.2 AddSpecialGapOfNumericalSemigroup

▷ AddSpecialGapOfNumericalSemigroup(*g*, *S*) (function)

S is a numerical semigroup and *g* is a special gap of *S*.

The output is the numerical semigroup $S \cup \{g\}$ (see [RGSGGJM03], where it is explained why this set is a numerical semigroup).

Example

```
gap> s:=NumericalSemigroup(3,5,7);;
gap> s2:=RemoveMinimalGeneratorFromNumericalSemigroup(5,s);
<Numerical semigroup with 3 generators>
gap> s3:=AddSpecialGapOfNumericalSemigroup(5,s2);
<Numerical semigroup>
gap> SmallElementsOfNumericalSemigroup(s) =
> SmallElementsOfNumericalSemigroup(s3);
true
gap> s=s3;
true
```

5.2 Intersections, and quotients and multiples by integers

We provide functions to build numerical semigroups from others by means of intersections, quotients, multiples and related constructions.

5.2.1 Intersection (for numerical semigroups)

▷ Intersection(*S*, *T*) (operation)
▷ IntersectionOfNumericalSemigroups(*S*, *T*) (function)

S and *T* are numerical semigroups. Computes the intersection of *S* and *T* (which is a numerical semigroup).

Example

```
gap> S := NumericalSemigroup("modular", 5,53);
<Modular numerical semigroup satisfying 5x mod 53 <= x >
gap> T := NumericalSemigroup(2,17);
<Numerical semigroup with 2 generators>
gap> SmallElements(S);
[ 0, 11, 12, 13, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 37, 38, 39, 43 ]
gap> SmallElements(T);
[ 0, 2, 4, 6, 8, 10, 12, 14, 16 ]
gap> IntersectionOfNumericalSemigroups(S,T);
<Numerical semigroup>
gap> SmallElements(last);
[ 0, 12, 22, 23, 24, 25, 26, 32, 33, 34, 35, 36, 37, 38, 39, 43 ]
```

5.2.2 QuotientOfNumericalSemigroup

▷ QuotientOfNumericalSemigroup(*S*, *n*) (function)
▷ \/(*S*, *n*) (operation)

S is a numerical semigroup and *n* is an integer. Computes the quotient of *S* by *n*, that is, the set $\{x \in \mathbb{N} \mid nx \in S\}$, which is again a numerical semigroup. `S / n` may be used as a short for `QuotientOfNumericalSemigroup(S, n)`.

Example

```
gap> s:=NumericalSemigroup(3,29);
<Numerical semigroup with 2 generators>
```

```
gap> SmallElements(s);
[ 0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 29, 30, 32, 33, 35, 36, 38,
39, 41, 42, 44, 45, 47, 48, 50, 51, 53, 54, 56 ]
gap> t:=QuotientOfNumericalSemigroup(s,7);
<Numerical semigroup>
gap> SmallElements(t);
[ 0, 3, 5, 6, 8 ]
gap> u := s / 7;
<Numerical semigroup>
gap> SmallElements(u);
[ 0, 3, 5, 6, 8 ]
```

5.2.3 MultipleOfNumericalSemigroup

▷ `MultipleOfNumericalSemigroup(S, a, b)` (function)

`S` is a numerical semigroup, and `a` and `b` are positive integers. Computes $aS \cup \{b, b+1, \rightarrow\}$. If `b` is smaller than ac, with c the conductor of S, then a warning is displayed.

Example
```
gap> N:=NumericalSemigroup(1);;
gap> s:=MultipleOfNumericalSemigroup(N,4,20);;
gap> SmallElements(s);
[ 0, 4, 8, 12, 16, 20 ]
```

5.2.4 Difference (for numerical semigroups)

▷ `Difference(S, T)` (operation)
▷ `DifferenceOfNumericalSemigroups(S, T)` (function)

`S, T` are numerical semigroups. The output is the set $S \setminus T$.

Example
```
gap> ns1 := NumericalSemigroup(5,7);;
gap> ns2 := NumericalSemigroup(7,11,12);;
gap> Difference(ns1,ns2);
[ 5, 10, 15, 17, 20, 27 ]
gap> Difference(ns2,ns1);
[ 11, 18, 23 ]
gap> DifferenceOfNumericalSemigroups(ns2,ns1);
[ 11, 18, 23 ]
```

5.2.5 NumericalDuplication

▷ `NumericalDuplication(S, E, b)` (function)

`S` is a numerical semigroup, and `E` and ideal of `S`, and `b` is a positive odd integer, so that $2S \cup (2E+b)$ is a numerical semigroup (this extends slightly the original definition where `b` was imposed to be in `S`, [DS13]; now the condition imposed is $E+E+b \subseteq S$). Computes $2S \cup (2E+b)$.

Example
```
gap> s:=NumericalSemigroup(3,5,7);
<Numerical semigroup with 3 generators>
```

```
gap> e:=6+s;
<Ideal of numerical semigroup>
gap> ndup:=NumericalDuplication(s,e,3);
<Numerical semigroup with 4 generators>
gap> SmallElements(ndup);
[ 0, 6, 10, 12, 14, 15, 16, 18, 20, 21, 22, 24 ]
```

5.2.6 InductiveNumericalSemigroup

▷ `InductiveNumericalSemigroup(a, b)` (function)

`a` and `b` are lists of positive integers, with k the length of `a` and `b`, and such that $b[i+1] \geq a[i]b[i]$ $(0 \leq i \leq k-1)$. Computes inductively $S_0 = \mathbb{N}$ and $S_{i+1} = a[i]S_i \cup \{a[i]b[i], a[i]b[i]+1, \rightarrow\}$, and returns S_k.

Example
```
gap> s:=InductiveNumericalSemigroup([4,2],[5,23]);;
gap> SmallElements(s);
[ 0, 8, 16, 24, 32, 40, 42, 44, 46 ]
```

5.3 Constructing the set of all numerical semigroups containing a given numerical semigroup

In order to construct the set of numerical semigroups containing a fixed numerical semigroup S, one first constructs its unitary extensions, that is to say, the sets $S \cup \{g\}$ that are numerical semigroups with g a positive integer. This is achieved by constructing the special gaps of the semigroup, and then adding each of them to the numerical semigroup. Then we repeat the process for each of these new numerical semigroups until we reach $\mathbb{N}$.

These procedures are described in [RGSGGJM03].

5.3.1 OverSemigroupsNumericalSemigroup

▷ `OverSemigroupsNumericalSemigroup(s)` (function)

`s` is a numerical semigroup. The output is the set of numerical semigroups containing it.

Example
```
gap> OverSemigroupsNumericalSemigroup(NumericalSemigroup(3,5,7));
[ <The numerical semigroup N>, <Numerical semigroup with 2 generators>,
  <Numerical semigroup with 3 generators>,
  <Numerical semigroup with 3 generators> ]
gap> List(last,s->MinimalGenerators(s));
[ [ 1 ], [ 2, 3 ], [ 3 .. 5 ], [ 3, 5, 7 ] ]
```

5.4 Constructing the set of numerical semigroup with given Frobenius number

Finding the set of all numerical semigroups with a given Frobenius number is not accomplished via over semigroups. In order to achieve this, we use fundamental gaps.

5.4.1 NumericalSemigroupsWithFrobeniusNumber

▷ NumericalSemigroupsWithFrobeniusNumber(*f*) (function)

f is an integer. The output is the set of numerical semigroups with Frobenius number *f*. The algorithm implemented is given in [RGSGGJM04].

Example
```
gap> Length(NumericalSemigroupsWithFrobeniusNumber(15));
200
```

5.5 Constructing the set of numerical semigroups with genus g

Given a numerical semigroup of genus g (that is, with exactly g gaps), removing minimal generators, one obtains numerical semigroups of genus g+1. In order to avoid repetitions, we only remove minimal generators greater than the Frobenius number of the numerical semigroup (this is accomplished with the local function sons).

These procedures are described in [RGSGGB03] and [BA08].

5.5.1 NumericalSemigroupsWithGenus

▷ NumericalSemigroupsWithGenus(*g*) (function)

g is a nonnegative integer. The output is the set of numerical semigroups with genus *g*.

Example
```
gap> NumericalSemigroupsWithGenus(5);
[ <Numerical semigroup with 6 generators>,
  <Numerical semigroup with 5 generators>,
  <Numerical semigroup with 5 generators>,
  <Numerical semigroup with 5 generators>,
  <Numerical semigroup with 5 generators>,
  <Numerical semigroup with 4 generators>,
  <Numerical semigroup with 4 generators>,
  <Numerical semigroup with 4 generators>,
  <Numerical semigroup with 4 generators>,
  <Numerical semigroup with 3 generators>,
  <Numerical semigroup with 3 generators>,
  <Numerical semigroup with 2 generators> ]
gap> List(last,MinimalGenerators);
[ [ 6 .. 11 ], [ 5, 7, 8, 9, 11 ], [ 5, 6, 8, 9 ], [ 5, 6, 7, 9 ],
  [ 5, 6, 7, 8 ], [ 4, 6, 7 ], [ 4, 7, 9, 10 ], [ 4, 6, 9, 11 ],
  [ 4, 5, 11 ], [ 3, 8, 10 ], [ 3, 7, 11 ], [ 2, 11 ] ]
```

5.6 Constructing the set of numerical semigroups with a given set of pseudo-Frobenius numbers

Refer to PseudoFrobeniusOfNumericalSemigroup (3.1.22).

These procedures are described in [DGSRP16], and are used to find the set of numerical semigroups with a prescribed set of pseudo-Frobenius numbers.

5.6.1 ForcedIntegersForPseudoFrobenius

▷ ForcedIntegersForPseudoFrobenius(*PF*) (function)

PF is a list of positive integers (given as a list or individual elements). The output is:

- in case there exists a numerical semigroup S such that $PF(S) = PF$:
  - a list $[forced\_gaps, forced\_elts]$ such that:
    - $forced\_gaps$ is contained in $\mathbb{N} - S$ for any numerical semigroup S such that $PF(S) = \{g\_1, \ldots, g\_n\}$
    - forced_elts is contained in S for any numerical semigroup S such that $PF(S) = \{g\_1, \ldots, g\_n\}$
- "fail" in case it is found some condition that fails.

Example
```
gap> pf := [ 58, 64, 75 ];
[ 58, 64, 75 ]
gap> ForcedIntegersForPseudoFrobenius(pf);
[ [ 1, 2, 3, 4, 5, 6, 7, 8, 11, 15, 16, 17, 25, 29, 32, 58, 64, 75 ],
  [ 0, 59, 60, 67, 68, 69, 70, 71, 72, 73, 74, 76 ] ]
```

5.6.2 SimpleForcedIntegersForPseudoFrobenius

▷ SimpleForcedIntegersForPseudoFrobenius(*fg, fe, PF*) (function)

Is just a quicker version of ForcedIntegersForPseudoFrobenius (5.6.1)

fg is a list of integers that we require to be gaps of the semigroup; *fe* is a list of integers that we require to be elements of the semigroup; *PF* is a list of positive integers. The output is:

- in case there exists a numerical semigroup S such that $PF(S) = PF$:
  - a list $[forced\_gaps, forced\_elts]$ such that:
    - $forced\_gaps$ is contained in $\mathbb{N} - S$ for any numerical semigroup S such that $PF(S) = \{g\_1, \ldots, g\_n\}$
    - forced_elts is contained in S for any numerical semigroup S such that $PF(S) = \{g\_1, \ldots, g\_n\}$
- "fail" in case it is found some condition that fails.

Example
```
gap> pf := [ 15, 20, 27, 35 ];;
gap> fint := ForcedIntegersForPseudoFrobenius(pf);
[ [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 16, 20, 27, 35 ],
  [ 0, 19, 23, 25, 26, 28, 29, 30, 31, 32, 33, 34, 36 ] ]
gap> free := Difference([1..Maximum(pf)],Union(fint));
[ 11, 13, 14, 17, 18, 21, 22, 24 ]
gap> SimpleForcedIntegersForPseudoFrobenius(fint[1],Union(fint[2],[free[1]]),pf);
[ [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 15, 16, 20, 24, 27, 35 ],
  [ 0, 11, 19, 22, 23, 25, 26, 28, 29, 30, 31, 32, 33, 34, 36 ] ]
```

5.6.3 NumericalSemigroupsWithPseudoFrobeniusNumbers

▷ `NumericalSemigroupsWithPseudoFrobeniusNumbers(PF)` (function)

PF is a list of positive integers (given as a list or individual elements). The output is: a list of numerical semigroups S such that PF(S)=PF. When Length(PF)=1, it makes use of the function `NumericalSemigroupsWithFrobeniusNumber` (5.4.1)

Example

```
gap> pf := [ 58, 64, 75 ];
[ 58, 64, 75 ]
gap> Length(NumericalSemigroupsWithPseudoFrobeniusNumbers(pf));
561
gap> pf := [11,19,22];;
gap> NumericalSemigroupsWithPseudoFrobeniusNumbers(pf);
[ <Numerical semigroup>, <Numerical semigroup>, <Numerical semigroup>,
  <Numerical semigroup>, <Numerical semigroup> ]
gap> List(last,MinimalGenerators);
[ [ 7, 9, 17, 20 ], [ 7, 10, 13, 16, 18 ], [ 9, 12, 14, 15, 16, 17, 20 ],
  [ 10, 13, 14, 15, 16, 17, 18, 21 ],
  [ 12, 13, 14, 15, 16, 17, 18, 20, 21, 23 ] ]
gap> Set(last2,PseudoFrobeniusOfNumericalSemigroup);
[ [ 11, 19, 22 ] ]
g
```

5.6.4 ANumericalSemigroupWithPseudoFrobeniusNumbers

▷ `ANumericalSemigroupWithPseudoFrobeniusNumbers(PF)` (function)

PF is a list of positive integers (given as a list or individual elements). Alternatively, a record with fields "pseudo_frobenius" and "max_attempts" may be given. The output is: A numerical semigroup S such that $PF(S) = PF$. Returns fail if it concludes that it does not exist and suggests to use NumericalSemigroupsWithPseudoFrobeniusNumbers if it is not able to conclude...

When $Length(PF) = 1$ or $Length(PF) = 2$ and $2 * PF[1] = PF[2]$, it makes use of the function `AnIrreducibleNumericalSemigroupWithFrobeniusNumber` (6.1.4).

Example

```
gap> pf := [ 83, 169, 173, 214, 259 ];;
gap> ANumericalSemigroupWithPseudoFrobeniusNumbers(pf);
<Numerical semigroup>
gap> gen := MinimalGeneratingSystem(last);
[ 38, 57, 64, 72, 79, 98, 99, 106, 118, 120, 124, 132, 134, 146, 147, 154,
  165, 168, 179 ]
gap> ns := NumericalSemigroup(gen);
<Numerical semigroup with 19 generators>
gap> PseudoFrobeniusOfNumericalSemigroup(ns);
[ 83, 169, 173, 214, 259 ]
```

Chapter 6

Irreducible numerical semigroups

An irreducible numerical semigroup is a semigroup that cannot be expressed as the intersection of numerical semigroups properly containing it.

It is not difficult to prove that a semigroup is irreducible if and only if it is maximal (with respect to set inclusion) in the set of all numerical semigroups having its same Frobenius number (see [RB03]). Hence, according to [FGR87] (respectively [BDF97]), symmetric (respectively pseudo-symmetric) numerical semigroups are those irreducible numerical semigroups with odd (respectively even) Frobenius number.

In [RGSGGJM03] it is shown that a nontrivial numerical semigroup is irreducible if and only if it has only one special gap. We use this characterization.

In old versions of the package, we first constructed an irreducible numerical semigroup with the given Frobenius number (as explained in [RGS04]), and then we constructed the rest from it. The present version uses a faster procedure presented in [BR13].

Every numerical semigroup can be expressed as an intersection of irreducible numerical semigroups. If S can be expressed as $S = S_1 \cap \cdots \cap S_n$, with S_i irreducible numerical semigroups, and no factor can be removed, then we say that this decomposition is minimal. Minimal decompositions can be computed by using Algorithm 26 in [RGSGGJM03].

6.1 Irreducible numerical semigroups

In this section we provide membership tests to the two families that conform the set of irreducible numerical semigroups. We also give a procedure to compute the set of all irreducible numerical semigroups with fixed Frobenius number (or equivalently genus, since for irreducible numerical semigroups once the Frobenius number is fixed, so is the genus). Also we give a function to compute the decomposition of a numerical semigroup as an intersection of irreducible numerical semigroups.

6.1.1 IsIrreducibleNumericalSemigroup

▷ `IsIrreducibleNumericalSemigroup(s)` (property)
▷ `IsIrreducible(s)` (property)

`s` is a numerical semigroup. The output is `true` if `s` is irreducible, `false` otherwise.

This filter implies `IsAlmostSymmetricNumericalSemigroup` (6.3.3) and `IsAcuteNumericalSemigroup` (3.1.28).

Example
```
gap> IsIrreducibleNumericalSemigroup(NumericalSemigroup(4,6,9));
true
gap> IsIrreducibleNumericalSemigroup(NumericalSemigroup(4,6,7,9));
false
```

6.1.2 IsSymmetricNumericalSemigroup

▷ IsSymmetricNumericalSemigroup(*s*) (attribute)
▷ IsSymmetric(*s*) (attribute)

s is a numerical semigroup. The output is `true` if *s* is symmetric, `false` otherwise.
This filter implies `IsIrreducibleNumericalSemigroup` (6.1.1).

Example
```
gap> IsSymmetric(NumericalSemigroup(10,23));
true
gap> IsSymmetricNumericalSemigroup(NumericalSemigroup(10,11,23));
false
```

6.1.3 IsPseudoSymmetric (for numerical semigroups)

▷ IsPseudoSymmetric(*s*) (property)
▷ IsPseudoSymmetricNumericalSemigroup(*s*) (property)

s is a numerical semigroup. The output is `true` if *s* is pseudo-symmetric, `false` otherwise.
This filter implies `IsIrreducibleNumericalSemigroup` (6.1.1).

Example
```
gap> IsPseudoSymmetricNumericalSemigroup(NumericalSemigroup(6,7,8,9,11));
true
gap> IsPseudoSymmetricNumericalSemigroup(NumericalSemigroup(4,6,9));
false
```

6.1.4 AnIrreducibleNumericalSemigroupWithFrobeniusNumber

▷ AnIrreducibleNumericalSemigroupWithFrobeniusNumber(*f*) (function)

f is an integer. When $f = 0$ or $f \leq -2$, the output is `fail`. Otherwise, the output is an irreducible numerical semigroup with Frobenius number *f*. From the way the procedure is implemented, the resulting semigroup has at most four generators (see [RGS04]).

Example
```
gap> s := AnIrreducibleNumericalSemigroupWithFrobeniusNumber(28);
<Numerical semigroup with 3 generators>
gap> MinimalGenerators(s);
[ 3, 17, 31 ]
gap> FrobeniusNumber(s);
28
```

6.1.5 IrreducibleNumericalSemigroupsWithFrobeniusNumber

▷ IrreducibleNumericalSemigroupsWithFrobeniusNumber(*f*) (function)

f is an integer. The output is the set of all irreducible numerical semigroups with Frobenius number *f*.

Example
```
gap> Length(IrreducibleNumericalSemigroupsWithFrobeniusNumber(19));
20
```

6.1.6 DecomposeIntoIrreducibles (for numerical semigroup)

▷ DecomposeIntoIrreducibles(*s*) (function)

s is a numerical semigroup. The output is a set of irreducible numerical semigroups containing it. These elements appear in a minimal decomposition of *s* as intersection into irreducibles.

Example
```
gap> DecomposeIntoIrreducibles(NumericalSemigroup(5,6,8));
[ <Numerical semigroup with 3 generators>,
  <Numerical semigroup with 4 generators> ]
```

6.2 Complete intersection numerical semigroups

The cardinality of a minimal presentation of a numerical semigroup is always greater than or equal to its embedding dimension minus one. Complete intersection numerical semigroups are numerical semigroups reaching this bound, and they are irreducible. It can be shown that every complete intersection (other that $\mathbb{N}$) is a complete intersection if and only if it is the gluing of two complete intersections. When in this gluing, one of the copies is isomorphic to $\mathbb{N}$, then we obtain a free semigroup in the sense of [BC77]. Two special kinds of free semigroups are telescopic semigroups ([KP95]) and those associated to an irreducible planar curve ([Zar86]). We use the algorithms presented in [AGS13] to find the set of all complete intersections (also free, telescopic and associated to irreducible planar curves) numerical semigroups with given Frobenius number.

6.2.1 AsGluingOfNumericalSemigroups

▷ AsGluingOfNumericalSemigroups(*s*) (function)

s is a numerical semigroup. Returns all partitions $\{A_1,A_2\}$ of the minimal generating set of *s* such that *s* is a gluing of $\langle A_1\rangle$ and $\langle A_2\rangle$ by $gcd(A_1)gcd(A_2)$.

Example
```
gap> s := NumericalSemigroup( 10, 15, 16 );
<Numerical semigroup with 3 generators>
gap> AsGluingOfNumericalSemigroups(s);
[ [ [ 10, 15 ], [ 16 ] ], [ [ 10, 16 ], [ 15 ] ] ]
gap> s := NumericalSemigroup( 18, 24, 34, 46, 51, 61, 74, 8 );
<Numerical semigroup with 8 generators>
gap> AsGluingOfNumericalSemigroups(s);
[  ]
```

6.2.2 IsCompleteIntersection

▷ IsCompleteIntersection(*s*) (property)
▷ IsACompleteIntersectionNumericalSemigroup(*s*) (property)

s is a numerical semigroup. The output is true if the numerical semigroup is a complete intersection, that is, the cardinality of a (any) minimal presentation equals its embedding dimension minus one.

This filter implies IsSymmetricNumericalSemigroup (6.1.2) and IsCyclotomicNumericalSemigroup (10.1.8).

Example
```
gap> s := NumericalSemigroup( 10, 15, 16 );
<Numerical semigroup with 3 generators>
gap> IsACompleteIntersectionNumericalSemigroup(s);
true
gap> s := NumericalSemigroup( 18, 24, 34, 46, 51, 61, 74, 8 );
<Numerical semigroup with 8 generators>
gap> IsACompleteIntersectionNumericalSemigroup(s);
false
```

6.2.3 CompleteIntersectionNumericalSemigroupsWithFrobeniusNumber

▷ CompleteIntersectionNumericalSemigroupsWithFrobeniusNumber(*f*) (function)

f is an integer. The output is the set of all complete intersection numerical semigroups with Frobenius number *f*.

Example
```
gap> Length(CompleteIntersectionNumericalSemigroupsWithFrobeniusNumber(57));
34
```

6.2.4 IsFree

▷ IsFree(*s*) (property)
▷ IsFreeNumericalSemigroup(*s*) (property)

s is a numerical semigroup. The output is true if the numerical semigroup is free in the sense of [BC77]: it is either $\mathbb{N}$ or the gluing of a copy of $\mathbb{N}$ with a free numerical semigroup.

This filter implies IsACompleteIntersectionNumericalSemigroup (6.2.2).

Example
```
gap> IsFreeNumericalSemigroup(NumericalSemigroup(10,15,16));
true
gap> IsFreeNumericalSemigroup(NumericalSemigroup(3,5,7));
false
```

6.2.5 FreeNumericalSemigroupsWithFrobeniusNumber

▷ FreeNumericalSemigroupsWithFrobeniusNumber(*f*) (function)

f is an integer. The output is the set of all free numerical semigroups with Frobenius number *f*.

Example
```
gap> Length(FreeNumericalSemigroupsWithFrobeniusNumber(57));
33
```

6.2.6 IsTelescopic

▷ IsTelescopic(s) (property)
▷ IsTelescopicNumericalSemigroup(s) (property)

s is a numerical semigroup. The output is true if the numerical semigroup is telescopic in the sense of [KP95]: it is either $\mathbb{N}$ or the gluing of $\langle n_e \rangle$ and $s' = \langle n_1/d, \ldots, n_{e-1}/d \rangle$, and s' is again a telescopic numerical semigroup, where $n_1 < \cdots < n_e$ are the minimal generators of s.

This filter implies IsAperySetBetaRectangular (6.2.11) and IsFreeNumericalSemigroup (6.2.4).

Example
```
gap> IsTelescopicNumericalSemigroup(NumericalSemigroup(4,11,14));
false
gap> IsFreeNumericalSemigroup(NumericalSemigroup(4,11,14));
true
```

6.2.7 TelescopicNumericalSemigroupsWithFrobeniusNumber

▷ TelescopicNumericalSemigroupsWithFrobeniusNumber(f) (function)

f is an integer. The output is the set of all telescopic numerical semigroups with Frobenius number f.

Example
```
gap> Length(TelescopicNumericalSemigroupsWithFrobeniusNumber(57));
20
```

6.2.8 IsNumericalSemigroupAssociatedIrreduciblePlanarCurveSingularity

▷ IsNumericalSemigroupAssociatedIrreduciblePlanarCurveSingularity(s) (property)

s is a numerical semigroup. The output is true if the numerical semigroup is associated to an irreducible planar curve singularity ([Zar86]). These semigroups are telescopic.

This filter implies IsAperySetAlphaRectangular (6.2.12) and IsTelescopicNumericalSemigroup (6.2.6).

Example
```
gap> ns := NumericalSemigroup(4,11,14);;
gap> IsNumericalSemigroupAssociatedIrreduciblePlanarCurveSingularity(ns);
false
gap> ns := NumericalSemigroup(4,11,19);;
gap> IsNumericalSemigroupAssociatedIrreduciblePlanarCurveSingularity(ns);
true
```

6.2.9 NumericalSemigroupsPlanarSingularityWithFrobeniusNumber

▷ `NumericalSemigroupsPlanarSingularityWithFrobeniusNumber(f)` (function)

`f` is an integer. The output is the set of all numerical semigroups associated to irreducible planar curves singularities with Frobenius number `f`.

Example
```
gap> Length(NumericalSemigroupsPlanarSingularityWithFrobeniusNumber(57));
7
```

6.2.10 IsAperySetGammaRectangular

▷ `IsAperySetGammaRectangular(S)` (function)

`S` is a numerical semigroup.

Test for the γ-rectangularity of the Apéry Set of a numerical semigroup. This test is the implementation of the algorithm given in [DMS14]. Numerical Semigroups with this property are free and thus complete intersections.

This filter implies `IsFreeNumericalSemigroup` (6.2.4).

Example
```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> IsAperySetGammaRectangular(s);
false
gap> s:=NumericalSemigroup(4,6,11);;
gap> IsAperySetGammaRectangular(s);
true
```

6.2.11 IsAperySetBetaRectangular

▷ `IsAperySetBetaRectangular(S)` (function)

`S` is a numerical semigroup.

Test for the β-rectangularity of the Apéry Set of a numerical semigroup. This test is the implementation of the algorithm given in [DMS14]; β-rectangularity implies γ-rectangularity.

This filter implies `IsAperySetGammaRectangular` (6.2.10).

