Arithmetic Subgroups of $\mathrm{SL}_2(\mathbf{Z})$

Release 10.4

The Sage Development Team

Jul 23, 2024

CONTENTS

This chapter describes the basic functionality for finite index subgroups of the modular group $\mathrm{SL}_2(\mathbf{Z})$.

CHAPTER

ONE

ARITHMETIC SUBGROUPS, FINITE INDEX SUBGROUPS OF $\mathrm{SL}_2(\mathbf{Z})$

class sage.modular.arithgroup.arithgroup_generic.**ArithmeticSubgroup**

Bases: Group

Base class for arithmetic subgroups of $\mathrm{SL}_2(\mathbf{Z})$. Not intended to be used directly, but still includes quite a few general-purpose routines which compute data about an arithmetic subgroup assuming that it has a working element testing routine.

Element

alias of *ArithmeticSubgroupElement*

are_equivalent(*x*, *y*, *trans=False*)

Test whether or not cusps x and y are equivalent modulo self.

If self has a reduce_cusp() method, use that; otherwise do a slow explicit test.

If trans = False, returns True or False. If trans = True, then return either False or an element of self mapping x onto y.

EXAMPLES:

```
sage: Gamma0(7).are_equivalent(Cusp(1/3), Cusp(0), trans=True)
[  3  -1]
[-14   5]
sage: Gamma0(7).are_equivalent(Cusp(1/3), Cusp(1/7))
False
```

```
>>> from sage.all import *
>>> Gamma0(Integer(7)).are_equivalent(Cusp(Integer(1)/Integer(3)),␣
↪Cusp(Integer(0)), trans=True)
[  3  -1]
[-14   5]
>>> Gamma0(Integer(7)).are_equivalent(Cusp(Integer(1)/Integer(3)),␣
↪Cusp(Integer(1)/Integer(7)))
False
```

as_permutation_group()

Return `self` as an arithmetic subgroup defined in terms of the permutation action of $SL(2, \mathbf{Z})$ on its right cosets.

This method uses Todd-Coxeter enumeration (via the method *todd_coxeter()*) which can be extremely slow for arithmetic subgroups with relatively large index in $SL(2, \mathbf{Z})$.

EXAMPLES:

```
sage: # needs sage.groups
sage: G = Gamma(3)
sage: P = G.as_permutation_group(); P
Arithmetic subgroup of index 24
sage: G.ncusps() == P.ncusps()
True
sage: G.nu2() == P.nu2()
True
sage: G.nu3() == P.nu3()
True
sage: G.an_element() in P
True
sage: P.an_element() in G
True
```

```
>>> from sage.all import *
>>> # needs sage.groups
>>> G = Gamma(Integer(3))
>>> P = G.as_permutation_group(); P
Arithmetic subgroup of index 24
>>> G.ncusps() == P.ncusps()
True
>>> G.nu2() == P.nu2()
True
>>> G.nu3() == P.nu3()
True
>>> G.an_element() in P
True
>>> P.an_element() in G
True
```

coset_reps(*G=None*)

Return right coset representatives for self \ G, where G is another arithmetic subgroup that contains self. If G = None, default to G = SL2Z.

For generic arithmetic subgroups G this is carried out by Todd-Coxeter enumeration; here G is treated as a black box, implementing nothing but membership testing.

EXAMPLES:

```
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().coset_
↪reps()
Traceback (most recent call last):
...
NotImplementedError: Please implement _contains_sl2 for <class 'sage.modular.
↪arithgroup.arithgroup_generic.ArithmeticSubgroup_with_category'>
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.coset_
↪reps(Gamma0(3))
[
[1 0]  [ 0 -1]  [ 0 -1]  [ 0 -1]
[0 1], [ 1  0], [ 1  1], [ 1  2]
]
```

```
>>> from sage.all import *
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().coset_
↪reps()
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
...
NotImplementedError: Please implement _contains_sl2 for <class 'sage.modular.
↪arithgroup.arithgroup_generic.ArithmeticSubgroup_with_category'>
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.coset_
↪reps(Gamma0(Integer(3)))
[
[1 0]  [ 0 -1]  [ 0 -1]  [ 0 -1]
[0 1], [ 1  0], [ 1  1], [ 1  2]
]
```

cusp_data(*c*)

Return a triple (g, w, t) where g is an element of self generating the stabiliser of the given cusp, w is the width of the cusp, and t is 1 if the cusp is regular and -1 if not.