Example
```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> IsAperySetBetaRectangular(s);
false
gap> s:=NumericalSemigroup(4,6,11);;
gap> IsAperySetBetaRectangular(s);
true
```

6.2.12 IsAperySetAlphaRectangular

▷ `IsAperySetAlphaRectangular(S)` (function)

S is a numerical semigroup.

Test for the α-rectangularity of the Apéry Set of a numerical semigroup. This test is the implementation of the algorithm given in [DMS14]; α-rectangularity implies β-rectangularity.

This filter implies `IsAperySetBetaRectangular` (6.2.11).

Example
```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> IsAperySetAlphaRectangular(s);
false
gap> s:=NumericalSemigroup(4,6,11);;
gap> IsAperySetAlphaRectangular(s);
true
```

6.3 Almost-symmetric numerical semigroups

A numerical semigroup is almost-symmetric ([BF97]) if its genus is the arithmetic mean of its Frobenius number and type. We use a procedure presented in [RGS14] to determine the set of all almost-symmetric numerical semigroups with given Frobenius number. In order to do this, we first calculate the set of all almost-symmetric numerical semigroups that can be constructed from an irreducible numerical semigroup.

6.3.1 AlmostSymmetricNumericalSemigroupsFromIrreducible

▷ `AlmostSymmetricNumericalSemigroupsFromIrreducible(s)` (function)

`s` is an irreducible numerical semigroup. The output is the set of almost-symmetric numerical semigroups that can be constructed from `s` by removing some of its generators (as explained in [RGS14]).

Example
```
gap> ns := NumericalSemigroup(5,8,9,11);;
gap> AlmostSymmetricNumericalSemigroupsFromIrreducible(ns);
[ <Numerical semigroup with 4 generators>,
  <Numerical semigroup with 5 generators>,
  <Numerical semigroup with 5 generators> ]
gap> List(last,MinimalGeneratingSystemOfNumericalSemigroup);
[ [ 5, 8, 9, 11 ], [ 5, 8, 11, 14, 17 ], [ 5, 9, 11, 13, 17 ] ]
```

6.3.2 AlmostSymmetricNumericalSemigroupsFromIrreducibleAndGivenType

▷ `AlmostSymmetricNumericalSemigroupsFromIrreducibleAndGivenType(s, t)` (function)

`s` is an irreducible numerical semigroup and `t` is a positive integer. The output is the set of almost-symmetric numerical semigroups with type `t` that can be constructed from `s` by removing some of its generators (as explained in [BOR18]).

Example
```
gap> ns := NumericalSemigroup(5,8,9,11);;
gap> AlmostSymmetricNumericalSemigroupsFromIrreducibleAndGivenType(ns,4);
[ <Numerical semigroup with 5 generators>,
  <Numerical semigroup with 5 generators> ]
```

```
gap> List(last,MinimalGenerators);
[ [ 5, 8, 11, 14, 17 ], [ 5, 9, 11, 13, 17 ] ]
```

6.3.3 IsAlmostSymmetric

▷ IsAlmostSymmetric(*s*) (function)
▷ IsAlmostSymmetricNumericalSemigroup(*s*) (function)

s is a numerical semigroup. The output is `true` if the numerical semigroup is almost symmetric.

Example
```
gap> IsAlmostSymmetricNumericalSemigroup(NumericalSemigroup(5,8,11,14,17));
true
```

6.3.4 AlmostSymmetricNumericalSemigroupsWithFrobeniusNumber

▷ AlmostSymmetricNumericalSemigroupsWithFrobeniusNumber(*f*[, *ts*]) (function)

f is an integer, and so is *ts*. The output is the set of all almost symmetric numerical semigroups with Frobenius number *f*, and type greater than or equal to *ts*. If *ts* is not specified, then it is considered to be equal to one, and so the output is the set of all almost symmetric numerical semigroups with Frobenius number *f*.

Example
```
gap> Length(AlmostSymmetricNumericalSemigroupsWithFrobeniusNumber(12));
15
gap> Length(IrreducibleNumericalSemigroupsWithFrobeniusNumber(12));
2
gap> List(AlmostSymmetricNumericalSemigroupsWithFrobeniusNumber(12,4),Type);
[ 12, 10, 8, 8, 6, 6, 6, 6, 4, 4, 4, 4, 4 ]
```

6.3.5 AlmostSymmetricNumericalSemigroupsWithFrobeniusNumberAndType

▷ AlmostSymmetricNumericalSemigroupsWithFrobeniusNumberAndType(*f*, *t*) (function)

f is an integer and so is *t*. The output is the set of all almost symmetric numerical semigroups with Frobenius number *f* and type *t*.

Example
```
gap> Length(AlmostSymmetricNumericalSemigroupsWithFrobeniusNumberAndType(12,4));
5
```

Chapter 7

Ideals of numerical semigroups

Let S be a numerical semigroup. A set I of integers is an *ideal relative* to a numerical semigroup S provided that $I+S \subseteq I$ and that there exists $d \in S$ such that $d+I \subseteq S$.

If $\{i_1,\dots,i_k\}$ is a subset of $\mathbb{Z}$, then the set $I = \{i_1,\dots,i_k\}+S = \bigcup_{n=1}^{k} i_n + S$ is an ideal relative to S, and $\{i_1,\dots,i_k\}$ is a system of generators of I. A system of generators M is minimal if no proper subset of M generates the same ideal. Usually, ideals are specified by means of its generators and the ambient numerical semigroup to which they are ideals (for more information see for instance [BDF97]).

7.1 Definitions and basic operations

We describe in this section the basic functions to create and compute notable elements of ideals of numerical semigroups. We also include iterators and functions to treat ideals as lists, which easies the access to its elements.

7.1.1 IdealOfNumericalSemigroup

▷ IdealOfNumericalSemigroup(*l*, *S*) (function)
▷ +(*l*, *S*) (function)

S is a numerical semigroup and *l* a list of integers. The output is the ideal of *S* generated by *l*. There are several shortcuts for this function, as shown in the example.

Example
```
gap> IdealOfNumericalSemigroup([3,5],NumericalSemigroup(9,11));
<Ideal of numerical semigroup>
gap> [3,5]+NumericalSemigroup(9,11);
<Ideal of numerical semigroup>
gap> last=last2;
true
gap> 3+NumericalSemigroup(5,9);
<Ideal of numerical semigroup>
```

7.1.2 IsIdealOfNumericalSemigroup

▷ IsIdealOfNumericalSemigroup(*Obj*) (function)

Tests if the object *Obj* is an ideal of a numerical semigroup.

Example
```
gap> I:=[1..7]+NumericalSemigroup(7,19);;
gap> IsIdealOfNumericalSemigroup(I);
true
gap> IsIdealOfNumericalSemigroup(2);
false
```

7.1.3 MinimalGenerators (for ideal of numerical semigroup)

▷ MinimalGenerators(*I*) (attribute)
▷ MinimalGeneratingSystem(*I*) (attribute)
▷ MinimalGeneratingSystemOfIdealOfNumericalSemigroup(*I*) (attribute)

I is an ideal of a numerical semigroup. The output is the minimal system of generators of *I*.

Example
```
gap> I:=[3,5,9]+NumericalSemigroup(2,11);;
gap> MinimalGeneratingSystemOfIdealOfNumericalSemigroup(I);
[ 3 ]
gap> MinimalGeneratingSystem(I);
[ 3 ]
gap> MinimalGenerators([3,5]+NumericalSemigroup(2,11));
[ 3 ]
```

7.1.4 Generators (for ideal of numerical semigroup)

▷ Generators(*I*) (attribute)
▷ GeneratorsOfIdealOfNumericalSemigroup(*I*) (attribute)

I is an ideal of a numerical semigroup. The output is a system of generators of the ideal.

Remark: from Version 1.0.1 on, this value does not change even when a set of minimal generators is computed.

Example
```
gap> I:=[3,5,9]+NumericalSemigroup(2,11);;
gap> GeneratorsOfIdealOfNumericalSemigroup(I);
[ 3, 5, 9 ]
gap> Generators(I);
[ 3, 5, 9 ]
```

7.1.5 AmbientNumericalSemigroupOfIdeal

▷ AmbientNumericalSemigroupOfIdeal(*I*) (function)

I is an ideal of a numerical semigroup, say S. The output is S.

Example
```
gap> I:=[3,5,9]+NumericalSemigroup(2,11);;
gap> AmbientNumericalSemigroupOfIdeal(I);
<Numerical semigroup with 2 generators>
```

7.1.6 IsIntegral

▷ IsIntegral(*I*) (property)
▷ IsIntegralIdealOfNumericalSemigroup(*I*) (property)

I is an ideal of a numerical semigroup, say S. Detects if $I \subseteq S$.

Example
```
gap> s:=NumericalSemigroup(3,7,5);;
gap> IsIntegralIdealOfNumericalSemigroup(4+s);
false
gap> IsIntegralIdealOfNumericalSemigroup(10+s);
true
gap> IsIntegral(10+s);
true
```

7.1.7 SmallElements (for ideal of numerical semigroup)

▷ SmallElements(*I*) (function)
▷ SmallElementsOfIdealOfNumericalSemigroup(*I*) (function)

I is an ideal of a numerical semigroup. The output is a list with the elements in *I* that are less than or equal to the greatest integer not belonging to the ideal plus one.

Example
```
gap> I:=[3,5,9]+NumericalSemigroup(2,11);;
gap> SmallElementsOfIdealOfNumericalSemigroup(I);
[ 3, 5, 7, 9, 11, 13 ]
gap> SmallElements(I) = SmallElementsOfIdealOfNumericalSemigroup(I);
true
gap> J:=[2,11]+NumericalSemigroup(2,11);;
gap> SmallElementsOfIdealOfNumericalSemigroup(J);
[ 2, 4, 6, 8, 10 ]
```

7.1.8 Conductor (for ideal of numerical semigroup)

▷ Conductor(*NS*) (attribute)
▷ ConductorOfIdealOfNumericalSemigroup(*I*) (function)

I is an ideal of a numerical semigroup. The output is the largest element in *SmallElements(I)*.

Example
```
gap> s:=NumericalSemigroup(3,7,5);;
gap> ConductorOfIdealOfNumericalSemigroup(10+s);
15
gap> Conductor(10+s);
15
```

7.1.9 Minimum (minimum of ideal of numerical semigroup)

▷ Minimum(*I*) (operation)

I is an ideal of a numerical semigroup. The output is the minimum of *I*.

Example
```
gap> J:=[2,11]+NumericalSemigroup(2,11);;
gap> Minimum(J);
2
```

7.1.10 BelongsToIdealOfNumericalSemigroup

▷ BelongsToIdealOfNumericalSemigroup(*n*, *I*) (function)
▷ \in(*n*, *I*) (operation)

I is an ideal of a numerical semigroup, *n* is an integer. The output is `true` if *n* belongs to *I*. *n* `in` *I* can be used for short.

Example
```
gap> J:=[2,11]+NumericalSemigroup(2,11);;
gap> BelongsToIdealOfNumericalSemigroup(9,J);
false
gap> 9 in J;
false
gap> BelongsToIdealOfNumericalSemigroup(10,J);
true
gap> 10 in J;
true
```

7.1.11 ElementNumber_IdealOfNumericalSemigroup

▷ ElementNumber_IdealOfNumericalSemigroup(*I*, *r*) (function)

I is an ideal of a numerical semigroup and *r* is an integer. It returns the *r*-th element of *I*.

Example
```
gap> I := [2,5]+ NumericalSemigroup(7,8,17);;
gap> ElementNumber_IdealOfNumericalSemigroup(I,10);
19
```

7.1.12 NumberElement_IdealOfNumericalSemigroup

▷ NumberElement_IdealOfNumericalSemigroup(*I*, *r*) (function)

I is an ideal of a numerical semigroup and *r* is an integer. It returns the position of *r* in *I* (and `fail` if the integer is not in the ideal).

Example
```
gap> I := [2,5]+ NumericalSemigroup(7,8,17);;
gap> NumberElement_IdealOfNumericalSemigroup(I,19);
10
```

7.1.13 \[\] (for ideals of numerical semigroups)

▷ \[\](*I*, *r*) (operation)

I is an ideal of a numerical semigroup and *r* is an integer. It returns the *r*-th element of *I*.

Example
```
gap> I := [2,5]+ NumericalSemigroup(7,8,17);;
gap> I[10];
19
```

7.1.14 \{ \} (for ideals of numerical semigroups)

▷ `\{ \}(I, ls)` (operation)

I is an ideal of a numerical semigroup and *ls* is a list of integers. It returns the list *[I[r] : r in ls]*.

Example
```
gap> I := [2,5]+ NumericalSemigroup(7,8,17);;
gap> I{[10..13]};
[ 19, 20, 21, 22 ]
```

7.1.15 Iterator (for ideals of numerical semigroups)

▷ `Iterator(I)` (operation)

I is an ideal of a numerical semigroup. It returns an iterator over *I*.

Example
```
gap> s:=NumericalSemigroup(4,10,11);;
gap> i:=[2,3]+s;;
gap> iter:=Iterator(i);
<iterator>
gap> NextIterator(iter);
2
gap> NextIterator(iter);
3
gap> NextIterator(iter);
6
gap> SmallElements(i);
[ 2, 3, 6, 7, 10 ]
```

7.1.16 SumIdealsOfNumericalSemigroup

▷ `SumIdealsOfNumericalSemigroup(I, J)` (function)

▷ `+(I, J)` (function)

I, *J* are ideals of a numerical semigroup. The output is the sum of both ideals $\{i+j \mid i \in I, j \in J\}$.

Example
```
gap> I:=[3,5,9]+NumericalSemigroup(2,11);;
gap> J:=[2,11]+NumericalSemigroup(2,11);;
gap> I+J;
<Ideal of numerical semigroup>
gap> MinimalGeneratingSystemOfIdealOfNumericalSemigroup(last);
[ 5, 14 ]
gap> SumIdealsOfNumericalSemigroup(I,J);
```

```
<Ideal of numerical semigroup>
gap> MinimalGeneratingSystemOfIdealOfNumericalSemigroup(last);
[ 5, 14 ]
```

7.1.17 MultipleOfIdealOfNumericalSemigroup

▷ MultipleOfIdealOfNumericalSemigroup(*n*, *I*) (function)
▷ *(*n*, *I*) (function)

I is an ideal of a numerical semigroup, *n* is a non negative integer. The output is the ideal $I+\cdots+I$ (n times).

n * *I* can be used for short.

Example
```
gap> I:=[0,1]+NumericalSemigroup(3,5,7);;
gap> MinimalGeneratingSystemOfIdealOfNumericalSemigroup(2*I);
[ 0, 1, 2 ]
```

7.1.18 SubtractIdealsOfNumericalSemigroup

▷ SubtractIdealsOfNumericalSemigroup(*I*, *J*) (function)
▷ -(*I*, *J*) (function)

I, *J* are ideals of a numerical semigroup. The output is the ideal $\{z\in\mathbb{Z} \mid z+J\subseteq I\}$.

$I-J$ can be used as a short for SubtractIdealsOfNumericalSemigroup(I,J).

$S-J$ is a synonym of $(0+S)-J$, if S is the ambient semigroup of *I* and *J*. The following example appears in [HS04].

Example
```
gap> S:=NumericalSemigroup(14, 15, 20, 21, 25);;
gap> I:=[0,1]+S;;
gap> II:=S-I;;
gap> MinimalGeneratingSystemOfIdealOfNumericalSemigroup(I);
[ 0, 1 ]
gap> MinimalGeneratingSystemOfIdealOfNumericalSemigroup(II);
[ 14, 20 ]
gap> MinimalGeneratingSystemOfIdealOfNumericalSemigroup(I+II);
[ 14, 15, 20, 21 ]
```

7.1.19 Difference (for ideals of numerical semigroups)

▷ Difference(*I*, *J*) (operation)
▷ DifferenceOfIdealsOfNumericalSemigroup(*I*, *J*) (function)

I, *J* are ideals of a numerical semigroup. *J* must be contained in *I*. The output is the set $I\setminus J$.

Example
```
gap> S:=NumericalSemigroup(14, 15, 20, 21, 25);;
gap> I:=[0,1]+S;
<Ideal of numerical semigroup>
gap> 2*I-2*I;
<Ideal of numerical semigroup>
```

```
gap> I-I;
<Ideal of numerical semigroup>
gap> ii := 2*I-2*I;
<Ideal of numerical semigroup>
gap> i := I-I;
<Ideal of numerical semigroup>
gap>  DifferenceOfIdealsOfNumericalSemigroup(last2,last);
[ 26, 27, 37, 38 ]
gap> Difference(ii,i);
[ 26, 27, 37, 38 ]
gap> Difference(i,ii);
[  ]
```

7.1.20 TranslationOfIdealOfNumericalSemigroup

▷ TranslationOfIdealOfNumericalSemigroup(*k*, *I*) (function)
▷ +(*k*, *I*) (function)

Given an ideal `I` of a numerical semigroup S and an integer `k`, returns an ideal of the numerical semigroup S generated by $\{i_1+k,\dots,i_n+k\}$, where $\{i_1,\dots,i_n\}$ is the system of generators of `I`.

As a synonym to `TranslationOfIdealOfNumericalSemigroup(k, I)` the expression `k + I` may be used.

Example
```
gap> s:=NumericalSemigroup(13,23);;
gap> l:=List([1..6], _ -> Random([8..34]));
[ 22, 29, 34, 25, 10, 12 ]
gap> I:=IdealOfNumericalSemigroup(l, s);;
gap> It:=TranslationOfIdealOfNumericalSemigroup(7,I);
<Ideal of numerical semigroup>
gap> It2:=7+I;
<Ideal of numerical semigroup>
gap> It2=It;
true
```

7.1.21 Intersection (for ideals of numerical semigroups)

▷ Intersection(*I*, *J*) (operation)
▷ IntersectionIdealsOfNumericalSemigroup(*I*, *J*) (function)

Given two ideals `I` and `J` of a numerical semigroup `S` returns the ideal of the numerical semigroup `S` which is the intersection of the ideals `I` and `J`.

Example
```
gap> i:=IdealOfNumericalSemigroup([75,89],s);;
gap> j:=IdealOfNumericalSemigroup([115,289],s);;
gap> IntersectionIdealsOfNumericalSemigroup(i,j);
<Ideal of numerical semigroup>
```

7.1.22 MaximalIdealOfNumericalSemigroup

▷ MaximalIdealOfNumericalSemigroup(S) (function)

Returns the maximal ideal of the numerical semigroup S.

Example
```
gap> MaximalIdealOfNumericalSemigroup(NumericalSemigroup(3,7));
<Ideal of numerical semigroup>
```

7.1.23 CanonicalIdealOfNumericalSemigroup

▷ CanonicalIdealOfNumericalSemigroup(S) (function)

S is a numerical semigroup. Computes the (standard) canonical ideal of S ([BF97]): $\{x \in \mathbb{Z} | g - x \notin S\}$, where g is the Frobenius number of S.

Example
```
gap> s:=NumericalSemigroup(4,6,11);;
gap> m:=MaximalIdealOfNumericalSemigroup(s);;
gap> c:=CanonicalIdealOfNumericalSemigroup(s);
<Ideal of numerical semigroup>
gap> c-(c-m)=m;
true
gap> id:=3+s;
<Ideal of numerical semigroup>
gap> c-(c-id)=id;
true
```

7.1.24 IsCanonicalIdeal

▷ IsCanonicalIdeal(E) (property)
▷ IsCanonicalIdealOfNumericalSemigroup(E) (property)

E is an ideal of a numerical semigroup, say S. Determines if E is a translation of the canonical ideal of S, or equivalently, for every ideal J, $E-(E-J)=J$.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> c:=3+CanonicalIdealOfNumericalSemigroup(s);;
gap> c-(c-(3+s))=3+s;
true
gap> IsCanonicalIdealOfNumericalSemigroup(c);
true
```

7.1.25 TypeSequenceOfNumericalSemigroup

▷ TypeSequenceOfNumericalSemigroup(S) (function)

S is a numerical semigroup.

Computes the type sequence of a numerical semigroup. That is, the secuence $t_i(S) = \sharp(S(i) \setminus S(i-1))$, with $S(i) = \{s \in S \mid s \geq s_i\}$ and s_i the ith element of S.

This function is the implementation of the algorithm given in [BDF97].

Example

```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> TypeSequenceOfNumericalSemigroup(s);
[ 13, 3, 4, 4, 7, 3, 3, 3, 2, 2, 2, 3, 3, 2, 4, 3, 2, 1, 3, 2, 1, 1, 2, 2, 1,
  1, 1, 2, 2, 1, 3, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
  1, 1, 1 ]
gap> s:=NumericalSemigroup(4,6,11);;
gap> TypeSequenceOfNumericalSemigroup(s);
[ 1, 1, 1, 1, 1, 1, 1 ]
```

7.2 Blow ups and closures

The blow up of an ideal I of a numerical semigroup is the ideal $\bigcup_{n\geq 0} nI - nI$. In this section we provide functions to compute the blow up and related invariants.

7.2.1 HilbertFunctionOfIdealOfNumericalSemigroup

▷ `HilbertFunctionOfIdealOfNumericalSemigroup(n, I)` (function)

I is an ideal of a numerical semigroup, n is a non negative integer. I must be contained in its ambient semigroup. The output is the cardinality of the set $nI \setminus (n+1)I$.

Example

```
gap> I:=[6,9,11]+NumericalSemigroup(6,9,11);;
gap> List([1..7],n->HilbertFunctionOfIdealOfNumericalSemigroup(n,I));
[ 3, 5, 6, 6, 6, 6, 6 ]
```

7.2.2 BlowUpIdealOfNumericalSemigroup

▷ `BlowUpIdealOfNumericalSemigroup(I)` (function)

I is an ideal of a numerical semigroup. The output is the ideal $\bigcup_{n\geq 0} nI - nI$.

Example

```
gap> I:=[0,2]+NumericalSemigroup(6,9,11);;
gap> BlowUpIdealOfNumericalSemigroup(I);;
gap> SmallElementsOfIdealOfNumericalSemigroup(last);
[ 0, 2, 4, 6, 8 ]
```

7.2.3 ReductionNumber (for ideals of numerical semigroups)

▷ `ReductionNumber(I)` (attribute)

▷ `ReductionNumberIdealNumericalSemigroup(I)` (attribute)

I is an ideal of a numerical semigroup. The output is the least integer such that $nI + i = (n+1)I$, where $i = min(I)$.

Example

```
gap> I:=[0,2]+NumericalSemigroup(6,9,11);;
gap> ReductionNumberIdealNumericalSemigroup(I);
2
```

7.2.4 BlowUpOfNumericalSemigroup

▷ `BlowUpOfNumericalSemigroup(S)` (function)

S is a numerical semigroup. If M is the maximal ideal of the numerical semigroup, then the output is the numerical semigroup $\bigcup_{n\geq 0} nM - nM$.

Example

```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> BlowUpOfNumericalSemigroup(s);
<Numerical semigroup with 10 generators>
gap> SmallElementsOfNumericalSemigroup(last);
[ 0, 5, 10, 12, 15, 17, 20, 22, 24, 25, 27, 29, 30, 32, 34, 35, 36, 37, 39,
  40, 41, 42, 44 ]
gap> m:=MaximalIdealOfNumericalSemigroup(s);
<Ideal of numerical semigroup>
gap> BlowUpIdealOfNumericalSemigroup(m);
<Ideal of numerical semigroup>
gap> SmallElementsOfIdealOfNumericalSemigroup(last);
[ 0, 5, 10, 12, 15, 17, 20, 22, 24, 25, 27, 29, 30, 32, 34, 35, 36, 37, 39,
  40, 41, 42, 44 ]
```

7.2.5 LipmanSemigroup

▷ `LipmanSemigroup(S)` (function)

This is just a synonym of `BlowUpOfNumericalSemigroup` (7.2.4).

Example

```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> LipmanSemigroup(s);
<Numerical semigroup with 10 generators>
gap> SmallElementsOfNumericalSemigroup(last);
[ 0, 5, 10, 12, 15, 17, 20, 22, 24, 25, 27, 29, 30, 32, 34, 35, 36, 37, 39,
  40, 41, 42, 44 ]
```

7.2.6 RatliffRushNumberOfIdealOfNumericalSemigroup

▷ `RatliffRushNumberOfIdealOfNumericalSemigroup(I)` (function)

I is an ideal of a numerical semigroup. The output is the least integer such that $(n+1)I - nI$ is the Ratliff-Rush closure of I (see [DGH01]).

Example

```
gap> I:=[0,2]+NumericalSemigroup(6,9,11);;
gap> RatliffRushNumberOfIdealOfNumericalSemigroup(I);
1
```

7.2.7 RatliffRushClosureOfIdealOfNumericalSemigroup

▷ `RatliffRushClosureOfIdealOfNumericalSemigroup(I)` (function)

I is an ideal of a numerical semigroup. The output is the Ratliff-Rush closure of I: $\bigcup_{n\in\mathbb{N}}(n+1)I - nI$ (see [DGH01]).

Example
```
gap> I:=[0,2]+NumericalSemigroup(6,9,11);;
gap> RatliffRushClosureOfIdealOfNumericalSemigroup(I);
<Ideal of numerical semigroup>
gap> MinimalGenerators(last);
[ 0, 2, 4 ]
```

7.2.8 AsymptoticRatliffRushNumberOfIdealOfNumericalSemigroup

▷ AsymptoticRatliffRushNumberOfIdealOfNumericalSemigroup(*I*) (function)

I is an ideal of a numerical semigroup. The output is the least n such that the Ratliff-Rush closure of mI equals mI for all $m \geq n$ (see [DGH01]).

Example
```
gap> I:=[0,2]+NumericalSemigroup(6,9,11);;
gap> AsymptoticRatliffRushNumberOfIdealOfNumericalSemigroup(I);
2
```

7.2.9 MultiplicitySequenceOfNumericalSemigroup

▷ MultiplicitySequenceOfNumericalSemigroup(*S*) (function)

S is a numerical semigroup. The output is a list with the multiplicities of the sequence $S \subseteq L(S) \subseteq \cdots \subseteq \mathbb{N}$, where $L(\cdot)$ means LipmanSemigroup (7.2.5).

Example
```
gap> s:=NumericalSemigroup(3,5);
<Numerical semigroup with 2 generators>
gap> MultiplicitySequenceOfNumericalSemigroup(s);
[ 3, 2, 1 ]
```

7.2.10 MicroInvariantsOfNumericalSemigroup

▷ MicroInvariantsOfNumericalSemigroup(*S*) (function)

Returns the microinvariants of the numerical semigroup S defined in [Eli01]. For their computation we have used the formula given in [BF06]. The Apéry set of S and its blow up are involved in this computation.

Example
```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> bu:=BlowUpOfNumericalSemigroup(s);;
gap> ap:=AperyListOfNumericalSemigroupWRTElement(s,30);;
gap> apbu:=AperyListOfNumericalSemigroupWRTElement(bu,30);;
gap> (ap-apbu)/30;
[ 0, 4, 4, 3, 2, 1, 3, 4, 4, 3, 2, 3, 1, 4, 4, 3, 3, 1, 4, 4, 4, 3, 2, 4, 2,
  5, 4, 3, 3, 2 ]
gap> MicroInvariantsOfNumericalSemigroup(s)=last;
true
```

7.2.11 AperyListOfIdealOfNumericalSemigroupWRTElement

▷ `AperyListOfIdealOfNumericalSemigroupWRTElement(I, n)` (function)

`I` is an ideal and `n` is an integer. Computes the set of elements x of `I` such that $x-$`n` is not in the ideal `I`, where `n` is supposed to be in the ambient semigroup of `I`. The element in the ith position of the output list (starting in 0) is congruent with i modulo `n`.

Example
```
gap> s:=NumericalSemigroup(10,11,13);;
gap> i:=[12,14]+s;;
gap> AperyListOfIdealOfNumericalSemigroupWRTElement(i,10);
[ 40, 51, 12, 23, 14, 25, 36, 27, 38, 49 ]
```

7.2.12 AperyTableOfNumericalSemigroup

▷ `AperyTableOfNumericalSemigroup(s)` (function)

Computes the Apéry table associated to the numerical semigroup `s` as explained in [CBJZA13], that is, a list containing the Apéry list of `s` with respect to its multiplicity and the Apéry lists of kM (with M the maximal ideal of `s`) with respect to the multiplicity of `s`, for $k \in \{1,\dots,r\}$, where r is the reduction number of M (see `ReductionNumberIdealNumericalSemigroup` (7.2.3)).

Example
```
gap> s:=NumericalSemigroup(10,11,13);;
gap> AperyTableOfNumericalSemigroup(s);
[ [ 0, 11, 22, 13, 24, 35, 26, 37, 48, 39 ],
  [ 10, 11, 22, 13, 24, 35, 26, 37, 48, 39 ],
  [ 20, 21, 22, 23, 24, 35, 26, 37, 48, 39 ],
  [ 30, 31, 32, 33, 34, 35, 36, 37, 48, 39 ],
  [ 40, 41, 42, 43, 44, 45, 46, 47, 48, 49 ] ]
```

7.2.13 StarClosureOfIdealOfNumericalSemigroup

▷ `StarClosureOfIdealOfNumericalSemigroup(i, is)` (function)

`i` is an ideal and `is` is a set of ideals (all from the same numerical semigroup$s$). The output is $i^{*_{is}}$, where $*_{is}$ is the star operation generated by `is`: $(s-(s-i))\bigcap_{k\in is}(k-(k-i))$. The implementation uses Section 3 of [Spi15].

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> StarClosureOfIdealOfNumericalSemigroup([0,2]+s,[[0,4]+s]);;
gap> MinimalGeneratingSystemOfIdealOfNumericalSemigroup(last);
[ 0, 2, 4 ]
```

7.3 Patterns for ideals

In this section we document the functions implemented by K. Stokes related to patterns of ideals in numerical semigroups. The correctness of the algorithms can be found in [Sto15].