EXAMPLES:

```
sage: Gamma1(4).cusp_data(Cusps(1/2))
(
[ 1 -1]
[ 4 -3], 1, -1
)
```

```
>>> from sage.all import *
>>> Gamma1(Integer(4)).cusp_data(Cusps(Integer(1)/Integer(2)))
(
[ 1 -1]
[ 4 -3], 1, -1
)
```

cusp_width(*c*)

Return the width of the orbit of cusps represented by c.

EXAMPLES:

```
sage: Gamma0(11).cusp_width(Cusps(oo))
1
sage: Gamma0(11).cusp_width(0)
11
sage: [Gamma0(100).cusp_width(c) for c in Gamma0(100).cusps()]
[100, 1, 4, 1, 1, 1, 4, 25, 1, 1, 4, 1, 25, 4, 1, 4, 1, 1]
```

```
>>> from sage.all import *
>>> Gamma0(Integer(11)).cusp_width(Cusps(oo))
1
>>> Gamma0(Integer(11)).cusp_width(Integer(0))
11
>>> [Gamma0(Integer(100)).cusp_width(c) for c in Gamma0(Integer(100)).cusps()]
[100, 1, 4, 1, 1, 1, 4, 25, 1, 1, 4, 1, 25, 4, 1, 4, 1, 1]
```

cusps(*algorithm='default'*)

Return a sorted list of inequivalent cusps for `self`, i.e. a set of representatives for the orbits of self on $\mathbb{P}^1(\mathbf{Q})$.

These should be returned in a reduced form where this makes sense.

INPUT:

- `algorithm` – which algorithm to use to compute the cusps of `self`. `'default'` finds representatives for a known complete set of cusps. `'modsym'` computes the boundary map on the space of weight two modular symbols associated to `self`, which finds the cusps for `self` in the process.

EXAMPLES:

```
sage: Gamma0(36).cusps()
[0, 1/18, 1/12, 1/9, 1/6, 1/4, 1/3, 5/12, 1/2, 2/3, 5/6, Infinity]
sage: Gamma0(36).cusps(algorithm='modsym') == Gamma0(36).cusps()            #␣
↪needs sage.libs.flint
True
sage: GammaH(36, [19,29]).cusps() == Gamma0(36).cusps()
True
sage: Gamma0(1).cusps()
[Infinity]
```

```
>>> from sage.all import *
>>> Gamma0(Integer(36)).cusps()
[0, 1/18, 1/12, 1/9, 1/6, 1/4, 1/3, 5/12, 1/2, 2/3, 5/6, Infinity]
>>> Gamma0(Integer(36)).cusps(algorithm='modsym') == Gamma0(Integer(36)).
↪cusps()             # needs sage.libs.flint
True
>>> GammaH(Integer(36), [Integer(19),Integer(29)]).cusps() ==␣
↪Gamma0(Integer(36)).cusps()
True
>>> Gamma0(Integer(1)).cusps()
[Infinity]
```

dimension_cusp_forms(*k=2*)

Return the dimension of the space of weight k cusp forms for this group. For $k \geq 2$, this is given by a standard formula in terms of k and various invariants of the group; see Diamond + Shurman, "A First Course in Modular Forms", section 3.5 and 3.6. If k is not given, default to k = 2.

For dimensions of spaces of cusp forms with character for Gamma1, use the dimension_cusp_forms method of the Gamma1 class, or the standalone function dimension_cusp_forms().

For weight 1 cusp forms this generic implementation only works in cases where one can prove solely via Riemann-Roch theory that there aren't any cusp forms (i.e. when the number of regular cusps is strictly greater than the degree of the canonical divisor). Otherwise a NotImplementedError is raised.

EXAMPLES:

```
sage: Gamma1(31).dimension_cusp_forms(2)
26
sage: Gamma(5).dimension_cusp_forms(1)
0
sage: Gamma1(4).dimension_cusp_forms(1) # irregular cusp
0
sage: Gamma(13).dimension_cusp_forms(1)
Traceback (most recent call last):
...
NotImplementedError: Computation of dimensions of weight 1 cusp forms spaces␣
↪not implemented in general
```

```
>>> from sage.all import *
>>> Gamma1(Integer(31)).dimension_cusp_forms(Integer(2))
26
>>> Gamma(Integer(5)).dimension_cusp_forms(Integer(1))
```

(continues on next page)

(continued from previous page)

```
0
>>> Gamma1(Integer(4)).dimension_cusp_forms(Integer(1)) # irregular cusp
0
>>> Gamma(Integer(13)).dimension_cusp_forms(Integer(1))
Traceback (most recent call last):
...
NotImplementedError: Computation of dimensions of weight 1 cusp forms spaces␣
↪not implemented in general
```

dimension_eis(*k=2*)

Return the dimension of the space of weight k Eisenstein series for this group, which is a subspace of the space of modular forms complementary to the space of cusp forms.

INPUT:

- k – an integer (default 2).

EXAMPLES:

```
sage: GammaH(33,[2]).dimension_eis()
7
sage: GammaH(33,[2]).dimension_eis(3)
0
sage: GammaH(33, [2,5]).dimension_eis(2)
3
sage: GammaH(33, [4]).dimension_eis(1)
4
```

```
>>> from sage.all import *
>>> GammaH(Integer(33),[Integer(2)]).dimension_eis()
7
>>> GammaH(Integer(33),[Integer(2)]).dimension_eis(Integer(3))
0
>>> GammaH(Integer(33), [Integer(2),Integer(5)]).dimension_eis(Integer(2))
3
>>> GammaH(Integer(33), [Integer(4)]).dimension_eis(Integer(1))
4
```

dimension_modular_forms(*k=2*)

Return the dimension of the space of weight k modular forms for this group. This is given by a standard formula in terms of k and various invariants of the group; see Diamond + Shurman, "A First Course in Modular Forms", section 3.5 and 3.6. If k is not given, defaults to k = 2.

For dimensions of spaces of modular forms with character for Gamma1, use the dimension_modular_forms method of the Gamma1 class, or the standalone function dimension_modular_forms().

For weight 1 modular forms this generic implementation only works in cases where one can prove solely via Riemann-Roch theory that there aren't any cusp forms (i.e. when the number of regular cusps is strictly greater than the degree of the canonical divisor). Otherwise a NotImplementedError is raised.

EXAMPLES:

```
sage: Gamma1(31).dimension_modular_forms(2)
55
sage: Gamma1(3).dimension_modular_forms(1)
1
sage: Gamma1(4).dimension_modular_forms(1) # irregular cusp
```

(continues on next page)

(continued from previous page)

```
1
sage: Gamma(13).dimension_modular_forms(1)
Traceback (most recent call last):
...
NotImplementedError: Computation of dimensions of weight 1 cusp forms spaces␣
↪not implemented in general
```

```
>>> from sage.all import *
>>> Gamma1(Integer(31)).dimension_modular_forms(Integer(2))
55
>>> Gamma1(Integer(3)).dimension_modular_forms(Integer(1))
1
>>> Gamma1(Integer(4)).dimension_modular_forms(Integer(1)) # irregular cusp
1
>>> Gamma(Integer(13)).dimension_modular_forms(Integer(1))
Traceback (most recent call last):
...
NotImplementedError: Computation of dimensions of weight 1 cusp forms spaces␣
↪not implemented in general
```

farey_symbol()

Return the Farey symbol associated to this subgroup. See the *farey_symbol* module for more information.