7.3.1 IsAdmissiblePattern

▷ `IsAdmissiblePattern(p)` (function)

p is the list of integers that are the coefficients of a pattern.
Returns `true` or `false` depending if the pattern is admissible or not (see [BAGS06]).

Example
```
gap> IsAdmissiblePattern([1,1,-1]);
true
gap> IsAdmissiblePattern([1,-2]);
false
```

7.3.2 IsStronglyAdmissiblePattern

▷ `IsStronglyAdmissiblePattern(p)` (function)

p is the list of integers that are the coefficients of a pattern.
Returns `true` or `false` depending if the pattern is strongly admissible or not (see [BAGS06]).

Example
```
gap> IsAdmissiblePattern([1,-1]);
true
gap> IsStronglyAdmissiblePattern([1,-1]);
false
gap> IsStronglyAdmissiblePattern([1,1,-1]);
true
```

7.3.3 AsIdealOfNumericalSemigroup

▷ `AsIdealOfNumericalSemigroup(I, T)` (function)

I is an ideal of a numerical semigroup S, and *T* is a numerical semigroup. Detects if *I* is an ideal of *T* and contained in *T* (integral ideal), and if so, returns *I* as an ideal of *T*. It returns `fail` if *I* is an ideal of some semigroup but not an integral ideal of *T*.

Example
```
gap> s:=NumericalSemigroup(3,7,5);;
gap> t:=NumericalSemigroup(10,11,14);;
gap> AsIdealOfNumericalSemigroup(10+s,t);
fail
gap> AsIdealOfNumericalSemigroup(100+s,t);
<Ideal of numerical semigroup>
```

7.3.4 BoundForConductorOfImageOfPattern

▷ `BoundForConductorOfImageOfPattern(p, C)` (function)

p is the list of integers that are the coefficients of an admissible pattern. *C* is a positive integer. Calculates an upper bound of the smallest element K in $p(I)$ such that all integers larger than K belong to $p(I)$, where I is an ideal of a numerical semigroup. Instead of taking I as parameter, the function takes C, which is assumed to be the conductor of I.

Example
```
gap> BoundForConductorOfImageOfPattern([1,1,-1],10);
10
```

7.3.5 ApplyPatternToIdeal

▷ ApplyPatternToIdeal(*p*, *I*) (function)

p is the list of integers that are the coefficients of a strongly admissible pattern. *I* is an ideal of a numerical semigroup.

Outputs $p(I)$, represented as `[d,p(I)/d]`, where `d` is the gcd of the coefficients of *p*. All elements of $p(I)$ are divisible by d, and $p(I)/d$ is an ideal of some numerical semigroup. It is returned as the maximal ideal of the numerical semigroup $p(I)/d \cup \{0\}$. The ambient numerical semigroup can later be changed with the function `AsIdealOfNumericalSemigroup`.

Example
```
gap> s:=NumericalSemigroup(3,7,5);;
gap> i:=10+s;;
gap> ApplyPatternToIdeal([1,1,-1],i);
[ 1, <Ideal of numerical semigroup> ]
```

7.3.6 ApplyPatternToNumericalSemigroup

▷ ApplyPatternToNumericalSemigroup(*p*, *S*) (function)

p is the list of integers that are the coefficients of a strongly admissible pattern. *S* is a numerical semigroup.

Outputs `ApplyPatternToIdeal(p,0+S)`.

Example
```
gap> s:=NumericalSemigroup(3,7,5);;
gap> ApplyPatternToNumericalSemigroup([1,1,-1],s);
[ 1, <Ideal of numerical semigroup> ]
gap> SmallElements(last[2]);
[ 0, 3, 5 ]
```

7.3.7 IsAdmittedPatternByIdeal

▷ IsAdmittedPatternByIdeal(*p*, *I*, *J*) (function)

p is the list of integers that are the coefficients of a strongly admissible pattern. *I* and *J* are ideals of certain numerical semigroups.

Tests whether or not $p(I)$ is contained in *J*.

Example
```
gap> s:=NumericalSemigroup(3,7,5);;
gap> i:=[3,5]+s;;
gap> IsAdmittedPatternByIdeal([1,1,-1],i,i);
false
gap> IsAdmittedPatternByIdeal([1,1,-1],i,0+s);
true
```

7.3.8 IsAdmittedPatternByNumericalSemigroup

▷ `IsAdmittedPatternByNumericalSemigroup(p, S, T)` (function)

p is the list of integers that are the coefficients of a strongly admissible pattern. *S* and *T* are numerical semigroups.

Tests whether or not $p(S)$ is contained in *T*.

Example
```
gap> s:=NumericalSemigroup(3,7,5);;
gap> IsAdmittedPatternByNumericalSemigroup([1,1,-1],s,s);
true
gap> IsArfNumericalSemigroup(s);
true
```

7.4 Graded associated ring of numerical semigroup

This section contains several functions to test properties of the graded (with respect to the maximal ideal) semigroup ring $\mathbb{K}[\![S]\!]$ (with S a numerical semigroup).

7.4.1 IsGradedAssociatedRingNumericalSemigroupCM

▷ `IsGradedAssociatedRingNumericalSemigroupCM(S)` (property)

S is a numerical semigroup. Returns `true` if the graded ring associated to $K[[S]]$ is Cohen-Macaulay, and `false` otherwise. This test is the implementation of the algorithm given in [BF06].

This filter implies `IsGradedAssociatedRingNumericalSemigroupBuchsbaum` (7.4.2).

Example
```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> IsGradedAssociatedRingNumericalSemigroupCM(s);
false
gap> MicroInvariantsOfNumericalSemigroup(s);
[ 0, 4, 4, 3, 2, 1, 3, 4, 4, 3, 2, 3, 1, 4, 4, 3, 3, 1, 4, 4, 4, 3, 2, 4, 2,
  5, 4, 3, 3, 2 ]
gap> List(AperyListOfNumericalSemigroupWRTElement(s,30),
> w->MaximumDegreeOfElementWRTNumericalSemigroup (w,s));
[ 0, 1, 4, 1, 2, 1, 3, 1, 4, 3, 2, 3, 1, 1, 4, 3, 3, 1, 4, 1, 4, 3, 2, 4, 2,
  5, 4, 3, 1, 2 ]
gap> last=last2;
false
gap> s:=NumericalSemigroup(4,6,11);;
gap> IsGradedAssociatedRingNumericalSemigroupCM(s);
true
gap> MicroInvariantsOfNumericalSemigroup(s);
[ 0, 2, 1, 1 ]
gap> List(AperyListOfNumericalSemigroupWRTElement(s,4),
> w->MaximumDegreeOfElementWRTNumericalSemigroup(w,s));
[ 0, 2, 1, 1 ]
```

7.4.2 IsGradedAssociatedRingNumericalSemigroupBuchsbaum

▷ `IsGradedAssociatedRingNumericalSemigroupBuchsbaum(S)` (property)

S is a numerical semigroup.

Returns `true` if the graded ring associated to $K[[S]]$ is Buchsbaum, and `false` otherwise. This test is the implementation of the algorithm given in [DMV09].

Example
```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> IsGradedAssociatedRingNumericalSemigroupBuchsbaum(s);
true
```

7.4.3 TorsionOfAssociatedGradedRingNumericalSemigroup

▷ `TorsionOfAssociatedGradedRingNumericalSemigroup(S)` (function)

S is a numerical semigroup.

This function returns the set of elements in the numerical semigroup S corresponding to a $\mathbb{K}$-basis of the torsion submodule of the associated graded ring of the numerical semigroup ring $\mathbb{K}[[S]]$. It uses the Apery table as explained in [CBJZA13].

Example
```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> TorsionOfAssociatedGradedRingNumericalSemigroup(s);
[ 181, 153, 157, 193, 169, 148 ]
```

7.4.4 BuchsbaumNumberOfAssociatedGradedRingNumericalSemigroup

▷ `BuchsbaumNumberOfAssociatedGradedRingNumericalSemigroup(S)` (function)

S is a numerical semigroup.

This function returns the smallest non-negative integer k for which the associated graded ring G of a given numerical semigroup ring is k-Buchsbaum, that is, the least k for which the torsion submodule of G is annihilated by the k-th power of the homogeneous maximal ideal of G.

Example
```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> BuchsbaumNumberOfAssociatedGradedRingNumericalSemigroup(s);
1
gap> IsGradedAssociatedRingNumericalSemigroupBuchsbaum(s);
true
```

7.4.5 IsMpure

▷ `IsMpure(S)` (property)

▷ `IsMpureNumericalSemigroup(S)` (property)

S is a numerical semigroup.

Test for the M-Purity of the numerical semigroup S S. This test is based on [Bry10].

This filter implies `IsPureNumericalSemigroup` (7.4.6).

Example

```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> IsMpureNumericalSemigroup(s);
false
gap> s:=NumericalSemigroup(4,6,11);;
gap> IsMpureNumericalSemigroup(s);
true
```

7.4.6 IsPure

▷ `IsPure(S)` (property)
▷ `IsPureNumericalSemigroup(S)` (property)

S is a numerical semigroup.

Test for the purity of the numerical semigroup S S. This test is based on [Bry10].

Example

```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> IsPureNumericalSemigroup(s);
false
gap> s:=NumericalSemigroup(4,6,11);;
gap> IsPureNumericalSemigroup(s);
true
```

7.4.7 IsGradedAssociatedRingNumericalSemigroupGorenstein

▷ `IsGradedAssociatedRingNumericalSemigroupGorenstein(S)` (function)

S is a numerical semigroup.

Returns `true` if the graded ring associated to $K[[S]]$ is Gorenstein, and `false` otherwise. This test is the implementation of the algorithm given in [DMS11].

This filter implies `IsGradedAssociatedRingNumericalSemigroupCM` (7.4.1), `IsMpureNumericalSemigroup` (7.4.5), and `IsSymmetricNumericalSemigroup` (6.1.2).

Example

```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> IsGradedAssociatedRingNumericalSemigroupGorenstein(s);
false
gap> s:=NumericalSemigroup(4,6,11);;
gap> IsGradedAssociatedRingNumericalSemigroupGorenstein(s);
true
```

7.4.8 IsGradedAssociatedRingNumericalSemigroupCI

▷ `IsGradedAssociatedRingNumericalSemigroupCI(S)` (function)

S is a numerical semigroup.

Returns `true` if the Complete Intersection property of the associated graded ring of a numerical semigroup ring associated to $K[[S]]$, and `false` otherwise. This test is the implementation of the algorithm given in [DMS13].

This filter implies `IsGradedAssociatedRingNumericalSemigroupGorenstein` (7.4.7) and `IsAperySetGammaRectangular` (6.2.10).

Example

```
gap> s:=NumericalSemigroup(30, 35, 42, 47, 148, 153, 157, 169, 181, 193);;
gap> IsGradedAssociatedRingNumericalSemigroupCI(s);
false
gap> s:=NumericalSemigroup(4,6,11);;
gap> IsGradedAssociatedRingNumericalSemigroupCI(s);
true
```

Chapter 8

Numerical semigroups with maximal embedding dimension

If S is a numerical semigroup and m is its multiplicity (the least positive integer belonging to it), then the embedding dimension e of S (the cardinality of the minimal system of generators of S) is less than or equal to m. We say that S has *maximal embedding dimension* (MED for short) when $e = m$. The intersection of two numerical semigroups with the same multiplicity and maximal embedding dimension is again of maximal embedding dimension. Thus we define the MED closure of a non-empty subset of positive integers $M = \{m < m_1 < \cdots < m_n < \cdots\}$ with $\gcd(M) = 1$ as the intersection of all MED numerical semigroups with multiplicity m.

Given a MED numerical semigroup S, we say that $M = \{m_1 < \cdots < m_k\}$ is a MED system of generators if the MED closure of M is S. Moreover, M is a minimal MED generating system for S provided that every proper subset of M is not a MED system of generators of S. Minimal MED generating systems are unique, and in general are smaller than the classical minimal generating systems (see [RGSGGB03]).

8.1 Numerical semigroups with maximal embedding dimension

This section describes the basic functions to deal with maximal embedding dimension numerical semigroups, and MED generating systems.

8.1.1 IsMED

▷ IsMED(S) (property)

▷ IsMEDNumericalSemigroup(S) (property)

S is a numerical semigroup. Returns true if S is a MED numerical semigroup and false otherwise.

Example
```
gap> IsMEDNumericalSemigroup(NumericalSemigroup(3,5,7));
true
gap> IsMEDNumericalSemigroup(NumericalSemigroup(3,5));
false
```

8.1.2 MEDNumericalSemigroupClosure

▷ MEDNumericalSemigroupClosure(S) (function)
▷ MEDClosure(S) (operation)

S is a numerical semigroup. Returns the MED closure of S.

Example
```
gap> MEDNumericalSemigroupClosure(NumericalSemigroup(3,5));
<Numerical semigroup>
gap> MinimalGenerators(last);
[ 3, 5, 7 ]
```

8.1.3 MinimalMEDGeneratingSystemOfMEDNumericalSemigroup

▷ MinimalMEDGeneratingSystemOfMEDNumericalSemigroup(S) (function)

S is a MED numerical semigroup. Returns the minimal MED generating system of S.

Example
```
gap> MinimalMEDGeneratingSystemOfMEDNumericalSemigroup(
> NumericalSemigroup(3,5,7));
[ 3, 5 ]
```

8.2 Numerical semigroups with the Arf property and Arf closures

A numerical semigroup S is *Arf* if for every x,y,z in S with $x \geq y \geq z$, one has that $x+y-z \in S$. Numerical semigroups with the Arf property are a special kind of numerical semigroups with maximal embedding dimension.

The intersection of two Arf numerical semigroups is again Arf, and thus we can consider the Arf closure of a set of nonnegative integers with greatest common divisor equal to one. Analogously as with MED numerical semigroups, we define Arf systems of generators and minimal Arf generating system for an Arf numerical semigroup. These are also unique (see [RGSGGB04]).

8.2.1 IsArf

▷ IsArf(S) (property)
▷ IsArfNumericalSemigroup(S) (property)

S is a numerical semigroup. Returns true if S is an Arf numerical semigroup and false otherwise. This property implies IsMED (8.1.1) and IsAcuteNumericalSemigroup (3.1.28).

Example
```
gap>  IsArfNumericalSemigroup(NumericalSemigroup(3,5,7));
true
gap>  IsArfNumericalSemigroup(NumericalSemigroup(3,7,11));
false
gap> IsMEDNumericalSemigroup(NumericalSemigroup(3,7,11));
true
```

8.2.2 ArfNumericalSemigroupClosure

▷ `ArfNumericalSemigroupClosure(S)` (function)
▷ `ArfClosure(S)` (operation)

S is a numerical semigroup. Returns the Arf closure of *S*.

Example
```
gap> ArfNumericalSemigroupClosure(NumericalSemigroup(3,7,11));
<Numerical semigroup>
gap> MinimalGenerators(last);
[ 3, 7, 8 ]
```

8.2.3 ArfCharactersOfArfNumericalSemigroup

▷ `ArfCharactersOfArfNumericalSemigroup(S)` (function)
▷ `MinimalArfGeneratingSystemOfArfNumericalSemigroup(S)` (function)

S is an Arf numerical semigroup. Returns the minimal Arf generating system of *S*. The current version of this algorithm is due to G. Zito.

Example
```
gap> MinimalArfGeneratingSystemOfArfNumericalSemigroup(
> NumericalSemigroup(3,7,8));
[ 3, 7 ]
```

8.2.4 ArfNumericalSemigroupsWithFrobeniusNumber

▷ `ArfNumericalSemigroupsWithFrobeniusNumber(f)` (function)

f is an integer. The output is the set of all Arf numerical semigroups with Frobenius number *f*. The current version of this algorithm is due to G. Zito.

Example
```
gap> ArfNumericalSemigroupsWithFrobeniusNumber(10);
[ <Numerical semigroup>, <Numerical semigroup>, <Numerical semigroup>,
  <Numerical semigroup>, <Numerical semigroup>, <Numerical semigroup>,
  <Numerical semigroup>, <Numerical semigroup>, <Numerical semigroup> ]
gap> Set(last,MinimalGenerators);
[ [ 3, 11, 13 ], [ 4, 11, 13, 14 ], [ 6, 9, 11, 13, 14, 16 ],
  [ 6, 11, 13, 14, 15, 16 ], [ 7, 9, 11, 12, 13, 15, 17 ],
  [ 7, 11, 12, 13, 15, 16, 17 ], [ 8, 11, 12, 13, 14, 15, 17, 18 ],
  [ 9, 11, 12, 13, 14, 15, 16, 17, 19 ], [ 11 .. 21 ] ]
```

8.2.5 ArfNumericalSemigroupsWithFrobeniusNumberUpTo

▷ `ArfNumericalSemigroupsWithFrobeniusNumberUpTo(f)` (function)

f is an integer. The output is the set of all Arf numerical semigroups with Frobenius number less than or equal to *f*. The current version of this algorithm is due to G. Zito.

Example
```
gap> Length(ArfNumericalSemigroupsWithFrobeniusNumberUpTo(10));
46
```

8.2.6 ArfNumericalSemigroupsWithGenus

▷ ArfNumericalSemigroupsWithGenus(g) (function)

g is a nonnegative integer. The output is the set of all Arf numerical semigroups with genus equal to g. The current version of this algorithm is due to G. Zito.

Example
```
gap> Length(ArfNumericalSemigroupsWithGenus(10));
21
```

8.2.7 ArfNumericalSemigroupsWithGenusUpTo

▷ ArfNumericalSemigroupsWithGenusUpTo(g) (function)

g is a nonnegative integer. The output is the set of all Arf numerical semigroups with genus less than or equal to g. The current version of this algorithm is due to G. Zito.

Example
```
gap> Length(ArfNumericalSemigroupsWithGenusUpTo(10));
86
```

8.2.8 ArfNumericalSemigroupsWithGenusAndFrobeniusNumber

▷ ArfNumericalSemigroupsWithGenusAndFrobeniusNumber(g, f) (function)

f and g are integers. The output is the set of all Arf numerical semigroups with genus g and Frobenius number f. The algorithm is explained in [GSHKR17].

Example
```
gap> ArfNumericalSemigroupsWithGenusAndFrobeniusNumber(10,13);
[ <Numerical semigroup>, <Numerical semigroup>, <Numerical semigroup>,
  <Numerical semigroup>, <Numerical semigroup> ]
gap> List(last,MinimalGenerators);
[ [ 8, 10, 12, 14, 15, 17, 19, 21 ], [ 6, 10, 14, 15, 17, 19 ],
  [ 5, 12, 14, 16, 18 ], [ 6, 9, 14, 16, 17, 19 ], [ 4, 14, 15, 17 ] ]
```

8.3 Saturated numerical semigroups

A numerical semigroup S is *saturated* if the following condition holds: $s, s_1, \dots, s_r$ in S are such that $s_i \leq s$ for all i in $\{1, \dots, r\}$ and $z_1, \dots, z_r$ in $\mathbb{Z}$ are such that $z_1 s_1 + \cdots + z_r s_r \geq 0$, then $s + z_1 s_1 + \cdots + z_r s_r$ in S. Saturated numerical semigroups are a special kind of numerical semigroups with maximal embedding dimension.

The intersection of two saturated numerical semigroups is again saturated, and thus we can consider the saturated closure of a set of nonnegative integers with greatest common divisor equal to one (see [RGS09]).

8.3.1 IsSaturated

▷ IsSaturated(S) (property)
▷ IsSaturatedNumericalSemigroup(S) (property)

S is a numerical semigroup. Returns `true` if *S* is a saturated numerical semigroup and `false` otherwise.

This property implies `IsArf` (8.2.1).

Example
```
gap> IsSaturatedNumericalSemigroup(NumericalSemigroup(4,6,9,11));
true
gap> IsSaturatedNumericalSemigroup(NumericalSemigroup(8, 9, 12, 13, 15, 19 ));
false
```

8.3.2 SaturatedNumericalSemigroupClosure

▷ `SaturatedNumericalSemigroupClosure(S)` (function)

▷ `SaturatedClosure(S)` (operation)

S is a numerical semigroup. Returns the saturated closure of *S*.

Example
```
gap> SaturatedNumericalSemigroupClosure(NumericalSemigroup(8, 9, 12, 13, 15));
<Numerical semigroup>
gap> MinimalGenerators(last);
[ 8 .. 15 ]
```

8.3.3 SaturatedNumericalSemigroupsWithFrobeniusNumber

▷ `SaturatedNumericalSemigroupsWithFrobeniusNumber(f)` (function)

f is an integer. The output is the set of all saturated numerical semigroups with Frobenius number *f*.

Example
```
gap> SaturatedNumericalSemigroupsWithFrobeniusNumber(10);
[ <Numerical semigroup with 3 generators>,
  <Numerical semigroup with 4 generators>,
  <Numerical semigroup with 6 generators>,
  <Numerical semigroup with 6 generators>,
  <Numerical semigroup with 7 generators>,
  <Numerical semigroup with 8 generators>,
  <Numerical semigroup with 9 generators>,
  <Numerical semigroup with 11 generators> ]
gap>  List(last,MinimalGenerators);
[ [ 3, 11, 13 ], [ 4, 11, 13, 14 ], [ 6, 9, 11, 13, 14, 16 ],
  [ 6, 11, 13, 14, 15, 16 ], [ 7, 11, 12, 13, 15, 16, 17 ],
  [ 8, 11, 12, 13, 14, 15, 17, 18 ], [ 9, 11, 12, 13, 14, 15, 16, 17, 19 ],
  [ 11 .. 21 ] ]
```

Chapter 9

Nonunique invariants for factorizations in numerical semigroups

Let S be a numerical semigroup minimally generated by $\{m_1,\ldots,m_n\}$. A *factorization* of an element $s \in S$ is an n-tuple $a = (a_1,\ldots,a_n)$ of nonnegative integers such that $n = a_1n_1 + \cdots + a_nm_n$. The *length* of a is $|a| = a_1 + \cdots + a_n$. Given two factorizations a and b of n, the *distance* between a and b is $d(a,b) = \max\{|a - \gcd(a,b)|, |b - \gcd(a,b)|\}$, where $\gcd((a_1,\ldots,a_n),(b_1,\ldots,b_n)) = (\min(a_1,b_1),\ldots,\min(a_n,b_n))$.

If $l_1 > \cdots > l_k$ are the lengths of all the factorizations of $s \in S$, the *delta set* associated to s is $\Delta(s) = \{l_1 - l_2,\ldots,l_k - l_{k-1}\}$.

The *catenary degree* of an element in S is the least positive integer c such that for any two of its factorizations a and b, there exists a chain of factorizations starting in a and ending in b and so that the distance between two consecutive links is at most c. The *catenary degree* of S is the supremum of the catenary degrees of the elements in S.

The *tame degree* of S is the least positive integer t such that for any factorization a of an element s in S, and any i such that $s - m_i \in S$, there exists another factorization b of s so that the distance to a is at most t and $b_i \neq 0$.

The ω*-primality* of an element s in S is the least positive integer k such that if $(\sum_{i\in I} s_i) - s \in S, s_i \in S$, then there exists $\Omega \subseteq I$ with cardinality k such that $(\sum_{i\in\Omega} s_i) - s \in S$. The ω*-primality* of S is the maximum of the ω-primality of its minimal generators.

The basic properties of these constants can be found in [GHK06]. The algorithm used to compute the catenary and tame degree is an adaptation of the algorithms appearing in [CGSL+06] for numerical semigroups (see [CGSD07]). The computation of the elasticity of a numerical semigroup reduces to m/n with m the multiplicity of the semigroup and n its largest minimal generator (see [CHM06] or [GHK06]).

9.1 Factorizations in Numerical Semigroups

Denumerants, sets of factorizations, R-classes, and L-shapes are described in this section.

9.1.1 FactorizationsIntegerWRTList

▷ `FactorizationsIntegerWRTList(n, ls)` (function) `ls` is a list of integers and `n` an integer. The output is the set of factorizations of `n` in terms of the elements in the list `ls`. This function uses `RestrictedPartitions` (**Reference: RestrictedPartitions**).

Example
```
gap> FactorizationsIntegerWRTList(100,[11,13,15,19]);
[ [ 2, 6, 0, 0 ], [ 3, 4, 1, 0 ], [ 4, 2, 2, 0 ], [ 5, 0, 3, 0 ],
  [ 5, 2, 0, 1 ], [ 6, 0, 1, 1 ], [ 0, 1, 2, 3 ], [ 1, 1, 0, 4 ] ]
```

9.1.2 FactorizationsElementWRTNumericalSemigroup

▷ `FactorizationsElementWRTNumericalSemigroup(n, S)` (function)

`S` is a numerical semigroup and `n` an element of `S`. The output is the set of factorizations of `n` in terms of the minimal generating set of `S`.