EXAMPLES:

```
sage: Gamma1(4).farey_symbol()
FareySymbol(Congruence Subgroup Gamma1(4))
```

```
>>> from sage.all import *
>>> Gamma1(Integer(4)).farey_symbol()
FareySymbol(Congruence Subgroup Gamma1(4))
```

gen(*i*)

Return the i-th generator of self, i.e. the i-th element of the tuple self.gens().

EXAMPLES:

```
sage: SL2Z.gen(1)
[1 1]
[0 1]
```

```
>>> from sage.all import *
>>> SL2Z.gen(Integer(1))
[1 1]
[0 1]
```

generalised_level()

Return the generalised level of `self`, i.e., the least common multiple of the widths of all cusps.

If `self` is *even*, Wohlfart's theorem tells us that this is equal to the (conventional) level of `self` when `self` is a congruence subgroup. This can fail if `self` is odd, but the actual level is at most twice the generalised level. See the paper by Kiming, Schuett and Verrill for more examples.

EXAMPLES:

```
sage: Gamma0(18).generalised_level()
18
sage: from sage.modular.arithgroup.arithgroup_perm import HsuExample18      #␣
↪needs sage.groups
sage: HsuExample18().generalised_level()                                    #␣
↪needs sage.groups
24
```

```
>>> from sage.all import *
>>> Gamma0(Integer(18)).generalised_level()
18
>>> from sage.modular.arithgroup.arithgroup_perm import HsuExample18      #␣
↪needs sage.groups
>>> HsuExample18().generalised_level()                                    #␣
↪needs sage.groups
24
```

In the following example, the actual level is twice the generalised level. This is the group G_2 from Example 17 of K-S-V.

```
sage: G = CongruenceSubgroup(8, [ [1,1,0,1], [3,-1,4,-1] ])
sage: G.level()
8
sage: G.generalised_level()
4
```

```
>>> from sage.all import *
>>> G = CongruenceSubgroup(Integer(8), [ [Integer(1),Integer(1),Integer(0),
↪Integer(1)], [Integer(3),-Integer(1),Integer(4),-Integer(1)] ])
>>> G.level()
8
>>> G.generalised_level()
4
```

generators(*algorithm='farey'*)

Return a list of generators for this congruence subgroup. The result is cached.

INPUT:

- `algorithm` (string): either `farey` or `todd-coxeter`.

If `algorithm` is set to `"farey"`, then the generators will be calculated using Farey symbols, which will always return a *minimal* generating set. See `farey_symbol` for more information.

If `algorithm` is set to `"todd-coxeter"`, a simpler algorithm based on Todd-Coxeter enumeration will be used. This is *exceedingly* slow for general subgroups, and the list of generators will be far from minimal (indeed it may contain repetitions).

EXAMPLES:

```
sage: Gamma(2).generators()
[
[1 2]  [ 3 -2]  [-1  0]
[0 1], [ 2 -1], [ 0 -1]
]
sage: Gamma(2).generators(algorithm="todd-coxeter")
[
[1 2]  [-1  0]  [ 1  0]  [-1  0]  [-1  2]  [-1  0]  [1 0]
```

(continues on next page)

(continued from previous page)

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

```
>>> from sage.all import *
>>> Gamma(Integer(2)).generators()
[
[1 2]  [ 3 -2]  [-1  0]
[0 1], [ 2 -1], [ 0 -1]
]
>>> Gamma(Integer(2)).generators(algorithm="todd-coxeter")
[
[1 2]  [-1  0]  [ 1  0]  [-1  0]  [-1  2]  [-1  0]  [1 0]
[0 1], [ 0 -1], [-2  1], [ 0 -1], [-2  3], [ 2 -1], [2 1]
]
```

gens (**args*, ***kwds*)

Return a tuple of generators for this congruence subgroup.

The generators need not be minimal. For arguments, see `generators()`.