Example
```
gap> s:=NumericalSemigroup(101,113,196,272,278,286);
<Numerical semigroup with 6 generators>
gap> FactorizationsElementWRTNumericalSemigroup(1100,s);
[ [ 0, 8, 1, 0, 0, 0 ], [ 0, 0, 0, 2, 2, 0 ], [ 5, 1, 1, 0, 0, 1 ],
  [ 0, 2, 3, 0, 0, 1 ] ]
```

9.1.3 FactorizationsElementListWRTNumericalSemigroup

▷ `FactorizationsElementListWRTNumericalSemigroup(l, S)` (function)

`S` is a numerical semigroup and `l` a list of elements of `S`.

Computes the factorizations of all the elements in `l`.

Example
```
gap> s:=NumericalSemigroup(10,11,13);
<Numerical semigroup with 3 generators>
gap> FactorizationsElementListWRTNumericalSemigroup([100,101,103],s);
[ [ [ 0, 2, 6 ], [ 1, 7, 1 ], [ 3, 4, 2 ], [ 5, 1, 3 ], [ 10, 0, 0 ] ],
  [ [ 0, 8, 1 ], [ 1, 0, 7 ], [ 2, 5, 2 ], [ 4, 2, 3 ], [ 9, 1, 0 ] ],
  [ [ 0, 7, 2 ], [ 2, 4, 3 ], [ 4, 1, 4 ], [ 7, 3, 0 ], [ 9, 0, 1 ] ] ]
```

9.1.4 RClassesOfSetOfFactorizations

▷ `RClassesOfSetOfFactorizations(ls)` (function)

`ls` is a set of factorizations (a list of lists of nonnegative integers with the same length). The output is the set of $\mathscr{R}$-classes of this set of factorizations as defined in Chapter 7 of [RGS09].

Example
```
gap> s:=NumericalSemigroup(10,11,19,23);;
gap> BettiElementsOfNumericalSemigroup(s);
[ 30, 33, 42, 57, 69 ]
gap> FactorizationsElementWRTNumericalSemigroup(69,s);
[ [ 5, 0, 1, 0 ], [ 2, 1, 2, 0 ], [ 0, 0, 0, 3 ] ]
gap> RClassesOfSetOfFactorizations(last);
[ [ [ 2, 1, 2, 0 ], [ 5, 0, 1, 0 ] ], [ [ 0, 0, 0, 3 ] ] ]
```

9.1.5 LShapesOfNumericalSemigroup

▷ `LShapesOfNumericalSemigroup(S)` (function)

`S` is a numerical semigroup. The output is the number of LShapes associated to `S`. These are ways of arranging the set of factorizations of the elements in the Apéry set of the largest generator, so that if one factorization x is chosen for w and $w-w' \in S$, then only the factorization of x' of w' with $x' \leq x$ can be in the LShape (and if there is no such a factorization, then we have no LShape with x in it), see [AGGS10].

Example
```
gap> s:=NumericalSemigroup(4,6,9);;
gap> LShapesOfNumericalSemigroup(s);
[ [ [ 0, 0 ], [ 1, 0 ], [ 0, 1 ], [ 2, 0 ], [ 1, 1 ], [ 0, 2 ], [ 2, 1 ],
      [ 1, 2 ], [ 2, 2 ] ],
  [ [ 0, 0 ], [ 1, 0 ], [ 0, 1 ], [ 2, 0 ], [ 1, 1 ], [ 3, 0 ], [ 2, 1 ],
      [ 4, 0 ], [ 5, 0 ] ] ]
```

9.1.6 DenumerantOfElementInNumericalSemigroup

▷ `DenumerantOfElementInNumericalSemigroup(n, S)` (function)

`S` is a numerical semigroup and `n` a positive integer. The output is the number of factorizations of `n` in terms of the minimal generating set of `S`.

Example
```
gap> s:=NumericalSemigroup(101,113,195,272,278,286);;
gap> DenumerantOfElementInNumericalSemigroup(1311,s);
6
```

9.2 Invariants based on lengths

This section is devoted to nonunique factorization invariants based on lengths of factorizations. There are some families of numerical semigroups related to maximal denumerantes; membership tests for these families are provede here.

9.2.1 LengthsOfFactorizationsIntegerWRTList

▷ `LengthsOfFactorizationsIntegerWRTList(n, ls)` (function)

`ls` is a list of integers and `n` an integer. The output is the set of lengths of the factorizations of `n` in terms of the elements in `ls`.

Example
```
gap> LengthsOfFactorizationsIntegerWRTList(100,[11,13,15,19]);
[ 6, 8 ]
```

9.2.2 LengthsOfFactorizationsElementWRTNumericalSemigroup

▷ `LengthsOfFactorizationsElementWRTNumericalSemigroup(n, S)` (function)

S is a numerical semigroup and n an element of S. The output is the set of lengths of the factorizations of n in terms of the minimal generating set of S.

Example
```
gap> s:=NumericalSemigroup(101,113,196,272,278,286);
<Numerical semigroup with 6 generators>
gap> LengthsOfFactorizationsElementWRTNumericalSemigroup(1100,s);
[ 4, 6, 8, 9 ]
```

9.2.3 ElasticityOfFactorizationsElementWRTNumericalSemigroup

▷ `ElasticityOfFactorizationsElementWRTNumericalSemigroup(n, S)` (function)

S is a numerical semigroup and n an element of S. The output is the maximum length divided by the minimum length of the factorizations of n in terms of the minimal generating set of S.

Example
```
gap> s:=NumericalSemigroup(101,113,196,272,278,286);
<Numerical semigroup with 6 generators>
gap> ElasticityOfFactorizationsElementWRTNumericalSemigroup(1100,s);
9/4
```

9.2.4 ElasticityOfNumericalSemigroup

▷ `ElasticityOfNumericalSemigroup(S)` (function)

S is a numerical semigroup. The output is the elasticity of S.

Example
```
gap> s:=NumericalSemigroup(101,113,196,272,278,286);
<Numerical semigroup with 6 generators>
gap> ElasticityOfNumericalSemigroup(s);
286/101
```

9.2.5 DeltaSetOfSetOfIntegers

▷ `DeltaSetOfSetOfIntegers(ls)` (function)

`ls` is list of integers. The output is the Delta set of the elements in `ls`, that is, the set of differences of consecutive elements in the list.

Example
```
gap> LengthsOfFactorizationsIntegerWRTList(100,[11,13,15,19]);
[ 6, 8 ]
gap> DeltaSetOfSetOfIntegers(last);
[ 2 ]
```

9.2.6 DeltaSetOfFactorizationsElementWRTNumericalSemigroup

▷ `DeltaSetOfFactorizationsElementWRTNumericalSemigroup(n, S)` (function)

S is a numerical semigroup and n an element of S. The output is the Delta set of the factorizations of n in terms of the minimal generating set of S.

Example
```
gap> s:=NumericalSemigroup(101,113,196,272,278,286);
<Numerical semigroup with 6 generators>
gap> DeltaSetOfFactorizationsElementWRTNumericalSemigroup(1100,s);
[ 1, 2 ]
```

9.2.7 DeltaSetPeriodicityBoundForNumericalSemigroup

▷ DeltaSetPeriodicityBoundForNumericalSemigroup(*S*) (function)

S is a numerical semigroup. Computes the bound were the periodicity starts for Delta sets of the elements in *S*; see [GGMFVT15].

Example
```
gap> s:=NumericalSemigroup(5,7,11);;
gap> DeltaSetPeriodicityBoundForNumericalSemigroup(s);
60
```

9.2.8 DeltaSetPeriodicityStartForNumericalSemigroup

▷ DeltaSetPeriodicityStartForNumericalSemigroup(*S*) (function)

S is a numerical semigroup.

Computes the element were the periodicity starts for Delta sets of the elements in *S*.

Example
```
gap> s:=NumericalSemigroup(5,7,11);;
gap> DeltaSetPeriodicityStartForNumericalSemigroup(s);
21
```

9.2.9 DeltaSetListUpToElementWRTNumericalSemigroup

▷ DeltaSetListUpToElementWRTNumericalSemigroup(*n*, *S*) (function)

S is a numerical semigroup, *n* an integer.

Computes the Delta sets of the integers up to (and including) *n*, if an integer is not in *S*, the corresponding Delta set is empty.

Example
```
gap> s:=NumericalSemigroup(5,7,11);;
gap> DeltaSetListUpToElementWRTNumericalSemigroup(31,s);
[ [  ], [  ], [  ], [  ], [  ], [  ], [  ], [  ], [  ], [  ], [  ], [  ], [  ],
  [  ], [  ], [  ], [  ], [  ], [  ], [  ], [  ], [ 2 ], [  ], [  ], [ 2 ], [  ],
  [ 2 ], [  ], [ 2 ], [ 2 ], [  ] ]
```

9.2.10 DeltaSetUnionUpToElementWRTNumericalSemigroup

▷ DeltaSetUnionUpToElementWRTNumericalSemigroup(*n*, *S*) (function)

S is a numerical semigroup, n a nonnegative integer.

Computes the union of the delta sets of the elements of S up to and including n, using a ring buffer to conserve memory.

Example
```
gap> s:=NumericalSemigroup(5,7,11);;
gap> DeltaSetUnionUpToElementWRTNumericalSemigroup(60,s);
[ 2 ]
```

9.2.11 DeltaSetOfNumericalSemigroup

▷ DeltaSetOfNumericalSemigroup(*S*) (function)

S is a numerical semigroup.

Computes the Delta set of S.

Example
```
gap> s:=NumericalSemigroup(5,7,11);;
gap> DeltaSetOfNumericalSemigroup(s);
[ 2 ]
```

9.2.12 MaximumDegreeOfElementWRTNumericalSemigroup

▷ MaximumDegreeOfElementWRTNumericalSemigroup(*n*, *S*) (function)

S is a numerical semigroup and n a nonnegative integer. The output is the maximum length of the factorizations of n in terms of the minimal generating set of S.

Example
```
gap> s:=NumericalSemigroup(101,113,196,272,278,286);
<Numerical semigroup with 6 generators>
gap> MaximumDegreeOfElementWRTNumericalSemigroup(1100,s);
9
```

9.2.13 MaximalDenumerantOfElementInNumericalSemigroup

▷ MaximalDenumerantOfElementInNumericalSemigroup(*n*, *S*) (function)

S is a numerical semigroup and n an element of S. The output is the number of factorizations of n in terms of the minimal generating set of S with maximal length.

Example
```
gap> s:=NumericalSemigroup(101,113,196,272,278,286);;
gap> MaximalDenumerantOfElementInNumericalSemigroup(1100,s);
1
gap> MaximalDenumerantOfElementInNumericalSemigroup(1311,s);
2
```

9.2.14 MaximalDenumerantOfSetOfFactorizations

▷ MaximalDenumerantOfSetOfFactorizations(*ls*) (function)

ls is list of factorizations (a list of lists of nonnegative integers with the same length). The output is number of elements in *ls* with maximal length.

Example
```
gap> FactorizationsIntegerWRTList(100,[11,13,15,19]);
[ [ 2, 6, 0, 0 ], [ 3, 4, 1, 0 ], [ 4, 2, 2, 0 ], [ 5, 0, 3, 0 ], [ 5, 2, 0, 1 ],
  [ 6, 0, 1, 1 ], [ 0, 1, 2, 3 ], [ 1, 1, 0, 4 ] ]
gap> MaximalDenumerantOfSetOfFactorizations(last);
6
```

9.2.15 MaximalDenumerantOfNumericalSemigroup

▷ MaximalDenumerantOfNumericalSemigroup(*S*) (function)

S is a numerical semigroup. The output is the maximal denumerant of *S*, that is, the maximum of the maximal denumerants of the elements in *S* (see [BH13]).

Example
```
gap> s:=NumericalSemigroup(101,113,196,272,278,286);;
gap> MaximalDenumerantOfNumericalSemigroup(s);
4
```

9.2.16 AdjustmentOfNumericalSemigroup

▷ AdjustmentOfNumericalSemigroup(*S*) (function)

S is a numerical semigroup. The output is the adjustment of *S* as defined in [BH13].

Example
```
gap> s:=NumericalSemigroup(101,113,196,272,278,286);;
gap> AdjustmentOfNumericalSemigroup(s);
[ 0, 12, 24, 36, 48, 60, 72, 84, 95, 96, 107, 108, 119, 120, 131, 132, 143,
  144, 155, 156, 167, 168, 171, 177, 179, 180, 183, 185, 189, 190, 191, 192,
  195, 197, 201, 203, 204, 207, 209, 213, 215, 216, 219, 221, 225, 227, 228,
  231, 233, 237, 239, 240, 243, 245, 249, 251, 252, 255, 257, 261, 263, 264,
  266, 267, 269, 273, 275, 276, 279, 280, 281, 285, 287, 288, 292, 293, 299,
  300, 304, 305, 311, 312, 316, 317, 323, 324, 328, 329, 335, 336, 340, 341,
  342, 347, 348, 352, 353, 354, 356, 359, 360, 361, 362, 364, 365, 366, 368,
  370, 371, 372, 374, 376, 377, 378, 380, 382, 383, 384, 388, 389, 390, 394,
  395, 396, 400, 401, 402, 406, 407, 408, 412, 413, 414, 418, 419, 420, 424,
  425, 426, 430, 431, 432, 436, 437, 438, 442, 444, 448, 450, 451, 454, 456,
  460, 465, 466, 472, 477, 478, 484, 489, 490, 496, 501, 502, 508, 513, 514,
  519, 520, 525, 526, 527, 531, 532, 533, 537, 539, 543, 545, 549, 551, 555,
  561, 567, 573, 579, 585, 591, 597, 603, 609, 615, 621, 622, 627, 698, 704,
  710, 716, 722 ]
```

9.2.17 IsAdditiveNumericalSemigroup

▷ IsAdditiveNumericalSemigroup(*S*) (function)

`S` is a numerical semigroup. Detects if `S` is additive, that is, $ord(m+x) = ord(x)+1$ for all x in `S`, where m is the multiplicity of `S` and *ord* stands for MaximumDegreeOfElementWRTNumericalSemigroup. For these semigroups $gr_m(K[[S]])$ is Cohen-Macaulay(see [BH13]).

Example

```
gap> l:=IrreducibleNumericalSemigroupsWithFrobeniusNumber(31);;
gap> Length(l);
109
gap> Length(Filtered(l,IsAdditiveNumericalSemigroup));
20
```

9.2.18 IsSuperSymmetricNumericalSemigroup

▷ `IsSuperSymmetricNumericalSemigroup(S)` (function)

`S` is a numerical semigroup. Detects if `S` is supersymmetric, that is, it is symmetric, additive and whenever $w+w' = f+m$ (with m the multiplicity and f the Frobenius number) we have $ord(w+w') = ord(w)+ord(w')$, where *ord* stands for MaximumDegreeOfElementWRTNumericalSemigroup.

Example

```
gap> l:=IrreducibleNumericalSemigroupsWithFrobeniusNumber(31);;
gap> Length(l);
109
gap> Length(Filtered(l,IsSuperSymmetricNumericalSemigroup));
7
```

9.3 Invariants based on distances

This section is devoted to invariants that rely on the concept of distance between two factorizations.

9.3.1 CatenaryDegreeOfSetOfFactorizations

▷ `CatenaryDegreeOfSetOfFactorizations(ls)` (function)
▷ `CatenaryDegree(ls)` (operation)

`ls` is a set of factorizations (a list of lists of nonnegative integers with the same length). The output is the catenary degree of this set of factorizations.

Example

```
gap> FactorizationsIntegerWRTList(100,[11,13,15,19]);
[ [ 2, 6, 0, 0 ], [ 3, 4, 1, 0 ], [ 4, 2, 2, 0 ], [ 5, 0, 3, 0 ],
  [ 5, 2, 0, 1 ], [ 6, 0, 1, 1 ], [ 0, 1, 2, 3 ], [ 1, 1, 0, 4 ] ]
gap> CatenaryDegreeOfSetOfFactorizations(last);
5
```

9.3.2 AdjacentCatenaryDegreeOfSetOfFactorizations

▷ `AdjacentCatenaryDegreeOfSetOfFactorizations(ls)` (function)

`ls` is a set of factorizations. The output is the adjacent catenary degree of this set of factorizations, that is, the supremum of the distance between to sets of factorizations with adjacent lengths. More precisely, if $l_1,\ldots,l_t$ are the lengths of the factorizations of the elements in `ls`, and Z_{l_i} is the set of factorizations in `ls` with length l_i, then the adjacent catenary degree is the maximum of the distances $\mathrm{d}(Z_{l_i},Z_{l_{i+1}})$.

Example

```
gap> FactorizationsIntegerWRTList(100,[11,13,15,19]);
[ [ 2, 6, 0, 0 ], [ 3, 4, 1, 0 ], [ 4, 2, 2, 0 ], [ 5, 0, 3, 0 ], [ 5, 2, 0, 1 ],
  [ 6, 0, 1, 1 ], [ 0, 1, 2, 3 ], [ 1, 1, 0, 4 ] ]
gap> AdjacentCatenaryDegreeOfSetOfFactorizations(last);
5
```

9.3.3 EqualCatenaryDegreeOfSetOfFactorizations

▷ `EqualCatenaryDegreeOfSetOfFactorizations(ls)` (function)

`ls` is a set of factorizations. The same as CatenaryDegreeOfSetOfFactorizations, but now the factorizations joined by the chain must have the same length, and the elements in the chain also. Equivalently, if $l_1,\ldots,l_t$ are the lengths of the factorizations of the elements in `ls`, and Z_{l_i} is the set of factorizations in `ls` with length l_i, then the equal catenary degree is the maximum of the CatenaryDegreeOfSetOfFactorizations of $\mathrm{d}(Z_{l_i},Z_{l_{i+1}})$.

Example

```
gap> FactorizationsIntegerWRTList(100,[11,13,15,19]);
[ [ 2, 6, 0, 0 ], [ 3, 4, 1, 0 ], [ 4, 2, 2, 0 ], [ 5, 0, 3, 0 ], [ 5, 2, 0, 1 ],
  [ 6, 0, 1, 1 ], [ 0, 1, 2, 3 ], [ 1, 1, 0, 4 ] ]
gap> EqualCatenaryDegreeOfSetOfFactorizations(last);
2
```

9.3.4 MonotoneCatenaryDegreeOfSetOfFactorizations

▷ `MonotoneCatenaryDegreeOfSetOfFactorizations(ls)` (function)

`ls` is a set of factorizations. The same as CatenaryDegreeOfSetOfFactorizations, but now the factorizations are joined by a chain with nondecreasing lengths. Equivalently, it is the maximum of the AdjacentCatenaryDegreeOfSetOfFactorizations and the EqualCatenaryDegreeOfSetOfFactorizations.

Example

```
gap> FactorizationsIntegerWRTList(100,[11,13,15,19]);
[ [ 2, 6, 0, 0 ], [ 3, 4, 1, 0 ], [ 4, 2, 2, 0 ], [ 5, 0, 3, 0 ], [ 5, 2, 0, 1 ],
  [ 6, 0, 1, 1 ], [ 0, 1, 2, 3 ], [ 1, 1, 0, 4 ] ]
gap> MonotoneCatenaryDegreeOfSetOfFactorizations(last);
5
```

9.3.5 CatenaryDegreeOfElementInNumericalSemigroup

▷ `CatenaryDegreeOfElementInNumericalSemigroup(n, S)` (function)
▷ `CatenaryDegree(n, S)` (operation)
▷ `CatenaryDegree(S, n)` (operation)

`n` is a nonnegative integer and `S` is a numerical semigroup. The output is the catenary degree of `n` relative to `S`.

Example
```
gap> CatenaryDegreeOfElementInNumericalSemigroup(157,NumericalSemigroup(13,18));
0
gap> CatenaryDegreeOfElementInNumericalSemigroup(1157,NumericalSemigroup(13,18));
18
```

9.3.6 TameDegreeOfSetOfFactorizations

▷ `TameDegreeOfSetOfFactorizations(ls)` (function)

ls is a set of factorizations (a list of lists of nonnegative integers with the same length). The output is the tame degree of this set of factorizations.

Example
```
gap> FactorizationsIntegerWRTList(100,[11,13,15,19]);
[ [ 2, 6, 0, 0 ], [ 3, 4, 1, 0 ], [ 4, 2, 2, 0 ], [ 5, 0, 3, 0 ],
  [ 5, 2, 0, 1 ], [ 6, 0, 1, 1 ], [ 0, 1, 2, 3 ], [ 1, 1, 0, 4 ] ]
gap> TameDegreeOfSetOfFactorizations(last);
4
```

9.3.7 CatenaryDegreeOfNumericalSemigroup

▷ `CatenaryDegreeOfNumericalSemigroup(S)` (function)
▷ `CatenaryDegree(S)` (operation)

S is a numerical semigroup. The output is the catenary degree of *S*.

Example
```
gap> s:=NumericalSemigroup(101,113,196,272,278,286);
<Numerical semigroup with 6 generators>
gap> CatenaryDegreeOfNumericalSemigroup(s);
8
```

9.3.8 DegreesOffEqualPrimitiveElementsOfNumericalSemigroup

▷ `DegreesOffEqualPrimitiveElementsOfNumericalSemigroup(S)` (function)

S is a numerical semigroup.

The output is the set of elements s in *S* such that there exists a minimal solution to $msg \cdot x - msg \cdot y = 0$, such that x, y are factorizations with the same length of s, and msg is the minimal generating system of *S*. These elements are used to compute the equal catenary degree of *S*.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> DegreesOfEqualPrimitiveElementsOfNumericalSemigroup(s);
[ 3, 5, 7, 10 ]
```

9.3.9 EqualCatenaryDegreeOfNumericalSemigroup

▷ `EqualCatenaryDegreeOfNumericalSemigroup(S)` (function)

S is a numerical semigroup. The output is the equal catenary degree of *S*.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> EqualCatenaryDegreeOfNumericalSemigroup(s);
2
```

9.3.10 DegreesOfMonotonePrimitiveElementsOfNumericalSemigroup

▷ `DegreesOfMonotonePrimitiveElementsOfNumericalSemigroup(S)` (function)

S is a numerical semigroup.

The output is the set of elements s in S such that there exists a minimal solution to $msg \cdot x - msg \cdot y = 0$, such that x, y are factorizations of s, with $|x| \leq |y|$; msg stands the minimal generating system of S. These elements are used to compute the monotone catenary degree of S.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> DegreesOfMonotonePrimitiveElementsOfNumericalSemigroup(s);
[ 3, 5, 7, 10, 12, 14, 15, 21, 28, 35 ]
```

9.3.11 MonotoneCatenaryDegreeOfNumericalSemigroup

▷ `MonotoneCatenaryDegreeOfNumericalSemigroup(S)` (function)

S is a numerical semigroup. The output is the monotone catenary degree of S.

Example
```
gap> s:=NumericalSemigroup(10,23,31,44);;
gap> CatenaryDegreeOfNumericalSemigroup(s);
9
gap> MonotoneCatenaryDegreeOfNumericalSemigroup(s);
21
```

9.3.12 TameDegreeOfNumericalSemigroup

▷ `TameDegreeOfNumericalSemigroup(S)` (function)

S is a numerical semigroup. The output is the tame degree of S.

Example
```
gap> s:=NumericalSemigroup(101,113,196,272,278,286);
<Numerical semigroup with 6 generators>
gap> TameDegreeOfNumericalSemigroup(s);
14
```

9.3.13 TameDegreeOfElementInNumericalSemigroup

▷ `TameDegreeOfElementInNumericalSemigroup(n, S)` (function)

n is an element of the numerical semigroup S. The output is the tame degree of n in S.

Example
```
gap> s:=NumericalSemigroup(10,11,13);
<Numerical semigroup with 3 generators>
gap> TameDegreeOfElementInNumericalSemigroup(100,s);
5
```

9.4 Primality

There are no primes among the irreducible elements (minimal generators) of a numerical semigroup. However, there is a way to measure how far an element is frome being prime: the ω-primality.

9.4.1 OmegaPrimalityOfElementInNumericalSemigroup

▷ `OmegaPrimalityOfElementInNumericalSemigroup(n, S)` (function)

n is an element of the numerical semigroup *S*. The output is the ω-primality of *n* in *S* as explained in [BGSG11]. The current implementation is due to Chris O'Neill based on a work in progress with Pelayo and Thomas.

Example
```
gap> s:=NumericalSemigroup(10,11,13);
<Numerical semigroup with 3 generators>
gap> OmegaPrimalityOfElementInNumericalSemigroup(100,s);
13
```

9.4.2 OmegaPrimalityOfElementListInNumericalSemigroup

▷ `OmegaPrimalityOfElementListInNumericalSemigroup(l, S)` (function)

S is a numerical semigroup and *l* a list of elements of *S*.
Computes the omega-values of all the elements in *l*.

Example
```
gap> s:=NumericalSemigroup(10,11,13);;
gap> l:=FirstElementsOfNumericalSemigroup(100,s);;
gap> List(l,x->OmegaPrimalityOfElementInNumericalSemigroup(x,s)); time;
[ 0, 4, 5, 5, 4, 6, 7, 6, 6, 6, 6, 7, 8, 7, 7, 7, 7, 7, 8, 7, 8, 9, 8, 8, 8,
  8, 8, 8, 8, 9, 9, 10, 9, 9, 9, 9, 9, 9, 9, 9, 10, 11, 10, 10, 10, 10, 10,
  10, 10, 10, 11, 12, 11, 11, 11, 11, 11, 11, 11, 11, 12, 13, 12, 12, 12, 12,
  12, 12, 12, 12, 13, 14, 13, 13, 13, 13, 13, 13, 13, 13, 14, 15, 14, 14, 14,
  14, 14, 14, 14, 14, 15, 16, 15, 15, 15, 15, 15, 15, 15, 15 ]
218
gap> OmegaPrimalityOfElementListInNumericalSemigroup(l,s);time;
[ 0, 4, 5, 5, 4, 6, 7, 6, 6, 6, 6, 7, 8, 7, 7, 7, 7, 7, 8, 7, 8, 9, 8, 8, 8,
  8, 8, 8, 8, 9, 9, 10, 9, 9, 9, 9, 9, 9, 9, 9, 10, 11, 10, 10, 10, 10, 10,
  10, 10, 10, 11, 12, 11, 11, 11, 11, 11, 11, 11, 11, 12, 13, 12, 12, 12, 12,
  12, 12, 12, 12, 13, 14, 13, 13, 13, 13, 13, 13, 13, 13, 14, 15, 14, 14, 14,
  14, 14, 14, 14, 14, 15, 16, 15, 15, 15, 15, 15, 15, 15, 15 ]
10
```

9.4.3 OmegaPrimalityOfNumericalSemigroup

▷ OmegaPrimalityOfNumericalSemigroup(S) (function)

S is a numerical semigroup. The output is the maximum of the ω-primalities of the minimal generators of S.

Example
```
gap> s:=NumericalSemigroup(10,11,13);
<Numerical semigroup with 3 generators>
gap> OmegaPrimalityOfNumericalSemigroup(s);
5
```

9.5 Homogenization of Numerical Semigroups

Let S be a numerical semigroup minimally generated by $\{m_1,\ldots,m_n\}$. The homogenization of S, S^{hom} is the semigroup generated by $\{(1,0),(1,m_1),\ldots,(1,m_n)\}$. The catenary degree of S^{hom} coincides with the homogeneous catenary degree of S, and it is between the catenary and the monotone catenary degree of S. The advantage of this catenary degree is that is less costly to compute than the monotone catenary degree, and has some nice interpretations ([GSOSRN13]). This section contains the auxiliary functions needed to compute the homogeneous catenary degree.

9.5.1 BelongsToHomogenizationOfNumericalSemigroup

▷ BelongsToHomogenizationOfNumericalSemigroup(n, S) (function)

S is a numerical semigroup and n a list with two entries (a pair). The output is true if the n belongs to the homogenization of S.

Example
```
gap> s:=NumericalSemigroup(10,11,13);;
gap> BelongsToHomogenizationOfNumericalSemigroup([10,23],s);
true
gap> BelongsToHomogenizationOfNumericalSemigroup([1,23],s);
false
```

9.5.2 FactorizationsInHomogenizationOfNumericalSemigroup

▷ FactorizationsInHomogenizationOfNumericalSemigroup(n, S) (function)

S is a numerical semigroup and n a list with two entries (a pair). The output is the set of factorizations n in terms of the minimal generating system of the homogenization of S.

Example
```
gap> s:=NumericalSemigroup(10,11,13);;
gap> FactorizationsInHomogenizationOfNumericalSemigroup([20,230],s);
[ [ 0, 0, 15, 5 ], [ 0, 2, 12, 6 ], [ 0, 4, 9, 7 ],
  [ 0, 6, 6, 8 ], [ 0, 8, 3, 9 ], [ 0, 10, 0, 10 ],
  [ 1, 1, 7, 11 ], [ 1, 3, 4, 12 ], [ 1, 5, 1, 13 ],
  [ 2, 0, 2, 16 ] ]
gap> FactorizationsElementWRTNumericalSemigroup(230,s);
[ [ 23, 0, 0 ], [ 12, 10, 0 ], [ 1, 20, 0 ], [ 14, 7, 1 ],
```