EXAMPLES:

```
sage: SL2Z.gens()
(
[ 0 -1]  [1 1]
[ 1  0], [0 1]
)
```

```
>>> from sage.all import *
>>> SL2Z.gens()
(
[ 0 -1]  [1 1]
[ 1  0], [0 1]
)
```

genus ()

Return the genus of the modular curve of `self`.

EXAMPLES:

```
sage: Gamma1(5).genus()
0
sage: Gamma1(31).genus()
26
sage: from sage.modular.dims import dimension_cusp_forms
sage: Gamma1(157).genus() == dimension_cusp_forms(Gamma1(157), 2)
True
sage: GammaH(7, [2]).genus()
0
sage: [Gamma0(n).genus() for n in [1..23]]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 2, 2]
sage: [n for n in [1..200] if Gamma0(n).genus() == 1]
[11, 14, 15, 17, 19, 20, 21, 24, 27, 32, 36, 49]
```

```
>>> from sage.all import *
>>> Gamma1(Integer(5)).genus()
```

(continues on next page)

(continued from previous page)

```
0
>>> Gamma1(Integer(31)).genus()
26
>>> from sage.modular.dims import dimension_cusp_forms
>>> Gamma1(Integer(157)).genus() == dimension_cusp_forms(Gamma1(Integer(157)),
↪ Integer(2))
True
>>> GammaH(Integer(7), [Integer(2)]).genus()
0
>>> [Gamma0(n).genus() for n in (ellipsis_range(Integer(1),Ellipsis,
↪Integer(23)))]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 2, 2]
>>> [n for n in (ellipsis_range(Integer(1),Ellipsis,Integer(200))) if␣
↪Gamma0(n).genus() == Integer(1)]
[11, 14, 15, 17, 19, 20, 21, 24, 27, 32, 36, 49]
```

index()

Return the index of self in the full modular group.

EXAMPLES:

```
sage: Gamma0(17).index()
18
sage: sage.modular.arithgroup.congroup_generic.CongruenceSubgroup(5).index()
Traceback (most recent call last):
...
NotImplementedError
```

```
>>> from sage.all import *
>>> Gamma0(Integer(17)).index()
18
>>> sage.modular.arithgroup.congroup_generic.CongruenceSubgroup(Integer(5)).
↪index()
Traceback (most recent call last):
...
NotImplementedError
```

is_abelian()

Return True if this arithmetic subgroup is abelian.

Since arithmetic subgroups are always nonabelian, this always returns False.

EXAMPLES:

```
sage: SL2Z.is_abelian()
False
sage: Gamma0(3).is_abelian()
False
sage: Gamma1(12).is_abelian()
False
sage: GammaH(4, [3]).is_abelian()
False
```

```
>>> from sage.all import *
>>> SL2Z.is_abelian()
False
```

(continues on next page)

(continued from previous page)

```
>>> Gamma0(Integer(3)).is_abelian()
False
>>> Gamma1(Integer(12)).is_abelian()
False
>>> GammaH(Integer(4), [Integer(3)]).is_abelian()
False
```

is_congruence()

Return True if self is a congruence subgroup.

EXAMPLES:

```
sage: Gamma0(5).is_congruence()
True
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().is_
↪congruence()
Traceback (most recent call last):
...
NotImplementedError
```

```
>>> from sage.all import *
>>> Gamma0(Integer(5)).is_congruence()
True
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().is_
↪congruence()
Traceback (most recent call last):
...
NotImplementedError
```

is_even()

Return True precisely if this subgroup contains the matrix -1.

EXAMPLES:

```
sage: SL2Z.is_even()
True
sage: Gamma0(20).is_even()
True
sage: Gamma1(5).is_even()
False
sage: GammaH(11, [3]).is_even()
False
```

```
>>> from sage.all import *
>>> SL2Z.is_even()
True
>>> Gamma0(Integer(20)).is_even()
True
>>> Gamma1(Integer(5)).is_even()
False
>>> GammaH(Integer(11), [Integer(3)]).is_even()
False
```

is_finite()

Return True if this arithmetic subgroup is finite.

Since arithmetic subgroups are always infinite, this always returns False.

EXAMPLES:

```
sage: SL2Z.is_finite()
False
sage: Gamma0(3).is_finite()
False
sage: Gamma1(12).is_finite()
False
sage: GammaH(4, [3]).is_finite()
False
```

```
>>> from sage.all import *
>>> SL2Z.is_finite()
False
>>> Gamma0(Integer(3)).is_finite()
False
>>> Gamma1(Integer(12)).is_finite()
False
>>> GammaH(Integer(4), [Integer(3)]).is_finite()
False
```

is_normal()

Return True precisely if this subgroup is a normal subgroup of SL2Z.

EXAMPLES:

```
sage: Gamma(3).is_normal()
True
sage: Gamma1(3).is_normal()
False
```

```
>>> from sage.all import *
>>> Gamma(Integer(3)).is_normal()
True
>>> Gamma1(Integer(3)).is_normal()
False
```

is_odd()

Return True precisely if this subgroup does not contain the matrix -1.

EXAMPLES:

```
sage: SL2Z.is_odd()
False
sage: Gamma0(20).is_odd()
False
sage: Gamma1(5).is_odd()
True
sage: GammaH(11, [3]).is_odd()
True
```

```
>>> from sage.all import *
>>> SL2Z.is_odd()
False
>>> Gamma0(Integer(20)).is_odd()
False
>>> Gamma1(Integer(5)).is_odd()
```

(continues on next page)

(continued from previous page)

```
True
>>> GammaH(Integer(11), [Integer(3)]).is_odd()
True
```

is_parent_of(*x*)

Check whether this group is a valid parent for the element x. Required by Sage's testing framework.

EXAMPLES:

```
sage: Gamma(3).is_parent_of(ZZ(1))
False
sage: Gamma(3).is_parent_of([1,0,0,1])
False
sage: Gamma(3).is_parent_of(SL2Z([1,1,0,1]))
False
sage: Gamma(3).is_parent_of(SL2Z(1))
True
```

```
>>> from sage.all import *
>>> Gamma(Integer(3)).is_parent_of(ZZ(Integer(1)))
False
>>> Gamma(Integer(3)).is_parent_of([Integer(1),Integer(0),Integer(0),
↪Integer(1)])
False
>>> Gamma(Integer(3)).is_parent_of(SL2Z([Integer(1),Integer(1),Integer(0),
↪Integer(1)]))
False
>>> Gamma(Integer(3)).is_parent_of(SL2Z(Integer(1)))
True
```

is_regular_cusp(*c*)

Return True if the orbit of the given cusp is a regular cusp for self, otherwise False. This is automatically true if -1 is in self.

EXAMPLES:

```
sage: Gamma1(4).is_regular_cusp(Cusps(1/2))
False
sage: Gamma1(4).is_regular_cusp(Cusps(oo))
True
```

```
>>> from sage.all import *
>>> Gamma1(Integer(4)).is_regular_cusp(Cusps(Integer(1)/Integer(2)))
False
>>> Gamma1(Integer(4)).is_regular_cusp(Cusps(oo))
True
```

is_subgroup(*right*)

Return True if self is a subgroup of right, and False otherwise. For generic arithmetic subgroups this is done by the absurdly slow algorithm of checking all of the generators of self to see if they are in right.

EXAMPLES:

```
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().is_
↪subgroup(SL2Z)
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
...
NotImplementedError: Please implement _contains_sl2 for <class 'sage.modular.
↪arithgroup.arithgroup_generic.ArithmeticSubgroup_with_category'>
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.is_
↪subgroup(Gamma1(18), Gamma0(6))
True
```

```
>>> from sage.all import *
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().is_
↪subgroup(SL2Z)
Traceback (most recent call last):
...
NotImplementedError: Please implement _contains_sl2 for <class 'sage.modular.
↪arithgroup.arithgroup_generic.ArithmeticSubgroup_with_category'>
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.is_
↪subgroup(Gamma1(Integer(18)), Gamma0(Integer(6)))
True
```

matrix_space()

Return the parent space of the matrices, which is always `MatrixSpace(ZZ, 2)`.