```
  [ 3, 17, 1 ], [ 16, 4, 2 ], [ 5, 14, 2 ], [ 18, 1, 3 ],
  [ 7, 11, 3 ], [ 9, 8, 4 ], [ 11, 5, 5 ], [ 0, 15, 5 ],
  [ 13, 2, 6 ], [ 2, 12, 6 ], [ 4, 9, 7 ], [ 6, 6, 8 ],
  [ 8, 3, 9 ], [ 10, 0, 10 ], [ 1, 7, 11 ], [ 3, 4, 12 ],
  [ 5, 1, 13 ], [ 0, 2, 16 ] ]
```

9.5.3 HomogeneousBettiElementsOfNumericalSemigroup

▷ HomogeneousBettiElementsOfNumericalSemigroup(*S*) (function)

S is a numerical semigroup. The output is the set of Betti elements of the homogenization of *S*.

Example

```
gap> s:=NumericalSemigroup(10,17,19);;
gap> BettiElementsOfNumericalSemigroup(s);
[ 57, 68, 70 ]
gap> HomogeneousBettiElementsOfNumericalSemigroup(s);
[ [ 5, 57 ], [ 5, 68 ], [ 6, 95 ], [ 7, 70 ], [ 9, 153 ] ]
```

9.5.4 HomogeneousCatenaryDegreeOfNumericalSemigroup

▷ HomogeneousCatenaryDegreeOfNumericalSemigroup(*S*) (function)

S is a numerical semigroup. The output is the homogeneous catenary degree of *S*. Observe that for a single element in the homogenization of *S*, its catenary degree can be computed with CatenaryDegreeOfSetOfFactorizations and FactorizationsInHomogenizationOfNumericalSemigroup.

Example

```
gap> s:=NumericalSemigroup(10,17,19);;
gap> CatenaryDegreeOfNumericalSemigroup(s);
7
gap> HomogeneousCatenaryDegreeOfNumericalSemigroup(s);
9
```

9.6 Divisors, posets

Given a numerical semigroup S and two integers a,b, we write $a \leq_S b$ if $b-a \in S$. We also say that a divides b (with respect to S). The semigroup S with this binary relation is a poset.

The set of divisors of n in S will be denoted by $\mathrm{D}_S(n)$. If we are given $n_1,\dots,n_r \in S$, the set of the divisors of these elements is $\mathrm{D}(n_1,\dots,n_r) = \bigcup_{i=1}^{r} \mathrm{D}(n_i)$.

9.6.1 MoebiusFunctionAssociatedToNumericalSemigroup

▷ MoebiusFunctionAssociatedToNumericalSemigroup(*S*, *n*) (function)

S is a numerical semigroup and *n* is an integer. As $(S,\leq_S)$ is a poset, we can define the Möbius function associated to it as in [CRA13]. The output is the value of the Möbius function in the integer *n*, that is, the alternate sum of the number of chains from 0 to *n*.

Example

```
gap> s:=NumericalSemigroup(3,5,7);;
gap> MoebiusFunctionAssociatedToNumericalSemigroup(s,10);
2
gap> MoebiusFunctionAssociatedToNumericalSemigroup(s,34);
25
```

9.6.2 DivisorsOfElementInNumericalSemigroup

▷ `DivisorsOfElementInNumericalSemigroup(S, n)` (operation)

`S` is a numerical semigroup and `n` is an integer. The arguments can also be given as `n, S`. The output is the set of divisors of `n` in `S`.

Example

```
gap> s:=NumericalSemigroup(5,7,11);;
gap> DivisorsOfElementInNumericalSemigroup(s,20);
[ 0, 5, 10, 15, 20 ]
gap> DivisorsOfElementInNumericalSemigroup(20,s);
[ 0, 5, 10, 15, 20 ]
```

9.7 Feng-Rao distances and numbers

Let S be a numerical semigroup and let $n \in S$. The Feng-Rao distance of n is then defined as $\delta_S(n) = \min\{\#\mathrm{D}(x) \mid n \leq x,\ x \in S\}$.

The rth generalized distance is $\delta_S^r(n) = \{\#\mathrm{D}(n_1,\ldots,n_r) \mid n \leq n_1 < \cdots < n_r,\ n_i \in S\}$.

9.7.1 FengRaoDistance

▷ `FengRaoDistance(S, r)` (function)

`S` is a numerical semigroup, `r` and `m` integers. The output is the `r`-th Feng-Rao distance of the element `m` in the numerical semigroup `S`.

Example

```
gap> S := NumericalSemigroup(7,9,17);;
gap> FengRaoDistance(S,6,100);
86
```

9.7.2 FengRaoNumber

▷ `FengRaoNumber(S, r)` (operation)

`S` is a numerical semigroup and `r` is an integer. The output is the `r`-th Feng-Rao number of the numerical semigroup `S`.

Example

```
gap> S := NumericalSemigroup(7,8,17);;
gap> FengRaoNumber(S,209);
224
```

```
gap> FengRaoNumber(209,S);
224
```

Chapter 10

Polynomials and numerical semigroups

Polynomials appear related to numerical semigroups in several ways. One of them is through their associated generating function (or Hilbert series), and another via value semigroups of a curve; and curves might be defined by polynomials. In this chapter we present several functions to compute the polynomial and Hilbert series associated to a numerical semigroup, and to calculate the respective numerical semigroups given a set of defining polynomials.

10.1 Generating functions or Hilbert series

Let S be a numerical semigroup. The Hilbert series or generating function associated to S is $H_S(x) = \sum_{s\in S} x^s$ (actually it is the Hilbert function of the ring $K[S]$ with K a field). See for instance [Mor14].

10.1.1 NumericalSemigroupPolynomial

▷ `NumericalSemigroupPolynomial(s, x)` (function)

`s` is a numerical semigroups and `x` a variable (or a value to evaluate in). The output is the polynomial $1+(x-1)\sum_{s\in \mathbb{N}\setminus S} x^s$, which equals $(1-x)H_S(x)$.

Example
```
gap> x:=X(Rationals,"x");;
gap> s:=NumericalSemigroup(5,7,9);;
gap> NumericalSemigroupPolynomial(s,x);
x^14-x^13+x^12-x^11+x^9-x^8+x^7-x^6+x^5-x+1
```

10.1.2 IsNumericalSemigroupPolynomial

▷ `IsNumericalSemigroupPolynomial(f)` (function)

`f` is a polynomial in one variable. The output is true if there exists a numerical semigroup S such that `f` equals $(1-x)H_S(x)$, that is, the polynomial associated to S (false otherwise).

Example
```
gap> x:=X(Rationals,"x");;
gap> s:=NumericalSemigroup(5,6,7,8);;
gap> f:=NumericalSemigroupPolynomial(s,x);
x^10-x^9+x^5-x+1
```

```
gap> IsNumericalSemigroupPolynomial(f);
true
```

10.1.3 NumericalSemigroupFromNumericalSemigroupPolynomial

▷ NumericalSemigroupFromNumericalSemigroupPolynomial(*f*) (function)

f is a polynomial associated to a numerical semigroup (otherwise yields error). The output is the numerical semigroup S such that *f* equals $(1-x)H_S(x)$.

Example

```
gap> x:=X(Rationals,"x");;
gap> s:=NumericalSemigroup(5,6,7,8);;
gap> f:=NumericalSemigroupPolynomial(s,x);
x^10-x^9+x^5-x+1
gap> NumericalSemigroupFromNumericalSemigroupPolynomial(f)=s;
true
```

10.1.4 HilbertSeriesOfNumericalSemigroup

▷ HilbertSeriesOfNumericalSemigroup(*s*, *x*) (function)

s is a numerical semigroup and *x* a variable (or a value to evaluate in). The output is the series $\sum_{s\in S} x^s$. The series is given as a rational function.

Example

```
gap> x:=X(Rationals,"x");;
gap> s:=NumericalSemigroup(5,7,9);;
gap> HilbertSeriesOfNumericalSemigroup(s,x);
(x^14-x^13+x^12-x^11+x^9-x^8+x^7-x^6+x^5-x+1)/(-x+1)
```

10.1.5 GraeffePolynomial

▷ GraeffePolynomial(*p*) (function)

p is a polynomial. Computes the Graeffe polynomial of *p*. Needed to test if *p* is a cyclotomic polynomial (see [BD89]).

Example

```
gap> x:=Indeterminate(Rationals,1);; SetName(x,"x");
gap> GraeffePolynomial(x^2-1);
x^2-2*x+1
```

10.1.6 IsCyclotomicPolynomial

▷ IsCyclotomicPolynomial(*p*) (function)

p is a polynomial. Detects if *p* is a cyclotomic polynomial using the procedure given in [BD89].

Example

```
gap> CyclotomicPolynomial(Rationals,3);
x^2+x+1
```

```
gap> IsCyclotomicPolynomial(last);
true
```

10.1.7 IsKroneckerPolynomial

▷ IsKroneckerPolynomial(*p*) (function)

p is a polynomial. Detects if *p* is a Kronecker polynomial, that is, a monic polynomial with integer coefficients having all its roots in the unit circumference, or equivalently, a product of cyclotomic polynomials. The current implementation has been done with A. Herrera-Poyatos, following [BD89].

Example
```
gap> x:=X(Rationals,"x");;
gap>  s:=NumericalSemigroup(3,5,7);;
gap>  t:=NumericalSemigroup(4,6,9);;
gap> p:=NumericalSemigroupPolynomial(s,x);
x^5-x^4+x^3-x+1
gap> q:=NumericalSemigroupPolynomial(t,x);
x^12-x^11+x^8-x^7+x^6-x^5+x^4-x+1
gap> IsKroneckerPolynomial(p);
false
gap> IsKroneckerPolynomial(q);
true
```

10.1.8 IsCyclotomicNumericalSemigroup

▷ IsCyclotomicNumericalSemigroup(*s*) (function)

s is a numerical semigroup. Detects if the polynomial associated to *s* is a Kronecker polynomial.

Example
```
gap> l:=CompleteIntersectionNumericalSemigroupsWithFrobeniusNumber(21);;
gap> ForAll(l,IsCyclotomicNumericalSemigroup);
true
```

10.1.9 IsSelfReciprocalUnivariatePolynomial

▷ IsSelfReciprocalUnivariatePolynomial(*p*) (function)

p is a univariate polynomial. Detects if *p* is selfreciprocal. A numerical semigroup is symmetric if and only if it is selfreciprocal, [Mor14]. The current implementation is due to A. Herrera-Poyatos.

Example
```
gap> l:=IrreducibleNumericalSemigroupsWithFrobeniusNumber(13);;
gap> x:=X(Rationals,"x");;
gap> ForAll(l, s->
> IsSelfReciprocalUnivariatePolynomial(NumericalSemigroupPolynomial(s,x)));
true
```

10.2 Semigroup of values of algebraic curves

Let $f(x,y) \in \mathbb{K}[x,y]$, with $\mathbb{K}$ an algebraically closed field of characteristic zero. Let $f(x,y) = y^n + a_1(x)y^{n-1} + \ldots + a_n(x)$ be a nonzero polynomial of $\mathbb{K}[x][y]$. After possibly a change of variables, we may assume that, that $\deg_x(a_i(x)) \leq i-1$ for all $i \in \{1,\ldots,n\}$. For $g \in \mathbb{K}[x,y]$ that is not a multiple of f, define $\mathrm{int}(f,g) = \dim_{\mathbb{K}} \frac{\mathbb{K}[x,y]}{(f,g)}$. If f has one place at infinity, then the set $\{\mathrm{int}(f,g) \mid g \in \mathbb{K}[x,y] \setminus (f)\}$ is a free numerical semigroup (and thus a complete intersection).

10.2.1 SemigroupOfValuesOfPlaneCurveWithSinglePlaceAtInfinity

▷ `SemigroupOfValuesOfPlaneCurveWithSinglePlaceAtInfinity(f)` (function)

`f` is a polynomial in the variables X(Rationals,1) and X(Rationals,2). Computes the semigroup $\{\mathrm{int}(f,g) \mid g \in \mathbb{K}[x,y] \setminus (f)\}$, where $\mathrm{int}(f,g) = \dim_{\mathbb{K}}(\mathbb{K}[x,y]/(f,g))$. The algorithm checks if `f` has one place at infinity. If the extra argument "all" is given, then the output is the δ-sequence and approximate roots of `f`. The method is explained in [AGS16].

Example
```
gap> x:=Indeterminate(Rationals,1);; SetName(x,"x");
gap> y:=Indeterminate(Rationals,2);; SetName(y,"y");
gap> f:=((y^3-x^2)^2-x*y^2)^4-(y^3-x^2);;
gap> SemigroupOfValuesOfPlaneCurveWithSinglePlaceAtInfinity(f,"all");
[ [ 24, 16, 28, 7 ], [ y, y^3-x^2, y^6-2*x^2*y^3+x^4-x*y^2 ] ]
```

10.2.2 IsDeltaSequence

▷ `IsDeltaSequence(l)` (function)

`l` is a list of positive integers. Assume that `l` equals $a_0, a_1, \ldots, a_h$. Then `l` is a δ-sequence if $\gcd(a_0,\ldots,a_h) = 1$, $\langle a_0,\cdots,a_s\rangle$ is free, $a_kD_k > a_{k+1}D_{k+1}$ and $a_0 > a_1 > D_2 > D_3 > \ldots > D_{h+1}$, where $D_1 = a_0$, $D_k = \gcd(D_{k-1}, a_{k-1})$.

Every δ-sequence generates a numerical semigroup that is the semigroup of values of a plane curve with one place at infinity.

Example
```
gap> IsDeltaSequence([24,16,28,7]);
true
```

10.2.3 DeltaSequencesWithFrobeniusNumber

▷ `DeltaSequencesWithFrobeniusNumber(f)` (function)

`f` is an integer. Computes the set of all δ-sequences generating numerical semigroups with Frobenius number `f`.

Example
```
gap> DeltaSequencesWithFrobeniusNumber(21);
[ [ 8, 6, 11 ], [ 10, 4, 15 ], [ 12, 8, 6, 11 ], [ 14, 4, 11 ],
  [ 15, 10, 4 ], [ 23, 2 ] ]
```

10.2.4 CurveAssociatedToDeltaSequence

▷ CurveAssociatedToDeltaSequence(*l*) (function)

l is a δ-sequence. Computes a curve in the variables X(Rationals,1) and X(Rationals,2) whose semigroup of values is generated by the *l*.

Example
```
gap> CurveAssociatedToDeltaSequence([24,16,28,7]);
y^24-8*x^2*y^21+28*x^4*y^18-56*x^6*y^15-4*x*y^20+70*x^8*y^12+24*x^3*y^17-56*x^\
10*y^9-60*x^5*y^14+28*x^12*y^6+80*x^7*y^11+6*x^2*y^16-8*x^14*y^3-60*x^9*y^8-24\
*x^4*y^13+x^16+24*x^11*y^5+36*x^6*y^10-4*x^13*y^2-24*x^8*y^7-4*x^3*y^12+6*x^10\
*y^4+8*x^5*y^9-4*x^7*y^6+x^4*y^8-y^3+x^2
gap> SemigroupOfValuesOfPlaneCurveWithSinglePlaceAtInfinity(last,"all");
[ [ 24, 16, 28, 7 ], [ y, y^3-x^2, y^6-2*x^2*y^3+x^4-x*y^2 ] ]
```

10.2.5 SemigroupOfValuesOfPlaneCurve

▷ SemigroupOfValuesOfPlaneCurve(*f*) (function)

f is a polynomial in the variables X(Rationals,1) and X(Rationals,2). The singular package is mandatory. Either by loading it prior to numerical semigroups or by using *NumSgpsUseSingular()*. If f is irreducible, computes the semigroup $\{\mathrm{int}(f,g) \mid g \in \mathbb{K}[x,y] \setminus (f)\}$, where $\mathrm{int}(f,g) = \dim_{\mathbb{K}}(\mathbb{K}[\![x,y]\!]/(f,g))$. If it has two components, the output is the value semigroup in two variables, and thus a good semigroup. If there are more components, then the output is that of *semigroup* in the alexpoly singular library.

Example
```
gap> x:=X(Rationals,"x");; y:=X(Rationals,"y");;
gap> f:= y^4-2*x^3*y^2-4*x^5*y+x^6-x^7;
-x^7+x^6-4*x^5*y-2*x^3*y^2+y^4
gap> SemigroupOfValuesOfPlaneCurve(f);
<Numerical semigroup with 3 generators>
gap> MinimalGenerators(last);
[ 4, 6, 13 ]
gap> f:=(y^4-2*x^3*y^2-4*x^5*y+x^6-x^7)*(y^2-x^3);;
gap> SemigroupOfValuesOfPlaneCurve(f);
<Good semigroup>
gap> MinimalGenerators(last);
[ [ 4, 2 ], [ 6, 3 ], [ 13, 15 ], [ 29, 13 ] ]
```

10.2.6 SemigroupOfValuesOfCurve_Local

▷ SemigroupOfValuesOfCurve_Local(*arg*) (function)

The function admits one or two parameters. In any case, the first is a list of polynomials *pols*. And the second can be the string *"basis"* or an integer *val*.

If only one argument is given, the output is the semigroup of all possible orders of $K[[pols]]$ provided that $K[[x]]/K[[pols]]$ has finite length. If the second argument *"basis"* is given, then the output is a (reduced) basis of the algebra $K[[pols]]$ such that the orders of the basis elements generate minimally the semigroup of orders of $K[[pols]]$. If an integer *val* is the second argument, then the output is a polynomial in $K[[pols]]$ with order `val` (`fail` if there is no such polynomial, that is, `val` is not in the semigroup of values).

The method is explained in [AGSM17].

Example
```
gap> x:=Indeterminate(Rationals,"x");;
gap> SemigroupOfValuesOfCurve_Local([x^4,x^6+x^7,x^13]);
<Numerical semigroup with 4 generators>
gap> MinimalGeneratingSystem(last);
[ 4, 6, 13, 15 ]
gap> SemigroupOfValuesOfCurve_Local([x^4,x^6+x^7,x^13], "basis");
[ x^4, x^7+x^6, x^13, x^15 ]
gap> SemigroupOfValuesOfCurve_Local([x^4,x^6+x^7,x^13], 20);
x^20
```

10.2.7 SemigroupOfValuesOfCurve_Global

▷ `SemigroupOfValuesOfCurve_Global(arg)` (function)

The function admits one or two parameters. In any case, the first is a list of polynomials `pols`. And the second can be the string `"basis"` or an integer `val`.

If only one argument is given, the output is the semigroup of all possible degrees of $K[pols]$ provided that $K[x]/K[pols]$ has finite length. If the second argument `"basis"` is given, then the output is a (reduced) basis of the algebra $K[pols]$ such that the degrees of the basis elements generate minimally the semigroup of degrees of $K[pols]$. If an integer `val` is the second argument, then the output is a polynomial in $K[pols]$ with degree `val` (`fail` if there is no such polynomial, that is, `val` is not in the semigroup of values).

The method is explained in [AGSM17].

Example
```
gap> x:=Indeterminate(Rationals,"x");;
gap> SemigroupOfValuesOfCurve_Global([x^4,x^6+x^7,x^13]);
<Numerical semigroup with 3 generators>
gap> MinimalGeneratingSystem(last);
[ 4, 7, 13 ]
gap> SemigroupOfValuesOfCurve_Global([x^4,x^6+x^7,x^13],"basis");
[ x^4, x^7+x^6, x^13 ]
gap> SemigroupOfValuesOfCurve_Global([x^4,x^6+x^7,x^13],12);
x^12
gap> SemigroupOfValuesOfCurve_Global([x^4,x^6+x^7,x^13],6);
fail
```

10.2.8 GeneratorsModule_Global

▷ `GeneratorsModule_Global(A, M)` (function)

`A` and `M` are lists of polynomials in the same variable. The output is a basis of the ideal $MK[A]$, that is, a set F such that $deg(F)$ generates the ideal $deg(MK[A])$ of $deg(K[A])$, where deg stands for degree. The method is explained in [AAGS17].

Example
```
gap> t:=Indeterminate(Rationals,"t");;
gap> A:=[t^6+t,t^4];;
```

```
gap> M:=[t^3,t^4];;
gap> GeneratorsModule_Global(A,M);
[ t^3, t^4, t^5, t^6 ]
```

10.2.9 GeneratorsKahlerDifferentials

▷ `GeneratorsKahlerDifferentials(A, M)` (function)

A is a list of polynomials in the same variable. The output is *GeneratorsModule_Global(A,M)*, with *M* the set of derivatives of the elements in *A*.

Example
```
gap> t:=Indeterminate(Rationals,"t");;
gap> GeneratorsKahlerDifferentials([t^3,t^4]);
[ t^2, t^3 ]
```

10.2.10 IsMonomialNumericalSemigroup

▷ `IsMonomialNumericalSemigroup(S)` (property)

S is a numerical semigroup. Tests whether *S* a monomial numerical semigroup.

Let R a Noetherian ring such that $K \subseteq R \subseteq K[[t]]$, K is a field of characteristic zero, the algebraic closure of R is $K[[t]]$, and the conductor $(R : K[[t]])$ is not zero. If $v : K((t)) \to \mathbb{Z}$ is the natural valuation for $K((t))$, then $v(R)$ is a numerical semigroup.

Let S be a numerical semigroup minimally generated by $\{n_1,\ldots,n_e\}$. The semigroup ring associated to S is $K[[S]] = K[[t^{n_1},\ldots,t^{n_e}]]$. A ring is called a semigroup ring if it is of the form $K[[S]]$, for some numerical semigroup S. We say that S is a monomial numerical semigroup if for any R as above with $v(R) = S$, R is a semigroup ring. See [Mic02] for details.

Example
```
gap> IsMonomialNumericalSemigroup(NumericalSemigroup(4,6,7));
true
gap> IsMonomialNumericalSemigroup(NumericalSemigroup(4,6,11));
false
```

Chapter 11

Affine semigroups

An *affine semigroup* S is a finitely generated cancellative monoid that is reduced (no units other than 0) and is torsion-free ($as = bs$ implies $a = b$, with $a, b \in \mathbb{N}$ and $s \in S$). Up to isomorphism any affine semigroup can be viewed as a finitely generated submonoid of $\mathbb{N}^k$ for some positive integer k. Thus affine semigroups are a natural generalization of numerical semigroups.

Some of the functions in this chapter may work considerably faster when some external package is installed and its use is allowed. When this is the case, it is referred in the function documentation. We refer the user to Chapter 13 for details on the use of external packages.

11.1 Defining affine semigroups

The most common way to give an affine semigroup is by any of its systems of generators. As for numerical semigroups, any affine semigroup admits a unique minimal system of generators. A system of generators can be represented as a list of lists of nonnegative integers; all lists in the list having the same length (a matrix actually). If G is a subgroup of $\mathbb{Z}^k$, then $S = G \cap \mathbb{N}^k$ is an affine semigroup (these semigroups are called *full affine semigroups*). As G can be represented by its defining equations (homogeneous and some of them possibly in congruences), we can represent S by the defining equations of G; indeed S is just the set of nonnegative solutions of this system of equations. We can represent the equations as a list of lists of integers, all with the same length. Every list is a row of the matrix of coefficients of the system of equations. For the equations in congruences, if we arrange them so that they are the first ones in the list, we provide the corresponding moduli in a list. So for instance, the equations $x + y \equiv 0 \bmod 2$, $x - 2y = 0$ will be represented as [[1,1],[1,-2]] and the moduli [2].

As happens with numerical semigroups, there are different ways to specify an affine semigroup S, namely, by means of a system of generators, a system of homogeneous linear Diophantine equations or a system of homogeneous linear Diophantine inequalities, just to mention some. In this section we describe functions that may be used to specify, in one of these ways, an affine semigroup in GAP.

11.1.1 AffineSemigroupByGenerators

▷ `AffineSemigroupByGenerators(List)` (function)
▷ `AffineSemigroup(String, List)` (function)

`List` is a list of n-tuples of nonnegative integers, if the semigroup to be created is n-dimensional. The n-tuples may be given as a list or by a sequence of individual elements. The output is the affine semigroup spanned by `List`.

`String` does not need to be present. When it is present, it must be `"generators"` and `List` must be a list, not a sequence of individual elements.

Example
```
gap> s1 := AffineSemigroupByGenerators([1,3],[7,2],[1,5]);
<Affine semigroup in 2 dimensional space, with 3 generators>
gap> s2 := AffineSemigroupByGenerators([[1,3],[7,2],[1,5]]);;
gap> s3 := AffineSemigroup("generators",[[1,3],[7,2],[1,5]]);;
gap> s4 := AffineSemigroup([1,3],[7,2],[1,5]);;
gap> s5 := AffineSemigroup([[1,3],[7,2],[1,5]]);;
gap> Length(Set([s1,s2,s3,s4,s5]));
1
```

11.1.2 AffineSemigroupByEquations

▷ `AffineSemigroupByEquations(List)` (function)
▷ `AffineSemigroup(String, List)` (function)

`List` is a list with two components. The first represents a matrix with integer coefficients, say $A=(a_{ij})$, and so it is a list of lists of integers all with the same length. The second component is a list of positive integers, say $d=(d_i)$, which may be empty. The list d must be of length less than or equal to the length of A (number of rows of A).

The output is the full semigroup of nonnegative integer solutions to the system of homogeneous equations
$a_{11}x_1+\cdots+a_{1n}x_n \equiv 0 \bmod d_1$,
$\ldots$
$a_{k1}x_1+\cdots+a_{kn}x_n \equiv 0 \bmod d_k$,
$a_{k+1\,1}x_1+\cdots+a_{k+1\,n} = 0$,
$\ldots$
$a_{m1}x_1+\cdots+a_{mn}x_n = 0$.

If d is empty, then there will be no equations in congruences.

As pointed at the beginning of the section, the equations $x+y \equiv 0 \bmod 2,\ x-2y=0$ will be represented as A equal to [[1,1],[1,-2]] and the moduli d equal to [2].

In the second form, `String` must be `"equations"`.

Example
```
gap> s1 := AffineSemigroupByEquations([[[-2,1]],[3]]);
<Affine semigroup>
gap> s2 := AffineSemigroup("equations",[[[1,1]],[3]]);
<Affine semigroup>
gap> s1=s2;
true
```

11.1.3 AffineSemigroupByInequalities

▷ `AffineSemigroupByInequalities(List)` (function)
▷ `AffineSemigroup(String, List)` (function)

List is a list of lists (a matrix) of integers that represents a set of inequalities.

Returns the (normal) affine semigroup of nonnegative integer solutions of the system of inequalities $List \times X \geq 0$.

In the second form, String must be "inequalities".

Example
```
gap> a1:=AffineSemigroupByInequalities([[2,-1],[-1,3]]);
<Affine semigroup>
gap> a2:=AffineSemigroup("inequalities",[[2,-1],[-1,3]]);
<Affine semigroup>
gap> a1=a2;
true
```

11.1.4 Generators (for affine semigroup)

▷ Generators(*S*) (function)
▷ GeneratorsOfAffineSemigroup(*S*) (function)

S is an affine semigroup, the output is a system of generators.

Example
```
gap> a:=AffineSemigroup([[1,0],[0,1],[1,1]]);
<Affine semigroup in 2 dimensional space, with 3 generators>
gap> Generators(a);
[ [ 0, 1 ], [ 1, 0 ], [ 1, 1 ] ]
```

11.1.5 MinimalGenerators (for affine semigroup)

▷ MinimalGenerators(*S*) (function)
▷ MinimalGeneratingSystem(*S*) (function)

S is an affine semigroup, the output is its system of minimal generators.

Example
```
gap> a:=AffineSemigroup([[1,0],[0,1],[1,1]]);
<Affine semigroup in 2 dimensional space, with 3 generators>
  gap> MinimalGenerators(a);
  [ [ 0, 1 ], [ 1, 0 ] ]
```

11.1.6 AsAffineSemigroup

▷ AsAffineSemigroup(*S*) (function)

S is a numerical semigroup, the output is *S* regarded as an affine semigroup.

Example
```
gap> s:=NumericalSemigroup(1310,1411,1546,1601);
<Numerical semigroup with 4 generators>
gap> MinimalPresentationOfNumericalSemigroup(s);;time;
2960
gap> a:=AsAffineSemigroup(s);
<Affine semigroup in 1 dimensional space, with 4 generators>
```

```
gap> GeneratorsOfAffineSemigroup(a);
[ [ 1310 ], [ 1411 ], [ 1546 ], [ 1601 ] ]
gap> MinimalPresentationOfAffineSemigroup(a);;time;
237972
```

If we use the package `SingularInterface`, the speed up is considerable.

Example
```
gap> NumSgpsUseSingularInterface();
...
gap> MinimalPresentationOfAffineSemigroup(a);;time;
32
```

11.1.7 IsAffineSemigroup

▷ `IsAffineSemigroup(AS)` (attribute)
▷ `IsAffineSemigroupByGenerators(AS)` (attribute)
▷ `IsAffineSemigroupByEquations(AS)` (attribute)
▷ `IsAffineSemigroupByInequalities(AS)` (attribute)

AS is an affine semigroup and these attributes are available (their names should be self explanatory). They reflect what is currently known about the semigroup.

Example
```
gap> a1:=AffineSemigroup([[3,0],[2,1],[1,2],[0,3]]);
<Affine semigroup in 2 dimensional space, with 4 generators>
gap> IsAffineSemigroupByEquations(a1);
false
gap> IsAffineSemigroupByGenerators(a1);
true
gap> ns := NumericalSemigroup(3,5);
<Numerical semigroup with 2 generators>
gap> IsAffineSemigroup(ns);
false
gap> as := AsAffineSemigroup(ns);
<Affine semigroup in 1 dimensional space, with 2 generators>
gap> IsAffineSemigroup(as);
true
```

11.1.8 BelongsToAffineSemigroup

▷ `BelongsToAffineSemigroup(v, a)` (function)
▷ `\in(v, a)` (operation)

v is a list of nonnegative integers and `a` an affine semigroup. Returns true if the vector is in the semigroup, and false otherwise.

If the semigroup is full and its equations are known (either because the semigroup was defined by equations, or because the user has called `IsFullAffineSemgiroup(a)` and the output was true), then membership is performed by evaluating *v* in the equations. The same holds for normal semigroups and its defining inequalities.

v in a can be used for short.

Example
```
gap> a:=AffineSemigroup([[2,0],[0,2],[1,1]]);;
gap> BelongsToAffineSemigroup([5,5],a);
true
gap> BelongsToAffineSemigroup([1,2],a);
false
gap> [5,5] in a;
true
gap> [1,2] in a;
false
```

11.1.9 IsFull

▷ IsFull(S) (property)
▷ IsFullAffineSemigroup(S) (property)

`S` is an affine semigroup.

Returns true if the semigroup is full, false otherwise. The semigroup is full if whenever $a,b \in S$ and $b-a \in \mathbb{N}^k$, then $a-b \in S$, where k is the dimension of S.

If the semigroup is full, then its equations are stored in the semigroup for further use.

Example
```
gap> a:=AffineSemigroup("equations",[[[1,1,1],[0,0,2]],[2,2]]);;
gap> IsFullAffineSemigroup(a);
true
```

11.1.10 HilbertBasisOfSystemOfHomogeneousEquations

▷ HilbertBasisOfSystemOfHomogeneousEquations(ls, m) (operation)

`ls` is a list of lists of integers and `m` a list of integers. The elements of `ls` represent the rows of a matrix A. The output is a minimal generating system (Hilbert basis) of the set of nonnegative integer solutions of the system $Ax=0$ where the k first equations are in the congruences modulo `m[i]`, with k the length of `m`.

If the package `NormalizInterface` has not been loaded, then Contejean-Devie algorithm is used [CD94] instead (if this is the case, congruences are treated as in [RGS98]).

Example
```
gap> HilbertBasisOfSystemOfHomogeneousEquations([[1,0,1],[0,1,-1]],[2]);
[ [ 0, 2, 2 ], [ 1, 1, 1 ], [ 2, 0, 0 ] ]
```

If C is a pointed cone (a cone in $\mathbb{Q}^k$ not containing lines and $0 \in C$), then $S = C \cap \mathbb{N}^k$ is an affine semigroup (known as normal affine semigroup). So another way to give an affine semigroup is by a set of homogeneous inequalities, and we can represent these inequalities by its coefficients. If we put them in a matrix S can be defined as the set of nonnegative integer solutions to $Ax \geq 0$.

11.1.11 HilbertBasisOfSystemOfHomogeneousInequalities

▷ HilbertBasisOfSystemOfHomogeneousInequalities(ls) (operation) `ls` is a list of lists of integers. The elements of `ls` represent the rows of a matrix A. The output is a minimal generating system (Hilbert basis) of the set of nonnegative integer solutions to $Ax \geq 0$.

If the package `NormalizInterface` has not been loaded, then Contejean-Devie algorithm is used [CD94] instead (the use of slack variables is described in [RGSB02]).

Example
```
gap> HilbertBasisOfSystemOfHomogeneousInequalities([[2,-3],[0,1]]);
[ [ 1, 0 ], [ 2, 1 ], [ 3, 2 ] ]
```

11.1.12 EquationsOfGroupGeneratedBy

▷ `EquationsOfGroupGeneratedBy(M)` (function)

`M` is a matrix of integers. The output is a pair $[A,m]$ that represents the set of defining equations of the group spanned by the rows of `M`: $Ax = 0 \in \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_t} \times \mathbb{Z}^k$, with $m = [n_1, \ldots, n_t]$.

Example
```
gap> EquationsOfGroupGeneratedBy([[1,2,0],[2,-2,2]]);
[ [ [ 0, 0, -1 ], [ -2, 1, 3 ] ], [ 2 ] ]
```

11.1.13 BasisOfGroupGivenByEquations

▷ `BasisOfGroupGivenByEquations(A, m)` (function)

`A` is a matrix of integers and `m` is a list of positive integers. The output is a basis for the group with defining equations $Ax = 0 \in \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_t} \times \mathbb{Z}^k$, with $m = [n_1, \ldots, n_t]$.

Example
```
gap> BasisOfGroupGivenByEquations([[0,0,1],[2,-1,-3]],[2]);
[ [ -1, -2, 0 ], [ -2, 2, -2 ] ]
```

11.2 Gluings of affine semigroups

Let S_1 and S_2 be two affine semigroups with the same dimension generated by A_1 and A_2, respectively. We say that the affine semigroup S generated by the union of A_1 and A_2 is a gluing of S_1 and S_2 if $G(S_1) \cap G(S_2) = d\mathbb{Z}$ ($G(\cdot)$ stands for group spanned by) for some $d \in S_1 \cap S_2$.

The algorithm used is explained in [RGS99b].

11.2.1 GluingOfAffineSemigroups

▷ `GluingOfAffineSemigroups(a1, a2)` (function)

`a1, a2` are affine semigroups. Determines if they can be glued, and if so, returns the gluing. Otherwise it returns fail.

Example
```
gap> a1:=AffineSemigroup([[2,0],[0,2]]);
<Affine semigroup in 2 dimensional space, with 2 generators>
gap> a2:=AffineSemigroup([[1,1]]);
<Affine semigroup in 2 dimensional space, with 1 generators>
gap> GluingOfAffineSemigroups(a1,a2);
<Affine semigroup in 2 dimensional space, with 3 generators>
```

```
gap> Generators(last);
[ [ 0, 2 ], [ 1, 1 ], [ 2, 0 ] ]
```

11.3 Presentations of affine semigroups

A *minimal presentation* of an affine semigroup is defined analogously as for numerical semigroups (See Chapter 9). We warn the user to take into account that generators are stored in a set, and thus might be arranged in a different way to the initial input.

11.3.1 GeneratorsOfKernelCongruence

▷ `GeneratorsOfKernelCongruence(M)` (operation)

M is matrix with nonnegative integer coefficients. The output is a system of generators of the congruence $\{(x,y) \mid xM = yM\}$.

The main difference with `MinimalPresentationOfAffineSemigroup` (11.3.4) is that the matrix *M* can have repeated columns and these are not treated as a set.

Example
```
gap> M := [[2,0],[0,2],[1,1]];
[ [ 2, 0 ], [ 0, 2 ], [ 1, 1 ] ]
gap> GeneratorsOfKernelCongruence(M);
[ [ [ 1, 1, 0 ], [ 0, 0, 2 ] ] ]
```

11.3.2 CanonicalBasisOfKernelCongruence

▷ `CanonicalBasisOfKernelCongruence(M, Ord)` (operation)

M is matrix with nonnegative integer coefficients, *Ord* a term ordering. The output is a canonical basis of the congruence $\{(x,y) \mid xM = yM\}$ (see [RGS99a]). This corresponds with the exponents of the Gröbner basis of the kernel ideal of the morphism $x_i \mapsto Y^{m_i}$, with m_i the ith row of M.

Accepted term orderings are lexicographic (`MonomialLexOrdering()`), graded lexicographic (*MonomialGrlexOrdering()*) and reversed graded lexicographic (*MonomialGrevlexOrdering()*).

Example
```
gap> M:=[[3],[5],[7]];;
gap> CanonicalBasisOfKernelCongruence(M,MonomialLexOrdering());
[ [ [ 0, 7, 0 ], [ 0, 0, 5 ] ], [ [ 1, 0, 1 ], [ 0, 2, 0 ] ],
  [ [ 1, 5, 0 ], [ 0, 0, 4 ] ], [ [ 2, 3, 0 ], [ 0, 0, 3 ] ],
  [ [ 3, 1, 0 ], [ 0, 0, 2 ] ], [ [ 4, 0, 0 ], [ 0, 1, 1 ] ] ]
gap> CanonicalBasisOfKernelCongruence(M,MonomialGrlexOrdering());
[ [ [ 0, 7, 0 ], [ 0, 0, 5 ] ], [ [ 1, 0, 1 ], [ 0, 2, 0 ] ],
  [ [ 1, 5, 0 ], [ 0, 0, 4 ] ], [ [ 2, 3, 0 ], [ 0, 0, 3 ] ],
  [ [ 3, 1, 0 ], [ 0, 0, 2 ] ], [ [ 4, 0, 0 ], [ 0, 1, 1 ] ] ]
gap> CanonicalBasisOfKernelCongruence(M,MonomialGrevlexOrdering());
[ [ [ 0, 2, 0 ], [ 1, 0, 1 ] ], [ [ 3, 1, 0 ], [ 0, 0, 2 ] ],
  [ [ 4, 0, 0 ], [ 0, 1, 1 ] ] ]
```

11.3.3 GraverBasis

▷ `GraverBasis(M)` (operation)

`M` is matrix with integer coefficients. The output is a Graver basis for `M`.

Example
```
gap> gr:=GraverBasis([[3,5,7]]);
[ [ -7, 0, 3 ], [ -5, 3, 0 ], [ -4, 1, 1 ], [ -3, -1, 2 ], [ -2, -3, 3 ],
  [ -1, -5, 4 ], [ -1, 2, -1 ], [ 0, -7, 5 ], [ 0, 7, -5 ], [ 1, -2, 1 ],
  [ 1, 5, -4 ], [ 2, 3, -3 ], [ 3, 1, -2 ], [ 4, -1, -1 ], [ 5, -3, 0 ],
  [ 7, 0, -3 ] ]
```

11.3.4 MinimalPresentationOfAffineSemigroup

▷ `MinimalPresentationOfAffineSemigroup(a)` (operation)
▷ `MinimalPresentation(a)` (operation)

`a` is an affine semigroup. The output is a minimal presentation for `a`.

There are four methods implemented for this function, depending on the packages loaded. All of them use elimination, and Herzog's correspondence, computing the kernel of a ring homomorphism ([Her70]). The fastest procedure is achieved when `SingularInterface` is loaded, followed by `Singular`. The procedure that does not use external packages uses internal GAP Gröbner basis computations and thus it is slower. Also in this case, from the Gröbner basis, a minimal set of generating binomials must be refined, and for this Rclasses are used (if `NormalizInterface` is loaded, then the factorizations are faster). The `4ti2` implementation uses `4ti2` internal Gröbner bases and factorizations are done via `zsolve`.

Example
```
gap> a:=AffineSemigroup([2,0],[0,2],[1,1]);;
gap> MinimalPresentationOfAffineSemigroup(a);
[ [ [ 1, 0, 1 ], [ 0, 2, 0 ] ] ]
gap> GeneratorsOfAffineSemigroup(a);
[ [ 0, 2 ], [ 1, 1 ], [ 2, 0 ] ]
```

11.3.5 BettiElementsOfAffineSemigroup

▷ `BettiElementsOfAffineSemigroup(a)` (operation)
▷ `BettiElements(a)` (operation)

`a` is an affine semigroup. The output is the set of Betti elements of `a` (defined as for numerical semigroups).

This function relies on the computation of a minimal presentation.

Example
```
gap> a:=AffineSemigroup([2,0],[0,2],[1,1]);;
gap> BettiElementsOfAffineSemigroup(a);
[ [ 2, 2 ] ]
```

11.3.6 ShadedSetOfElementInAffineSemigroup

▷ ShadedSetOfElementInAffineSemigroup(*v*, a) (function)

a is an affine semigroup and *v* is an element in a. This is a translation to affine semigroups of ShadedSetOfElementInNumericalSemigroup (4.1.5).

11.3.7 IsGeneric (for affine semigroups)

▷ IsGeneric(a) (property)
▷ IsGenericAffineSemigroup(a) (property)

a is an affine semigroup.
The same as IsGenericNumericalSemigroup (4.2.2) but for affine semigroups.
This property implies IsUniquelyPresentedAffineSemigroup (11.3.8).

11.3.8 IsUniquelyPresentedAffineSemigroup

▷ IsUniquelyPresentedAffineSemigroup(a) (property)

a is an affine semigroup.
The same as IsUniquelyPresentedNumericalSemigroup (4.2.1) but for affine semigroups.

11.3.9 DegreesOfPrimitiveElementsOfAffineSemigroup

▷ DegreesOfPrimitiveElementsOfAffineSemigroup(a) (operation)

a is an affine semigroup. The output is the set of primitive elements of a (defined as for numerical semigroups).

This function has three implementations (methods), one using Graver basis via the Lawrence lifting of a and the other (much faster) using NormalizInterface. Also a 4ti2 version using its Graver basis computation is provided.

Example
```
gap> a:=AffineSemigroup([2,0],[0,2],[1,1]);;
gap> DegreesOfPrimitiveElementsOfAffineSemigroup(a);
[ [ 0, 2 ], [ 1, 1 ], [ 2, 0 ], [ 2, 2 ] ]
```

11.4 Factorizations in affine semigroups

The invariants presented here are defined as for numerical semigroups (Chapter 9).

As with presentations, the user should take into account that generators are stored in a set, and thus might be arranged in a different way to the initial input.

11.4.1 FactorizationsVectorWRTList

▷ FactorizationsVectorWRTList(*v*, *ls*) (operation) `v` is a list of nonnegative integers and `ls` is a list of lists of nonnegative integers. The output is set of factorizations of `v` in terms of the elements of `ls`.

If no extra package is loaded, then factorizations are computed recursively; and thus slowly. If `NormalizInterface` is loaded, then a system of equations is solved with Normaliz, and the performance is much better. If `4ti2Interface` is loaded instead, then factorizations are calculated using `zsolve` command of `4ti2`.

Example
```
gap> FactorizationsVectorWRTList([5,5],[[2,0],[0,2],[1,1]]);
[ [ 2, 2, 1 ], [ 1, 1, 3 ], [ 0, 0, 5 ] ]
```

11.4.2 ElasticityOfAffineSemigroup

▷ `ElasticityOfAffineSemigroup(a)` (operation)

`a` is an affine semigroup. The output is the elasticity of `a` (defined as for numerical semigroups).

The procedure used is based on [Phi10], where it is shown that the elasticity can be computed by using circuits. The set of circuits is calculated using [ES96].

Example
```
gap> a:=AffineSemigroup([2,0],[0,2],[1,1]);;
gap> ElasticityOfAffineSemigroup(a);
1
```

11.4.3 DeltaSetOfAffineSemigroup

▷ `DeltaSetOfAffineSemigroup(a)` (function)

`a` is an affine semigroup. The output is the Delta set of `a` (defined as for numerical semigroups). The the procedure used is explained in [GSOW17].

Example
```
gap> a:=AffineSemigroup([2,0],[0,2],[1,1]);;
gap> DeltaSetOfAffineSemigroup(a);
[  ]
gap> s:=NumericalSemigroup(10,13,15,47);;
gap> a:=AsAffineSemigroup(s);;
gap> DeltaSetOfAffineSemigroup(a);
[ 1, 2, 3, 5 ]
```

11.4.4 CatenaryDegreeOfAffineSemigroup

▷ `CatenaryDegreeOfAffineSemigroup(a)` (function)
▷ `CatenaryDegree(a)` (operation)

`a` is an affine semigroup. The output is the catenary degree of `a` (defined as for numerical semigroups).

Example
```
gap> a:=AffineSemigroup([2,0],[0,2],[1,1]);;
gap> CatenaryDegreeOfAffineSemigroup(a);
2
```

11.4.5 EqualCatenaryDegreeOfAffineSemigroup

▷ EqualCatenaryDegreeOfAffineSemigroup(a) (function)

a is an affine semigroup. The output is the equal catenary degree of a (defined as for numerical semigroups).

This function relies on the results presented in [GSOSRN13].

11.4.6 HomogeneousCatenaryDegreeOfAffineSemigroup

▷ HomogeneousCatenaryDegreeOfAffineSemigroup(a) (function)

a is an affine semigroup. The output is the homogeneous catenary degree of a (defined as for numerical semigroups).

This function is based on [GSOSRN13].

11.4.7 MonotoneCatenaryDegreeOfAffineSemigroup

▷ MonotoneCatenaryDegreeOfAffineSemigroup(a) (function)

a is an affine semigroup. The output is the monotone catenary degree of a (defined as for numerical semigroups), computed as explained in [Phi10].

Example
```
gap> a:=AffineSemigroup("inequalities",[[2,-1],[-1,3]]);
<Affine semigroup>
gap> GeneratorsOfAffineSemigroup(a);
[ [ 1, 1 ], [ 1, 2 ], [ 2, 1 ], [ 3, 1 ] ]
gap> CatenaryDegreeOfAffineSemigroup(a);
3
gap> EqualCatenaryDegreeOfAffineSemigroup(a);
2
gap> HomogeneousCatenaryDegreeOfAffineSemigroup(a);
3
gap> MonotoneCatenaryDegreeOfAffineSemigroup(a);
3
```

11.4.8 TameDegreeOfAffineSemigroup

▷ TameDegreeOfAffineSemigroup(a) (operation)

a is an affine semigroup. The output is the tame degree of a (defined as for numerical semigroups). If a is given by equations (or its equations are known), then the procedure explained in [GSOW17] is used.

Example
```
gap> a:=AffineSemigroup([2,0],[0,2],[1,1]);;
gap> TameDegreeOfAffineSemigroup(a);
2
```

11.4.9 OmegaPrimalityOfElementInAffineSemigroup

▷ `OmegaPrimalityOfElementInAffineSemigroup(v, a)` (operation)

`v` is a list of nonnegative integers and `a` is an affine semigroup. The output is the omega primality of `a` (defined as for numerical semigroups). Returns 0 if the element is not in the semigroup.

The implementation of this procedure is performed as explained in [BGSG11] (also, if the semigroup has defining equations, then it takes advantage of this fact as explained in this reference).

Example
```
gap> a:=AffineSemigroup([2,0],[0,2],[1,1]);;
gap> OmegaPrimalityOfElementInAffineSemigroup([5,5],a);
6
```

11.4.10 OmegaPrimalityOfAffineSemigroup

▷ `OmegaPrimalityOfAffineSemigroup(a)` (function)

`a` is an affine semigroup. The output is the omega primality of `a` (defined as for numerical semigroups).

Example
```
gap> a:=AffineSemigroup([2,0],[0,2],[1,1]);;
gap> OmegaPrimalityOfAffineSemigroup(a);
2
```

Chapter 12

Good semigroups

We will only cover here good semigroups of $\mathbb{N}^2$.

A *good semigroup* S is a submonoid of $\mathbb{N}^2$, with the following properties.

(G1) It is closed under infimums (minimum componentwise).

(G2) If $a,b \in M$ and $a_i = b_i$ for some $i \in \{1,2\}$, then there exists $c \in M$ such that $c_i > a_i = b_i$ and $c_j = \min\{a_j, b_j\}$, with $j \in \{1,2\} \setminus \{i\}$.

(G3) There exists $C \in \mathbb{N}^n$ such that $C + \mathbb{N}^n \subseteq S$.

Value semigroups of algebroid branches are good semigroups, but there are good semigroups that are not of this form. Since good semigroups are closed under infimums, if C_1 and C_2 fulfill $C_i + \mathbb{N}^n \subseteq S$, then $C_1 \wedge C_2 + \mathbb{N}^n \subseteq S$. So there is a minimum C fulfilling $C + \mathbb{N}^n \subseteq S$, which is called the *conductor* of S.

The contents of this chapter are described in [DGSM16].

12.1 Defining good semigroups

Good semigroups can be constructed with numerical duplications, amalgamations, cartesian products, or by giving some of its generators and a candidate for conductor. Not every set determines a good semigroup; this is because the intersection of good semigroups might not be a good semigroup. So the terminology "good semigroup generated" by a set is a bit fragile.

12.1.1 IsGoodSemigroup

▷ `IsGoodSemigroup(S)` (function)

Detects if `S` is an object of type good semigroup.

12.1.2 NumericalSemigroupDuplication

▷ `NumericalSemigroupDuplication(S, E)` (function)

`S` is a numerical semigroup and `E` is an ideal of `S` with $E \subseteq S$. The output is $S \bowtie E = D \cup (E \times E) \cup \{a \wedge b \mid a \in D, b \in E \times E\}$, where $D = \{(s,s) \mid s \in S\}$.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=6+s;;
```

```
gap> dup:=NumericalSemigroupDuplication(s,e);
<Good semigroup>
gap> l:=Cartesian([1..11],[1..11]);;
gap> Intersection(dup,l);
[ [ 3, 3 ], [ 5, 5 ], [ 6, 6 ], [ 6, 7 ], [ 6, 8 ], [ 6, 9 ], [ 6, 10 ],
  [ 6, 11 ], [ 7, 6 ], [ 7, 7 ], [ 8, 6 ], [ 8, 8 ], [ 9, 6 ], [ 9, 9 ],
  [ 9, 10 ], [ 9, 11 ], [ 10, 6 ], [ 10, 9 ], [ 10, 10 ], [ 11, 6 ],
  [ 11, 9 ], [ 11, 11 ] ]
gap> [384938749837,349823749827] in dup;
true
```

12.1.3 AmalgamationOfNumericalSemigroups

▷ AmalgamationOfNumericalSemigroups(*S*, *E*, *b*) (function)

S is a numerical semigroup, E is an ideal of a numerical semigroup T with $E \subseteq T$, and b is an integer such that multiplication by b is a morphism from S to T, say g. The output is $S \bowtie^g E = D \cup (g^{-1}(E) \times E) \cup \{a \wedge b \mid a \in D, b \in g^{-1}(E) \times E\}$, where $D = \{(s, bs) \mid s \in S\}$.

Example

```
gap> s:=NumericalSemigroup(2,3);;
gap> t:=NumericalSemigroup(3,4);;
gap> e:=3+t;;
gap> dup:=AmalgamationOfNumericalSemigroups(s,e,2);;
gap> [2,3] in dup;
true
```

12.1.4 CartesianProductOfNumericalSemigroups

▷ CartesianProductOfNumericalSemigroups(*S*, *T*) (function)

S and T are numerical semigroups. The output is $S \times T$, which is a good semigroup.