EXAMPLES:

```
sage: Gamma(3).matrix_space()
Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
```

```
>>> from sage.all import *
>>> Gamma(Integer(3)).matrix_space()
Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
```

ncusps()

Return the number of cusps of this arithmetic subgroup. This is provided as a separate function since for dimension formulae in even weight all we need to know is the number of cusps, and this can be calculated very quickly, while enumerating all cusps is much slower.

EXAMPLES:

```
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.
↪ncusps(Gamma0(7))
2
```

```
>>> from sage.all import *
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.
↪ncusps(Gamma0(Integer(7)))
2
```

ngens()

Return the size of the minimal generating set of self returned by *generators()*.

EXAMPLES:

```
sage: Gamma0(22).ngens()
8
sage: Gamma1(14).ngens()
13
```

(continues on next page)

(continued from previous page)

```
sage: GammaH(11, [3]).ngens()
3
sage: SL2Z.ngens()
2
```

```
>>> from sage.all import *
>>> Gamma0(Integer(22)).ngens()
8
>>> Gamma1(Integer(14)).ngens()
13
>>> GammaH(Integer(11), [Integer(3)]).ngens()
3
>>> SL2Z.ngens()
2
```

nirregcusps()

Return the number of cusps of self that are "irregular", i.e. their stabiliser can only be generated by elements with both eigenvalues -1 rather than +1. If the group contains -1, every cusp is clearly regular.

EXAMPLES:

```
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.
↪nirregcusps(Gamma1(4))
1
```

```
>>> from sage.all import *
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.
↪nirregcusps(Gamma1(Integer(4)))
1
```

nregcusps()

Return the number of cusps of self that are "regular", i.e. their stabiliser has a generator with both eigenvalues +1 rather than -1. If the group contains -1, every cusp is clearly regular.

EXAMPLES:

```
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.
↪nregcusps(Gamma1(4))
2
```

```
>>> from sage.all import *
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.
↪nregcusps(Gamma1(Integer(4)))
2
```

nu2()

Return the number of orbits of elliptic points of order 2 for this arithmetic subgroup.

EXAMPLES:

```
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().nu2()
Traceback (most recent call last):
...
NotImplementedError: Please implement _contains_sl2 for <class 'sage.modular.
↪arithgroup.arithgroup_generic.ArithmeticSubgroup_with_category'>
```

(continues on next page)

(continued from previous page)

```
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.
↪nu2(Gamma0(1105)) == 8
True
```

```
>>> from sage.all import *
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().nu2()
Traceback (most recent call last):
...
NotImplementedError: Please implement _contains_sl2 for <class 'sage.modular.
↪arithgroup.arithgroup_generic.ArithmeticSubgroup_with_category'>
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.
↪nu2(Gamma0(Integer(1105))) == Integer(8)
True
```

nu3()

Return the number of orbits of elliptic points of order 3 for this arithmetic subgroup.

EXAMPLES:

```
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().nu3()
Traceback (most recent call last):
...
NotImplementedError: Please implement _contains_sl2 for <class 'sage.modular.
↪arithgroup.arithgroup_generic.ArithmeticSubgroup_with_category'>
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.
↪nu3(Gamma0(1729)) == 8
True
```

```
>>> from sage.all import *
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().nu3()
Traceback (most recent call last):
...
NotImplementedError: Please implement _contains_sl2 for <class 'sage.modular.
↪arithgroup.arithgroup_generic.ArithmeticSubgroup_with_category'>
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.
↪nu3(Gamma0(Integer(1729))) == Integer(8)
True
```

We test that a bug in handling of subgroups not containing -1 is fixed:

```
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.
↪nu3(GammaH(7, [2]))
2
```

```
>>> from sage.all import *
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup.
↪nu3(GammaH(Integer(7), [Integer(2)]))
2
```

order()

Return the number of elements in this arithmetic subgroup.