Example

```
gap> s:=NumericalSemigroup(2,3);;
gap> t:=NumericalSemigroup(3,4);;
gap> IsGoodSemigroup(CartesianProductOfNumericalSemigroups(s,t));
true
```

12.1.5 GoodSemigroup

▷ GoodSemigroup(*X*, *C*) (function)

X is a list of points with nonnegative integer coordinates and C is a pair of nonnegative integers (a list with two elements). If M is the affine and infimum closure of X, decides if it is a good semigroup, and if so, outputs it.

Example

```
gap> G:=[[4,3],[7,13],[11,17],[14,27],[15,27],[16,20],[25,12],[25,16]];
[ [ 4, 3 ], [ 7, 13 ], [ 11, 17 ], [ 14, 27 ], [ 15, 27 ], [ 16, 20 ],
  [ 25, 12 ], [ 25, 16 ] ]
gap> C:=[25,27];
```

```
[ 25, 27 ]
gap> GoodSemigroup(G,C);
<Good semigroup>
```

12.2 Notable elements

Good semigroups are a natural extension of numerical semigroups, and so some of their notable elements are called in the same way as in the one dimensional case.

12.2.1 BelongsToGoodSemigroup

▷ BelongsToGoodSemigroup(*v*, *S*) (operation)
▷ \in(*v*, *S*) (operation)

S is a good semigroup and *v* is a pair of integers. The output is `true` if *v* is in *S*, and `false` otherwise. Other ways to use this operation are `\in(v,S)` and `v in S`.

Example

```
gap> s:=NumericalSemigroup(2,3);;
gap> e:=6+s;;
gap> dup:=NumericalSemigroupDuplication(s,e);;
gap> BelongsToGoodSemigroup([2,2],dup);
true
gap> [2,2] in dup;
true
gap> [3,2] in dup;
false
```

12.2.2 Conductor (for good semigroup)

▷ Conductor(*S*) (function)
▷ ConductorOfGoodSemigroup(*S*) (function)

S is a good semigroup. The output is its conductor.

Example

```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=6+s;;
gap> dup:=NumericalSemigroupDuplication(s,e);
<Good semigroup>
gap> Conductor(dup);
[ 11, 11 ]
gap> ConductorOfGoodSemigroup(dup);
[ 11, 11 ]
```

12.2.3 SmallElements (for good semigroup)

▷ SmallElements(*S*) (function)
▷ SmallElementsOfGoodSemigroup(*S*) (function)

S is a good semigroup. The output is its set of small elements, that is, the elements smaller than its conductor with respect to the usual partial ordering.

Example

```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=6+s;;
gap> dup:=NumericalSemigroupDuplication(s,e);
<Good semigroup>
gap> SmallElementsOfGoodSemigroup(dup);
[ [ 0, 0 ], [ 3, 3 ], [ 5, 5 ], [ 6, 6 ], [ 6, 7 ], [ 6, 8 ], [ 6, 9 ],
  [ 6, 10 ], [ 6, 11 ], [ 7, 6 ], [ 7, 7 ], [ 8, 6 ], [ 8, 8 ], [ 9, 6 ],
  [ 9, 9 ], [ 9, 10 ], [ 9, 11 ], [ 10, 6 ], [ 10, 9 ], [ 10, 10 ],
  [ 11, 6 ], [ 11, 9 ], [ 11, 11 ] ]
```

12.2.4 RepresentsSmallElementsOfGoodSemigroup

▷ RepresentsSmallElementsOfGoodSemigroup(X) (function)

X is a list of points in the nonnegative orthant of the plane with integer coordinates. Determines if it represents the set of small elements of a good semigroup.

Example

```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=6+s;;
gap> dup:=NumericalSemigroupDuplication(s,e);
<Good semigroup>
gap> SmallElementsOfGoodSemigroup(dup);
[ [ 0, 0 ], [ 3, 3 ], [ 5, 5 ], [ 6, 6 ], [ 6, 7 ], [ 6, 8 ], [ 6, 9 ], [ 6, 10 ],
  [ 6, 11 ], [ 7, 6 ], [ 7, 7 ], [ 8, 6 ], [ 8, 8 ], [ 9, 6 ], [ 9, 9 ], [ 9, 10 ],
  [ 9, 11 ], [ 10, 6 ], [ 10, 9 ], [ 10, 10 ], [ 11, 6 ], [ 11, 9 ], [ 11, 11 ] ]
gap> RepresentsSmallElementsOfGoodSemigroup(last);
true
```

12.2.5 GoodSemigroupBySmallElements

▷ GoodSemigroupBySmallElements(X) (function)

X is a list of points in the nonnegative orthant of the plane with integer coordinates. Determines if it represents the set of small elements of a good semigroup, and then outputs the good semigroup having X as set of small elements.

Example

```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=6+s;;
gap> dup:=NumericalSemigroupDuplication(s,e);
<Good semigroup>
gap> SmallElementsOfGoodSemigroup(dup);
[ [ 0, 0 ], [ 3, 3 ], [ 5, 5 ], [ 6, 6 ], [ 6, 7 ], [ 6, 8 ], [ 6, 9 ], [ 6, 10 ],
  [ 6, 11 ], [ 7, 6 ], [ 7, 7 ], [ 8, 6 ], [ 8, 8 ], [ 9, 6 ], [ 9, 9 ], [ 9, 10 ],
  [ 9, 11 ], [ 10, 6 ], [ 10, 9 ], [ 10, 10 ], [ 11, 6 ], [ 11, 9 ], [ 11, 11 ] ]
gap> G:=GoodSemigroupBySmallElements(last);
<Good semigroup>
gap> dup=G;
true
```

12.2.6 MaximalElementsOfGoodSemigroup

▷ MaximalElementsOfGoodSemigroup(S) (attribute)

S is a good semigroup. The output is the set of elements (x,y) of S with the following property: there is no other element (x',y') in S with $(x,y) \leq (x',y')$ sharing a coordinate with (x,y).

Example

```
gap> G:=[[4,3],[7,13],[11,17]];;
gap> g:=GoodSemigroup(G,[11,17]);;
gap> mx:=MaximalElementsOfGoodSemigroup(g);
[ [ 0, 0 ], [ 4, 3 ], [ 7, 13 ], [ 8, 6 ] ]
```

12.2.7 IrreducibleMaximalElementsOfGoodSemigroup

▷ IrreducibleMaximalElementsOfGoodSemigroup(S) (attribute)

S is a good semigroup. The output is the set of elements nonzero maximal elements that cannot be expressed as a sum of two nonzero maximal elements of the good semigroup.

Example

```
gap> G:=[[4,3],[7,13],[11,17]];;
gap> g:=GoodSemigroup(G,[11,17]);;
gap> IrreducibleMaximalElementsOfGoodSemigroup(g);
[ [ 4, 3 ], [ 7, 13 ] ]
```

12.2.8 GoodSemigroupByMaximalElements

▷ GoodSemigroupByMaximalElements(S, T, M, C) (function)

S and T are numerical semigroups, M is a list of pairs in $S \times T$. C is the conductor, and thus a pair of nonnegative integers. The output is the set of elements of $S \times T$ that are not above an element in M, that is, if they share a coordinate with an element in M, then they must be smaller or equal to that element with respect to the usual partial ordering. The output is a good semigroup, if M is an correct set of maximal elements.

Example

```
gap> G:=[[4,3],[7,13],[11,17]];;
gap> g:=GoodSemigroup(G,[11,17]);;
gap> sm:=SmallElements(g);;
gap> mx:=MaximalElementsOfGoodSemigroup(g);;
gap> s:=NumericalSemigroupBySmallElements(Set(sm,x->x[1]));;
gap> t:=NumericalSemigroupBySmallElements(Set(sm,x->x[2]));;
gap> Conductor(g);
[ 11, 15 ]
gap> gg:=GoodSemigroupByMaximalElements(s,t,mx,[11,15]);
<Good semigroup>
gap> gg=g;
true
```

12.2.9 MinimalGoodGeneratingSystemOfGoodSemigroup

▷ MinimalGoodGeneratingSystemOfGoodSemigroup(*S*) (function)
▷ MinimalGoodGenerators(*S*) (attribute)

S is a good semigroup. The output is its minimal good generating system (which is unique in the local case).

MinimalGoodGeneratingSystemOfGoodSemigroup and MinimalGoodGenerators are synonyms.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=6+s;;
gap> dup:=NumericalSemigroupDuplication(s,e);
<Good semigroup>
gap> MinimalGoodGeneratingSystemOfGoodSemigroup(dup);
[ [ 3, 3 ], [ 5, 5 ], [ 6, 11 ], [ 7, 7 ], [ 11, 6 ] ]
gap> MinimalGoodGenerators(dup);
[ [ 3, 3 ], [ 5, 5 ], [ 6, 11 ], [ 7, 7 ], [ 11, 6 ] ]
```

12.3 Symmetric good semigroups

The concept of symmetry in a numerical semigroup extends to good semigroups. Here we describe a test for symmetry.

12.3.1 IsSymmetricGoodSemigroup

▷ IsSymmetricGoodSemigroup(*S*) (attribute)
▷ IsSymmetric(*S*) (attribute)

S is a good semigroup. Determines if *S* is a symmetric good semigroup.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=CanonicalIdealOfNumericalSemigroup(s);;
gap> e:=15+e;;
gap> dup:=NumericalSemigroupDuplication(s,e);;
gap> IsSymmetricGoodSemigroup(dup);
true
```

12.4 Arf good closure

The definition of Arf good semigroup is similar to the definition of Arf numerical semigroup. In this section, we provide a function to compute the Arf good closure of a good semigroup.

12.4.1 ArfGoodSemigroupClosure

▷ ArfGoodSemigroupClosure(*S*) (function)
▷ ArfClosure(*S*) (operation)

S is a good semigroup. Determines the Arf good semigroup closure of S.

Example
```
gap> G:=[[3,3],[4,4],[5,4],[4,6]];
[ [ 3, 3 ], [ 4, 4 ], [ 5, 4 ], [ 4, 6 ] ]
gap> C:=[6,6];
[ 6, 6 ]
gap> S:=GoodSemigroup(G,C);
<Good semigroup>
gap> SmallElements(S);
[ [ 0, 0 ], [ 3, 3 ], [ 4, 4 ], [ 4, 6 ], [ 5, 4 ], [ 6, 6 ] ]
gap> A:=ArfGoodSemigroupClosure(S);
<Good semigroup>
gap> SmallElements(A);
[ [ 0, 0 ], [ 3, 3 ], [ 4, 4 ] ]
```

12.5 Good ideals

A relative ideal I of a relative good semigroup M is a relative good ideal if I fulfills conditions (G1) and (G2) of the definition of good semigroup.

12.5.1 GoodIdeal

▷ `GoodIdeal(X, S)` (function)

X is a list of points with nonnegative integer coordinates and S is good semigroup. Decides if the closure of $X+S$ under infimums is a relative good ideal of S, and if so, outputs it.

Example
```
gap> G:=[[4,3],[7,13],[11,17],[14,27],[15,27],[16,20],[25,12],[25,16]];
[ [ 4, 3 ], [ 7, 13 ], [ 11, 17 ], [ 14, 27 ], [ 15, 27 ], [ 16, 20 ],
[ 25, 12 ], [ 25, 16 ] ]
gap> C:=[25,27];
[ 25, 27 ]
gap> g := GoodSemigroup(G,C);
<Good semigroup>
gap> i:=GoodIdeal([[2,3]],g);
<Good ideal of good semigroup>
```

12.5.2 GoodGeneratingSystemOfGoodIdeal

▷ `GoodGeneratingSystemOfGoodIdeal(I)` (function)

I is a good ideal of a good semigroup. The output is a good generating system of I.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=10+s;;
gap> d:=NumericalSemigroupDuplication(s,e);;
gap> e:=GoodIdeal([[2,3],[3,2],[2,2]],d);;
gap> GoodGeneratingSystemOfGoodIdeal(e);
[ [ 2, 2 ], [ 2, 3 ], [ 3, 2 ] ]
```

12.5.3 AmbientGoodSemigroupOfGoodIdeal

▷ AmbientGoodSemigroupOfGoodIdeal(*I*) (function)

If *I* is a good ideal of a good semigroup M, then the output is M. The output is a good generating system of *I*.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=10+s;;
gap> a:=AmalgamationOfNumericalSemigroups(s,e,5);;
gap> e:=GoodIdeal([[2,3],[3,2],[2,2]],a);;
gap> a=AmbientGoodSemigroupOfGoodIdeal(e);
true
```

12.5.4 MinimalGoodGeneratingSystemOfGoodIdeal

▷ MinimalGoodGeneratingSystemOfGoodIdeal(*I*) (function)

I is a good ideal of a good semigroup. The output is the minimal good generating system of *I*.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=10+s;;
gap> d:=NumericalSemigroupDuplication(s,e);;
gap> e:=GoodIdeal([[2,3],[3,2],[2,2]],d);;
gap> MinimalGoodGeneratingSystemOfGoodIdeal(e);
[ [ 2, 3 ], [ 3, 2 ] ]
```

12.5.5 BelongsToGoodIdeal

▷ BelongsToGoodIdeal(*v*, *I*) (operation)
▷ \in(*v*, *I*) (operation)

I is a good ideal of a good semigroup and *v* is a pair of integers. The output is `true` if *v* is in *I*, and `false` otherwise. Other ways to use this operation are `\in(v,I)` and `v in I`.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=10+s;;
gap> d:=NumericalSemigroupDuplication(s,e);;
gap> e:=GoodIdeal([[2,3],[3,2]],d);;
gap> [1,1] in e;
false
gap> [2,2] in e;
true
```

12.5.6 SmallElementsOfGoodIdeal

▷ SmallElementsOfGoodIdeal(*I*) (function)
▷ SmallElements(*I*) (function)

I is a good ideal. The output is its set of small elements, that is, the elements smaller than its conductor and larger than its minimum element (with respect to the usual partial ordering).

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=10+s;;
gap> d:=NumericalSemigroupDuplication(s,e);;
gap> e:=GoodIdeal([[2,3],[3,2]],d);;
gap> SmallElements(e);
[ [ 2, 2 ], [ 2, 3 ], [ 3, 2 ], [ 5, 5 ], [ 5, 6 ], [ 6, 5 ], [ 7, 7 ] ]
```

12.5.7 CanonicalIdealOfGoodSemigroup

▷ CanonicalIdealOfGoodSemigroup(*S*) (function)

S is a good semigroup. The output is the canonical ideal of *S*.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> e:=10+s;;
gap> d:=NumericalSemigroupDuplication(s,e);;
gap> c:=CanonicalIdealOfGoodSemigroup(d);;
gap> MinimalGoodGeneratingSystemOfGoodIdeal(c);
[ [ 0, 0 ], [ 2, 2 ] ]
```

Chapter 13

External packages

The use of the packages NormalizInterface [GHS14] (an interface to Normalize [BIRC14]; or in its absence 4ti2Interface[Gut], an interface to 4ti2[tt]), SingularInterface (an interface to Singular [DGPS12]; or in its absence Singular [CdG12]); or in its absence GradedModules [BGJ+14] is highly recommended for many of the functions presented in this chapter. However, whenever possible a method not depending on these packages is also provided (though slower). The package tests if the user has downloaded any of the above packages, and if so puts NumSgpsCanUsePackage to true, where Package is any of the above.

13.1 Using external packages

As mentioned above some methods are specifically implemented to take advantage of several external packages. The following functions can be used in case these packages have not been loaded prior to numericalsgps.

13.1.1 NumSgpsUse4ti2

▷ NumSgpsUse4ti2() (function)

Tries to load the package 4ti2Interface. If the package is available, then it also loads methods implemented using functions in this package.

13.1.2 NumSgpsUse4ti2gap

▷ NumSgpsUse4ti2gap() (function)

Tries to load the package 4ti2gap. If the package is available, then it also loads methods implemented using functions in this package.

13.1.3 NumSgpsUseNormalize

▷ NumSgpsUseNormalize() (function)

Tries to load the package NormalizInterface. If the package is available, then it also loads methods implemented using functions in this package.

13.1.4 NumSgpsUseSingular

▷ `NumSgpsUseSingular()` (function)

Tries to load the package `singular`. If the package is available, then it also loads methods implemented using functions in this package.

To prevent incompatibilities, the package will not load if `SingularInterface` has been already loaded.

13.1.5 NumSgpsUseSingularInterface

▷ `NumSgpsUseSingularInterface()` (function)

Tries to load the package `SingularInterface`. If the package is available, then it also loads methods implemented using functions in this package.

To prevent incompatibilities, the package will not load if `singular` has been already loaded.

13.1.6 NumSgpsUseSingularGradedModules

▷ `NumSgpsUseSingularGradedModules()` (function)

Tries to load the package `GradedModules`. If the package is available, then it also loads methods implemented using functions in this package.

It also creates a ring of rationals `NumSgpsRationals`.

Chapter 14

Dot functions

14.1 Dot functions

We provide several functions to translate graphs, Hasse diagrams or trees related to numerical and affine semigroups to the dot language. This can either be used with graphviz or any javascript library that interprets dot language. We give the alternative to use `DotSplash` that uses viz.js.

14.1.1 DotBinaryRelation

▷ `DotBinaryRelation(`*br*`)` (function)

`br` is a binary relation. Returns a GraphViz dot that represents the binary relation `br`. The set of vertices of the resulting graph is the source of `br`. Edges join those elements which are related with respect to `br`.

Example

```
gap> br:=BinaryRelationByElements(Domain([1,2]), [Tuple([1,2])]);
<general mapping: <object> -> <object> >
gap> Print(DotBinaryRelation(br));
digraph  NSGraph{rankdir = TB; edge[dir=back];
1 [label="1"];
2 [label="2"];
2 -> 1;
}
```

14.1.2 HasseDiagramOfNumericalSemigroup

▷ `HasseDiagramOfNumericalSemigroup(`*S*, *A*`)` (function)

`S` is a numerical semigroup and `A` is a set of integers. Returns a binary relation which is the Hasse diagram of `A` with respect to the ordering $a \preceq b$ if $b-a$ in `S`.

Example

```
gap> s:=NumericalSemigroup(3,5,7);;
gap> HasseDiagramOfNumericalSemigroup(s,[1,2,3]);
<general mapping: <object> -> <object> >
```

14.1.3 HasseDiagramOfBettiElementsOfNumericalSemigroup

▷ HasseDiagramOfBettiElementsOfNumericalSemigroup(*S*) (function)

S is a numerical semigroup. Applies HasseDiagramOfBettiElementsOfNumericalSemigroup with arguments S and its Betti elements.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> HasseDiagramOfBettiElementsOfNumericalSemigroup(s);
<general mapping: <object> -> <object> >
```

14.1.4 HasseDiagramOfAperyListOfNumericalSemigroup

▷ HasseDiagramOfAperyListOfNumericalSemigroup(*S*[, *n*]) (function)

S is a numerical semigroup, n is an integer (optional, if not provided, the multiplicity of the semigroup is taken as its value). Applies HasseDiagramOfBettiElementsOfNumericalSemigroup (14.1.3) with arguments S and the Apéry set of S with respect to n.

Example
```
gap> s:=NumericalSemigroup(3,5,7);;
gap> HasseDiagramOfAperyListOfNumericalSemigroup(s);
<general mapping: <object> -> <object> >
gap> HasseDiagramOfAperyListOfNumericalSemigroup(s,10);
<general mapping: <object> -> <object> >
```

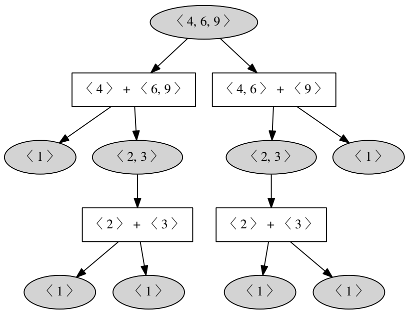
14.1.5 DotTreeOfGluingsOfNumericalSemigroup

▷ DotTreeOfGluingsOfNumericalSemigroup(*S*) (function)

S is a numerical semigroup. It outputs a tree (in dot) representing the many ways S can be decomposed as a gluing of numerical semigroups (and goes recursively in the factors).

Example
```
gap> s:=NumericalSemigroup(4,6,9);;
gap> Print(DotOverSemigroupsNumericalSemigroup(s));
digraph  NSGraph{rankdir = TB;
0 [label=" 4, 6, 9 "];
0 [label=" 4, 6, 9 ", style=filled];
1 [label=" 4  +  6, 9 " , shape=box];
2 [label=" 1 ", style=filled];
3 [label=" 2, 3 ", style=filled];
4 [label=" 2  +  3 " , shape=box];
5 [label=" 1 ", style=filled];
6 [label=" 1 ", style=filled];
7 [label=" 4, 6  +  9 " , shape=box];
8 [label=" 2, 3 ", style=filled];
10 [label=" 2  +  3 " , shape=box];
11 [label=" 1 ", style=filled];
12 [label=" 1 ", style=filled];
9 [label=" 1 ", style=filled];
0 -> 1;
```

```
1 -> 2;
1 -> 3;
3 -> 4;
4 -> 5;
4 -> 6;
0 -> 7;
7 -> 8;
7 -> 9;
8 -> 10;
10 -> 11;
10 -> 12;
}
```



14.1.6 DotOverSemigroupsNumericalSemigroup

▷ DotOverSemigroupsNumericalSemigroup(*S*) (function)

S is a numerical semigroup. It outputs the Hasse diagram (in dot) of oversemigroups of S.

Example

```
gap> s:=NumericalSemigroup(4,6,9);;
gap> Print(DotOverSemigroupsNumericalSemigroup(s));
digraph  NSGraph{rankdir = TB; edge[dir=back];
1 [label=" 1 ", style=filled];
2 [label=" 2, 3 ", style=filled];
3 [label=" 2, 5 ", style=filled];
4 [label=" 2, 7 ", style=filled];
5 [label=" 2, 9 ", style=filled];
6 [label=" 3, 4, 5 ", style=filled];
7 [label=" 3, 4 ", style=filled];
8 [label=" 4, 5, 6, 7 "];
9 [label=" 4, 5, 6 ", style=filled];
10 [label=" 4, 6, 7, 9 "];
11 [label=" 4, 6, 9, 11 "];
12 [label=" 4, 6, 9 ", style=filled];
1 -> 2;
2 -> 3;
2 -> 6;
3 -> 4;
3 -> 8;
```

```
4 -> 5;
4 -> 10;
5 -> 11;
6 -> 7;
6 -> 8;
7 -> 10;
8 -> 9;
8 -> 10;
9 -> 11;
10 -> 11;
11 -> 12;
}
```

14.1.7 DotRosalesGraph (for affine semigroup)

▷ DotRosalesGraph(*n*, *S*) (operation)
▷ DotRosalesGraph(*n*, *S*) (operation)

S is either numerical or an affine semigroup and n is an element in S. It outputs the graph associated to n in S (see GraphAssociatedToElementInNumericalSemigroup (4.1.2)).

Example

```
gap> s:=NumericalSemigroup(4,6,9);;
gap> Print(DotRosalesGraph(15,s));
graph  NSGraph{
1 [label="6"];
2 [label="9"];
2 -- 1;
}
```

14.1.8 DotFactorizationGraph

▷ DotFactorizationGraph(*f*) (operation)

f is a set of factorizations. Returns the graph (in dot) of factorizations associated to f: a complete graph whose vertices are the elements of f. Edges are labelled with distances between the nodes they join. Kruskal algorithm is used to draw in red a spanning tree with minimal distances. Thus the catenary degree is reached in the edges of the tree.

Example

```
gap> f:=FactorizationsIntegerWRTList(20,[3,5,7]);
[ [ 5, 1, 0 ], [ 0, 4, 0 ], [ 1, 2, 1 ], [ 2, 0, 2 ] ]
gap> Print(DotFactorizationGraph(f));
graph  NSGraph{
1 [label=" (5, 1, 0)"];
2 [label=" (0, 4, 0)"];
3 [label=" (1, 2, 1)"];
4 [label=" (2, 0, 2)"];
2 -- 3[label="2", color="red"];
3 -- 4[label="2", color="red"];
1 -- 3[label="4", color="red"];
1 -- 4[label="4" ];
```

```
2 -- 4[label="4" ];
1 -- 2[label="5" ];
}
```

14.1.9 DotEliahouGraph

▷ DotEliahouGraph(*f*) (operation)

f is a set of factorizations. Returns the Eliahou graph (in dot) of factorizations associated to f: a graph whose vertices are the elements of f, and there is an edge between two vertices if they have common support. Edges are labelled with distances between nodes they join.

Example

```
gap> f:=FactorizationsIntegerWRTList(20,[3,5,7]);
[ [ 5, 1, 0 ], [ 0, 4, 0 ], [ 1, 2, 1 ], [ 2, 0, 2 ] ]
gap> Print(DotEliahouGraph(f));
graph  NSGraph{
1 [label=" (5, 1, 0)"];
2 [label=" (0, 4, 0)"];
3 [label=" (1, 2, 1)"];
4 [label=" (2, 0, 2)"];
2 -- 3[label="2" ];
3 -- 4[label="2" ];
1 -- 3[label="4" ];
1 -- 4[label="4" ];
1 -- 2[label="5" ];
}
```

14.1.10 SetDotNSEngine

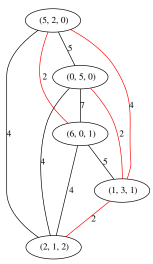
▷ SetDotNSEngine(*engine*) (function)

This function sets the value of DotNSEngine to engine, which must be any of the following "circo","dot","fdp","neato","osage","twopi". This tells viz.js which graphviz engine to use.

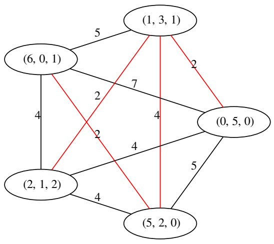
Example

```
gap> SetDotNSEngine("circo");
true
```

Here is an example with the default dot engine



And one with circo engine



14.1.11 DotSplash

▷ `DotSplash([dots])` (function)

Launches a browser and visualizes the dots diagrams provided as arguments. It outputs the html page displayed as a string, and prints the location of the temporary file that contains it.

Appendix A

Generalities

Here we describe some functions which are not specific for numerical semigroups but are used to do computations with them. As they may have interest by themselves, we describe them here.

A.1 Bézout sequences

A sequence of positive rational numbers $a_1/b_1 < \cdots < a_n/b_n$ with a_i, b_i positive integers is a *Bézout sequence* if $a_{i+1}b_i - a_i b_{i+1} = 1$ for all $i \in \{1, \ldots, n-1\}$.

The following function uses an algorithm presented in [BR09].

A.1.1 BezoutSequence

▷ `BezoutSequence(arg)` (function)

arg consists of two rational numbers or a list of two rational numbers. The output is a Bézout sequence with ends the two rational numbers given. (Warning: rational numbers are silently transformed into irreducible fractions.)

Example
```
gap> BezoutSequence(4/5,53/27);
[ 4/5, 1, 3/2, 5/3, 7/4, 9/5, 11/6, 13/7, 15/8, 17/9, 19/10, 21/11, 23/12,
  25/13, 27/14, 29/15, 31/16, 33/17, 35/18, 37/19, 39/20, 41/21, 43/22,
  45/23, 47/24, 49/25, 51/26, 53/27 ]
```

A.1.2 IsBezoutSequence

▷ `IsBezoutSequence(L)` (function)

L is a list of rational numbers. `IsBezoutSequence` returns `true` or `false` according to whether *L* is a Bézout sequence or not.

Example
```
gap> IsBezoutSequence([ 4/5, 1, 3/2, 5/3, 7/4, 9/5, 11/6]);
true
gap> IsBezoutSequence([ 4/5, 1, 3/2, 5/3, 7/4, 9/5, 11/3]);
Take the 6 and the 7 elements of the sequence
false
```

A.1.3 CeilingOfRational

▷ `CeilingOfRational(r)` (function)

Returns the smallest integer greater than or equal to the rational `r`.

Example
```
gap> CeilingOfRational(3/5);
1
```

A.2 Periodic subadditive functions

A periodic function f of period m from the set $\mathbb{N}$ of natural numbers into itself may be specified through a list of m natural numbers. The function f is said to be *subadditive* if $f(i+j) \leq f(i)+f(j)$ and $f(0)=0$.

A.2.1 RepresentsPeriodicSubAdditiveFunction

▷ `RepresentsPeriodicSubAdditiveFunction(L)` (function)

`L` is a list of integers. `RepresentsPeriodicSubAdditiveFunction` returns `true` or `false` according to whether `L` represents a periodic subadditive function f periodic of period m or not. To avoid defining $f(0)$ (which we assume to be 0) we define $f(m)=0$ and so the last element of the list must be 0. This technical need is due to the fact that positions in a list must be positive (not a 0).

Example
```
gap> RepresentsPeriodicSubAdditiveFunction([1,2,3,4,0]);
true
```

A.2.2 IsListOfIntegersNS

▷ `IsListOfIntegersNS(L)` (function)

Detects whether `L` is a nonempty list of integers.

Example
```
gap> IsListOfIntegersNS([1,-1,0]);
true
gap> IsListOfIntegersNS(2);
false
gap> IsListOfIntegersNS([[2],3]);
false
gap> IsListOfIntegersNS([]);
false
```

Appendix B

"Random" functions

Here we describe some functions which allow to create several "random" objects. We make use of the function `RandomList`.

B.1 Random functions

B.1.1 RandomNumericalSemigroup

▷ `RandomNumericalSemigroup(n, a[, b])` (function)

Returns a “random" numerical semigroup with no more than *n* generators in [1..*a*] (or in [*a*..*b*], if *b* is present).

Example
```
gap> RandomNumericalSemigroup(3,9);
<Numerical semigroup with 3 generators>
gap> RandomNumericalSemigroup(3,9,55);
<Numerical semigroup with 3 generators>
```

B.1.2 RandomListForNS

▷ `RandomListForNS(n, a, b)` (function)

Returns a set of length not greater than *n* of random integers in `[a..b]` whose GCD is 1. It is used to create "random" numerical semigroups.

Example
```
gap> RandomListForNS(13,1,79);
[ 22, 26, 29, 31, 34, 46, 53, 61, 62, 73, 76 ]
```

B.1.3 RandomModularNumericalSemigroup

▷ `RandomModularNumericalSemigroup(k[, m])` (function)

Returns a “random" modular numerical semigroup $S(a,b)$ with $a \leq k$ (see 1) and multiplicity at least m, were m is the second argument, which may not be present..

Example
```
gap> RandomModularNumericalSemigroup(9);
<Modular numerical semigroup satisfying 5x mod 6 <= x >
gap> RandomModularNumericalSemigroup(10,25);
<Modular numerical semigroup satisfying 4x mod 157 <= x >
```

B.1.4 RandomProportionallyModularNumericalSemigroup

▷ `RandomProportionallyModularNumericalSemigroup(k[, m])` (function)

Returns a “random" proportionally modular numerical semigroup $S(a,b,c)$ with $a \leq k$ (see 1) and multiplicity at least m, were m is the second argument, which may not be present.

Example
```
gap> RandomProportionallyModularNumericalSemigroup(9);
<Proportionally modular numerical semigroup satisfying 2x mod 3 <= 2x >
gap> RandomProportionallyModularNumericalSemigroup(10,25);
<Proportionally modular numerical semigroup satisfying 6x mod 681 <= 2x >
```

B.1.5 RandomListRepresentingSubAdditiveFunction

▷ `RandomListRepresentingSubAdditiveFunction(m, a)` (function)

Produces a “random" list representing a subadditive function (see 1) which is periodic with period *m* (or less). When possible, the images are in `[a..20*a]`. (Otherwise, the list of possible images is enlarged.)

Example
```
gap> RandomListRepresentingSubAdditiveFunction(7,9);
[ 173, 114, 67, 0 ]
gap> RepresentsPeriodicSubAdditiveFunction(last);
true
```

B.1.6 NumericalSemigroupWithRandomElementsAndFrobenius

▷ `NumericalSemigroupWithRandomElementsAndFrobenius(n, mult, frob)` (function)

Produces a "random" semigroup containing (at least) `n` elements greater than or equal to `mult` and less than `frob`, chosen at random. The semigroup returned has multiplicity chosen at random but no smaller than `mult` and having Frobenius number chosen at random but not greater than `frob`. Returns *fail* if `frob` is greater than `mult`.

Example
```
gap> ns := NumericalSemigroupWithRandomElementsAndFrobenius(5,10,50);
<Numerical semigroup with 17 generators>
gap> MinimalGeneratingSystem(ns);
[ 12, 13, 19, 27, 47 ]
gap> SmallElements(ns);
[ 0, 12, 13, 19, 24, 25, 26, 27, 31, 32, 36, 37, 38, 39, 40, 43 ]
gap> ns2 := NumericalSemigroupWithRandomElementsAndFrobenius(5,10,9);
#I  The third argument must not be smaller than the second
fail
```

```
gap> ns3 := NumericalSemigroupWithRandomElementsAndFrobenius(5,10,10);
<Proportionally modular numerical semigroup satisfying 20x mod 200 <= 10x >
gap> MinimalGeneratingSystem(ns3);
[ 10 .. 19 ]
gap> SmallElements(ns3);
[ 0, 10 ]
```

Appendix C

Contributions

Sebastian Gutsche helped in the implementation of inference of properties from already known properties. Max Horn adapted the definition of the objects numerical and affine semigroups; the behave like lists of integers or lists of lists of integers (affine case), and one can intersect numerical semigroups with lists of integers, or affine semigroup with cartesian products of lists of integers.

C.1 Functions implemented by A. Sammartano

A. Sammartano implemented the following functions.
`IsAperySetGammaRectangular` (6.2.10),
`IsAperySetBetaRectangular` (6.2.11),
`IsAperySetAlphaRectangular` (6.2.12),
`TypeSequenceOfNumericalSemigroup` (7.1.25),
`IsGradedAssociatedRingNumericalSemigroupBuchsbaum` (7.4.2),
`IsGradedAssociatedRingNumericalSemigroupBuchsbaum` (7.4.2),
`TorsionOfAssociatedGradedRingNumericalSemigroup` (7.4.3),
`BuchsbaumNumberOfAssociatedGradedRingNumericalSemigroup` (7.4.4),
`IsMpureNumericalSemigroup` (7.4.5),
`IsPureNumericalSemigroup` (7.4.6),
`IsGradedAssociatedRingNumericalSemigroupGorenstein` (7.4.7),
`IsGradedAssociatedRingNumericalSemigroupCI` (7.4.8).

C.2 Functions implemented by C. O'Neill

C. O'Neill implemented the following functions described in [BOP14]:
`OmegaPrimalityOfElementListInNumericalSemigroup` (9.4.2),
`FactorizationsElementListWRTNumericalSemigroup` (9.1.3),
`DeltaSetPeriodicityBoundForNumericalSemigroup` (9.2.7),
`DeltaSetPeriodicityStartForNumericalSemigroup` (9.2.8),
`DeltaSetListUpToElementWRTNumericalSemigroup` (9.2.9),
`DeltaSetUnionUpToElementWRTNumericalSemigroup` (9.2.10),
`DeltaSetOfNumericalSemigroup` (9.2.11).
And contributed to:
`DeltaSetOfAffineSemigroup` (11.4.3).

C.3 Functions implemented by K. Stokes

Klara Stokes helped with the implementation of functions related to patterns for ideals of numerical semigroups 7.3.

C.4 Functions implemented by I. Ojeda and C. J. Moreno Ávila

Ignacio and Carlos Jesús implemented the algorithms given in [Rou08] and [MCOT15] for the calculation of the Frobenius number and Apéry set of a numerical semigroup using Gröbner basis calculations. These methods will be used if 4ti2 is loaded (either 4ti2Interface or 4ti2gap). A faster algorithm is employed provided that singular is loaded.

C.5 Functions implemented by I. Ojeda

Ignacio also implemented the following functions.
`AlmostSymmetricNumericalSemigroupsFromIrreducibleAndGivenType` (6.3.2),
`AlmostSymmetricNumericalSemigroupsWithFrobeniusNumberAndType` (6.3.5),
Ignacio also implemented the new version of
`AlmostSymmetricNumericalSemigroupsWithFrobeniusNumber` (6.3.4),

C.6 Functions implemented by A. Sánchez-R. Navarro

Alfredo helped in the implementation of methods for *4ti2gap* of the following functions.
`FactorizationsVectorWRTList` (11.4.1),
`DegreesOfPrimitiveElementsOfAffineSemigroup` (11.3.9),
`MinimalPresentationOfAffineSemigroup` (11.3.4).
He also helped in preliminary versions of the following functions.
`CatenaryDegreeOfSetOfFactorizations` (9.3.1),
`TameDegreeOfSetOfFactorizations` (9.3.6),
`TameDegreeOfNumericalSemigroup` (9.3.12),
`TameDegreeOfAffineSemigroup` (11.4.8),
`OmegaPrimalityOfElementInAffineSemigroup` (11.4.9),
`CatenaryDegreeOfAffineSemigroup` (11.4.4),
`MonotoneCatenaryDegreeOfSetOfFactorizations` (9.3.4).
`EqualCatenaryDegreeOfSetOfFactorizations` (9.3.3).
`AdjacentCatenaryDegreeOfSetOfFactorizations` (9.3.2).
`HomogeneousCatenaryDegreeOfAffineSemigroup` (11.4.6).

C.7 Functions implemented by G. Zito

Giuseppe gave the algorithms for the current version functions
`ArfNumericalSemigroupsWithFrobeniusNumber` (8.2.4),
`ArfNumericalSemigroupsWithFrobeniusNumberUpTo` (8.2.5),
`ArfNumericalSemigroupsWithGenus` (8.2.6),
`ArfNumericalSemigroupsWithGenusUpTo` (8.2.7),

`ArfCharactersOfArfNumericalSemigroup` (8.2.3).

C.8 Functions implemented by A. Herrera-Poyatos

Andrés Herrera-Poyatos gave new implementations of
`IsSelfReciprocalUnivariatePolynomial` (10.1.9) and
`IsKroneckerPolynomial` (10.1.7). Andrés is also coauthor of the dot functions, see Chapter 14

C.9 Functions implemented by Benjamin Heredia

Benjamin Heredia implemented a preliminary version of
`FengRaoDistance` (9.7.1).

C.10 Functions implemented by Juan Ignacio García-García

Juan Ignacio implemented a preliminary version of
`NumericalSemigroupsWithFrobeniusNumber` (5.4.1).

References

[AAGS17] A. Abbas, A. Assi, and P. A. García-Sánchez. Canonical bases of modules over one dimensional k-algebras. *arxiv:1703.02825*, 2017. 92

[AGGS10] F. Aguiló-Gost and P. A. García-Sánchez. Factoring in embedding dimension three numerical semigroups. *Electron. J. Combin.*, 17(1):Research Paper 138, 21, 2010. 73

[AGS13] A. Assi and P. A. García-Sánchez. Constructing the set of complete intersection numerical semigroups with a given frobenius number. *Applicable Algebra in Engineering, Communication and Computing*, 2013. 42

[AGS16] A. Assi and P. A. García-Sánchez. Algorithms for curves with one place at infinity. *Journal of Symbolic Computation*, 74:475–492, 2016. 90

[AGSM17] A. Assi, P. A. García-Sánchez, and V. Micale. Bases of subalgebras of k[[x]] and k[x]. *Journal of Symbolic Computation*, 79:4–22, 2017. 92

[BA04] Maria Bras-Amorós. Acute semigroups, the order bound on the minimum distance, and the Feng-Rao improvements. *IEEE Trans. Inform. Theory*, 50(6):1282–1289, 2004. 25

[BA08] M. Bras-Amorós. Fibonacci-like behavior of the number of numerical semigroups of a given genus. *Semigroup Forum*, 76:379–384, 2008. 37

[BAGS06] Maria Bras-Amorós and Pedro A. García-Sánchez. Patterns on numerical semigroups. *Linear Algebra Appl.*, 414(2-3):652–669, 2006. 60

[BC77] J. Bertin and P. Carbonne. Semi-groupes d'entiers et application aux branches. *J. Algebra*, 49(1):81–95, 1977. 42, 43

[BD89] R. J. Bradford and J. H. Davenport. Effective tests for cyclotomic polynomials. In *Symbolic and algebraic computation (Rome, 1988)*, volume 358 of *Lecture Notes in Comput. Sci.*, pages 244–251. Springer, Berlin, 1989. 88, 89

[BDF97] V. Barucci, D. D. Dobbs, and M. Fontana. *Maximality properties in numerical semigroups and applications to one-dimensional analytically irreducible local domains*. Number 598 in Memoirs of the American Mathematical Society. American Mathematical Society, 1997. 40, 48, 55

[BF97] V. Barucci and R. Fröberg. One-dimensional almost gorenstein rings. *J. Algebra*, 188:418–442, 1997. 46, 55

[BF06] V. Barucci and R. Fröberg. Associated graded rings of one-dimensional analytically irreducible rings. *J. Algebra*, 304:349–358, 2006. 58, 62

[BGJ+14] M. Barakat, S. Gutsche, S. Jambor, M. Lange-Hegermann, A. Lorenz, and O. Motsak. GradedModules, a homalg based package for the abelian category of finitely presented graded modules over computable graded rings, Version 2014.09.17. `http://homalg.math.rwth-aachen.de/~barakat/homalg-project/GradedModules/`, Sep 2014. GAP package. 115

[BGSG11] V. Blanco, P. A. García-Sánchez, and A Geroldinger. Semigroup-theoretical characterizations of arithmetical invariants with applications to numerical monoids and krull monoids. *Illinois J. Math.*, 55:1385–1414, 2011. 32, 82, 105

[BH13] Lance Bryant and James Hamblin. The maximal denumerant of a numerical semigroup. *Semigroup Forum*, 86(3):571–582, 2013. 77, 78

[BIRC14] W. Bruns, B. Ichim, T. Römer, and Söger C. Normaliz. algorithms for rational cones and affine monoids. `http://www.math.uos.de/normaliz`, 2014. 115

[BOP14] T. Barron, C. O'Neill, and R. Pelayo. On the computation of delta sets and ω-primality in numerical monoids. *preprint*, 2014. 128

[BOR18] M. Branco, I. Ojeda, and J. C. Rosales. Almost symmetric numerical semigroups with a given frobenus number and type. *preprint*, 2018. 46

[BR09] M. Bullejos and J. C. Rosales. Proportionally modular Diophantine inequalities and the Stern-Brocot tree. *Math. Comp.*, 78(266):1211–1226, 2009. 123

[BR13] Victor Blanco and José Carlos Rosales. The tree of irreducible numerical semigroups with fixed Frobenius number. *Forum Math.*, 25(6):1249–1261, 2013. 40

[Bry10] L. Bryant. Goto numbers of a numerical semigroup ring and the Gorensteiness of associated graded rings. *Comm. Algebra*, 38(6):2092–2128, 2010. 63, 64

[CBJZA13] T. Cortadellas Benítez, R. Jafari, and S. Zarzuela Armengou. On teh apéry sets of monomial curves. *Semigroup Forum*, 86:289–320, 2013. 59, 63

[CD94] E. Contejean and H. Devie. An efficient incremental algorithm for solving systems of linear Diophantine equations. *Inform. and Comput.*, 113(1):143–172, 1994. 98, 99

[CdG12] M. Costantini and W. de Graaf. Gap package singular; the gap interface to singular. `http://gap-system.org/Packages/singular.html`, 2012. 115

[CGSD07] S. T. Chapman, P. A. García-Sánchez, and Llena D. The catenary and tame degree of numerical semigroups. *Forum Math.*, pages 1–13, 2007. 71

[CGSL+06] S. T. Chapman, P. A. García-Sánchez, D. Llena, V. Ponomarenko, and J. C. Rosales. The catenary and tame degree in finitely generated commutative cancellative monoids. *Manuscripta Math.*, 120(3):253–264, 2006. 71

[CHM06] S. T. Chapman, M. T. Holden, and T. A. Moore. Full elasticity in atomic monoids and integral domains. *Rocky Mountain J. Math.*, 36(5):1437–1455, 2006. 71

[CRA13] Jonathan Chappelon and Jorge Luis Ramírez Alfonsín. On the Möbius function of the locally finite poset associated with a numerical semigroup. *Semigroup Forum*, 87(2):313–330, 2013. 84

[DGH01] Marco D'Anna, Anna Guerrieri, and William Heinzer. Invariants of ideals having principal reductions. *Comm. Algebra*, 29(2):889–906, 2001. 57, 58

[DGPS12] Wolfram Decker, Gert-Martin Greuel, Gerhard Pfister, and Hans Schönemann. SINGULAR 3-1-6 — A computer algebra system for polynomial computations. `http://www.singular.uni-kl.de`, 2012. 115

[DGS16] M. Delgado and P. A. García-Sánchez. numericalsgps, a GAP package for numerical semigroups. *ACM Commun. Comput. Algebra*, 50(1):12–24, 2016. 7

[DGSM06] M. Delgado, P. A. García-Sánchez, and J. Morais. On the GAP package numericalsgps. In *Fifth Conference on Discrete Mathematics and Computer Science (Spanish)*, volume 23 of *Ciencias (Valladolid)*, pages 271–278. Univ. Valladolid, Secr. Publ. Intercamb. Ed., Valladolid, 2006. 7

[DGSM16] M. D'Anna, P. A. García-Sánchez, and E. Micale. Good semigroups. *preprint*, 2016. 106

[DGSRP16] M. Delgado, P. A. García-Sánchez, and A. M. Robles-Pérez. Numerical semigroups with a given set of pseudo-frobenius numbers. *LMS J. Comput. Math.*, 19:186–205, 2016. 37

[DMS11] M. D'Anna, V. Micale, and A. Sammartano. On the associated graded ring of a semigroup ring. *J. Commut. Algebra*, 3(2):147–168, 2011. 64

[DMS13] M. D'Anna, V. Micale, and A. Sammartano. When the associated graded ring of a semigroup ring is complete intersection. *J. Pure Appl. Algebra*, 217(6):1007–1017, 2013. 64

[DMS14] Marco D'Anna, Vincenzo Micale, and Alessio Sammartano. Classes of complete intersection numerical semigroups. *Semigroup Forum*, 88(2):453–467, 2014. 45, 46

[DMV09] M. D'Anna, M. Mezzasalma, and Micale V. On the buchsbaumness of the associated graded ring of a one-dimensional local ring. *Comm. Algebra*, 37:1594–1603, 2009. 63

[DS13] M. D'Anna and F. Strazzanti. The numerical duplication of a numerical semigroup. *Semigroup Forum*, 87(1):149–160, 2013. 35

[Eli01] J. Elias. On the deep structure of the blowing-up of curve singularities. *Math. Proc. Camb. Phil. Soc.*, 131:227–240, 2001. 58

[Eli15] S. Eliahou. Wilf's conjecture and Macaulay's theorem. *preprint*, 2015. 27

[ES96] David Eisenbud and Bernd Sturmfels. Binomial ideals. *Duke Math. J.*, 84(1):1–45, 1996. 103

[FGR87] R. Fröberg, C. Gottlieb, and Häggkvist R. On numerical semigroups. *Semigroup Forum*, 35(1):63–83, 1987. 40

[GGMFVT15] J. I. García-García, M. A. Moreno-Frías, and A. Vigneron-Tenorio. Computation of delta sets of numerical monoids. *Monatsh. Math.*, 178:457–472, 2015. 75

[GHK06] A. Geroldinger and F. Halter-Koch. *Non-unique Factorizations: Algebraic, Combinatorial and Analytic Theory*. Chapman & Hall/CRC, 2006. 71

[GHS14] S Gutsche, M. Horn, and C. Söger. Normalizinterface for gap. `https://github.com/fingolfin/NormalizInterface`, 2014. 115

[GSHKR17] P. A. García-Sánchez, B. A. Heredia, H. I. Karakas, and J. C. Rosales. Parametrizing arf numerical semigroups. *J. Algebra Appl.*, 16:1750209 (31 pages), 2017. 23, 69

[GSO10] P. A. García-Sánchez and I. Ojeda. Uniquely presented finitely generated commutative monoids. *Pacific J. Math.*, 249:91–105, 2010. 31, 32

[GSOSRN13] P. A. García Sánchez, I. Ojeda, and A. Sánchez-R.-Navarro. Factorization invariants in half-factorial affine semigroups. *Internat. J. Algebra Comput.*, 23(1):111–122, 2013. 83, 104

[GSOW17] P. A. García-Sánchez, C. O'Neill, and G. Webb. On the computation of factorization invariants for affine semigroups. *arxiv:1504.02998*, 2017. 103, 104

[Gut] S. Gutsche. 4ti2interface, a link to 4ti2. `http://www.gap-system.org/Packages/4ti2interface.html`. 115

[Her70] Jürgen Herzog. Generators and relations of abelian semigroups and semigroup rings. *Manuscripta Math.*, 3:175–193, 1970. 101

[HS04] K. Herzinger and R. Sanford. Minimal generating sets for relative ideals in numerical semigroups of multiplicity eight. *Communications in Algebra*, 32(12):4713–4731, 2004. 53

[KP95] Christoph Kirfel and Ruud Pellikaan. The minimum distance of codes in an array coming from telescopic semigroups. *IEEE Trans. Inform. Theory*, 41(6, part 1):1720–1732, 1995. Special issue on algebraic geometry codes. 42, 44

[KW14] E. Kunz and R. Waldi. Geometrical illustration of numerical semigroups and of some of their invariants. *Semigroup Forum*, 89(3):664–691, 2014. 26

[MCOT15] Guadalupe Márquez-Campos, Ignacio Ojeda, and José M. Tornero. On the computation of the Apéry set of numerical monoids and affine semigroups. *Semigroup Forum*, 91(1):139–158, 2015. 129

[Mic02] V. Micale. On monomial semigroups. *Communications in Algebra*, 30:4687 – 4698, 2002. 93

[Mor14] Pieter Moree. Numerical semigroups, cyclotomic polynomials, and Bernoulli numbers. *Amer. Math. Monthly*, 121(10):890–902, 2014. 87, 89

[Phi10] A. Philipp. A characterization of arithmetical invariants by the monoid of relations. *Semigroup Forum*, 81:424–434, 2010. 103, 104

[RB03] J. C. Rosales and M. B. Branco. Irreducible numerical semigroups. *Pacific J. Math.*, 209(1):131–143, 2003. 40

[RGS98] J.C. Rosales and Pedro A. García-Sánchez. Nonnegative elements of subgroups of $\mathbb{Z}^n$. *Linear Algebra and its Applications*, 270(1-3):351– 357, 1998. 98

[RGS99a] J. C. Rosales and P. A. García-Sánchez. *Finitely generated commutative monoids*. Nova Science Publishers, New York, 1999. 30, 100

[RGS99b] J. C. Rosales and P. A. García-Sánchez. On free affine semigroups. *Semigroup Forum*, 58(3):367–385, 1999. 99

[RGS04] J. C. Rosales and P. A. García-Sánchez. Every positive integer is the frobenius number of an irreducible numerical semigroup with at most four generators. *Ark. Mat.*, 42:301–306, 2004. 40, 41

[RGS09] J. C. Rosales and P. A. García-Sánchez. *Numerical Semigroups*. Springer, 2009. 7, 69, 72

[RGS14] J. C. Rosales and P. A. García-Sánchez. Constructing almost symmetric numerical semigroups from irreducible numerical semigroups. *Comm. Algebra*, 42(3):1362–1367, 2014. 46

[RGSB02] J. C. Rosales, García-García J. I. García-Sánchez, P. A. and, and M. B. Branco. Systems of inequalities and numerical semigroups. *Journal of the London Mathematical Society*, 65:611–623, 6 2002. 23, 99

[RGSGGB03] J. C. Rosales, P. A. García-Sánchez, J. I. García-García, and M. B. Branco. Numerical semigroups with maximal embedding dimension. *J. Algebra*, 2:47–53, 2003. 37, 66

[RGSGGB04] J. C. Rosales, P. A. García-Sánchez, J. I. García-García, and M. B. Branco. Arf numerical semigroups. *J. Algebra*, 276:3–12, 2004. 67

[RGSGGJM03] J. C. Rosales, P. A. García-Sánchez, J. I. García-García, and J. A. Jiménez-Madrid. The oversemigroups of a numerical semigroup. *Semigroup Forum*, 67:145–158, 2003. 33, 36, 40

[RGSGGJM04] J. C. Rosales, P. A. García-Sánchez, J. I. García-García, and J. A. Jiménez Madrid. Fundamental gaps in numerical semigroups. *J. Pure Appl. Algebra*, 189(1-3):301–313, 2004. 37

[Ros96a] J. C. Rosales. An algorithmic method to compute a minimal relation for any numerical semigroup. *Internat. J. Algebra Comput.*, 6:441–455, 1996. 30, 31

[Ros96b] J. C. Rosales. On numerical semigroups. *Semigroup Forum*, 52:307–318, 1996. 14

[Rou08] Bjarke Hammersholt Roune. Solving thousand-digit Frobenius problems using Gröbner bases. *J. Symbolic Comput.*, 43(1):1–7, 2008. 129

[Spi15] Dario Spirito. Star operations on numerical semigroups. *Comm. Algebra*, 43(7):2943–2963, 2015. 59

[Sto15] K. Stokes. Patterns of ideals of numerical semigroups. *preprint*, 2015. 59

[SW86] L. A. Székely and N. C. Wormald. Generating functions for the Frobenius problem with 2 and 3 generators. *Math. Chronicle*, 15:49–57, 1986. 32

[tt] 4ti2 team. 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces. Available at www.4ti2.de. 115

[Ugo16] Simone Ugolini. On numerical semigroups closed with respect to the action of affine maps. *arXiv*, 1505.06580, 2016. 11

[Wil78] Herbert S. Wilf. A circle-of-lights algorithm for the "money-changing problem". *Amer. Math. Monthly*, 85(7):562–565, 1978. 27

[Zar86] O. Zariski. *Le problème des modules pour les courbes planes*. Hermann, 1986. 42, 44

Index