Since arithmetic subgroups are always infinite, this always returns infinity.

EXAMPLES:

```
sage: SL2Z.order()
+Infinity
sage: Gamma0(5).order()
+Infinity
sage: Gamma1(2).order()
+Infinity
sage: GammaH(12, [5]).order()
+Infinity
```

```
>>> from sage.all import *
>>> SL2Z.order()
+Infinity
>>> Gamma0(Integer(5)).order()
+Infinity
>>> Gamma1(Integer(2)).order()
+Infinity
>>> GammaH(Integer(12), [Integer(5)]).order()
+Infinity
```

projective_index()

Return the index of the image of self in $\mathrm{PSL}_2(\mathbf{Z})$. This is equal to the index of self if self contains -1, and half of this otherwise.

This is equal to the degree of the natural map from the modular curve of self to the j-line.

EXAMPLES:

```
sage: Gamma0(5).projective_index()
6
sage: Gamma1(5).projective_index()
12
```

```
>>> from sage.all import *
>>> Gamma0(Integer(5)).projective_index()
6
>>> Gamma1(Integer(5)).projective_index()
12
```

reduce_cusp(c)

Given a cusp $c \in \mathbb{P}^1(\mathbf{Q})$, return the unique reduced cusp equivalent to c under the action of self, where a reduced cusp is an element $\frac{r}{s}$ with r,s coprime non-negative integers, s as small as possible, and r as small as possible for that s.

NOTE: This function should be overridden by all subclasses.

EXAMPLES:

```
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().reduce_
↪cusp(1/4)
Traceback (most recent call last):
...
NotImplementedError
```

```
>>> from sage.all import *
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().reduce_
↪cusp(Integer(1)/Integer(4))
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
...
NotImplementedError
```

sturm_bound(*weight=2*)

Returns the Sturm bound for modular forms of the given weight and level this subgroup.

INPUT:

- `weight` – an integer ≥ 2 (default: 2)

EXAMPLES:

```
sage: Gamma0(11).sturm_bound(2)
2
sage: Gamma0(389).sturm_bound(2)
65
sage: Gamma0(1).sturm_bound(12)
1
sage: Gamma0(100).sturm_bound(2)
30
sage: Gamma0(1).sturm_bound(36)
3
sage: Gamma0(11).sturm_bound()
2
sage: Gamma0(13).sturm_bound()
3
sage: Gamma0(16).sturm_bound()
4
sage: GammaH(16,[13]).sturm_bound()
8
sage: GammaH(16,[15]).sturm_bound()
16
sage: Gamma1(16).sturm_bound()
32
sage: Gamma1(13).sturm_bound()
28
sage: Gamma1(13).sturm_bound(5)
70
```

```
>>> from sage.all import *
>>> Gamma0(Integer(11)).sturm_bound(Integer(2))
2
>>> Gamma0(Integer(389)).sturm_bound(Integer(2))
65
>>> Gamma0(Integer(1)).sturm_bound(Integer(12))
1
>>> Gamma0(Integer(100)).sturm_bound(Integer(2))
30
>>> Gamma0(Integer(1)).sturm_bound(Integer(36))
3
>>> Gamma0(Integer(11)).sturm_bound()
2
>>> Gamma0(Integer(13)).sturm_bound()
3
>>> Gamma0(Integer(16)).sturm_bound()
4
>>> GammaH(Integer(16),[Integer(13)]).sturm_bound()
```

(continues on next page)

(continued from previous page)

```
8
>>> GammaH(Integer(16),[Integer(15)]).sturm_bound()
16
>>> Gamma1(Integer(16)).sturm_bound()
32
>>> Gamma1(Integer(13)).sturm_bound()
28
>>> Gamma1(Integer(13)).sturm_bound(Integer(5))
70
```

FURTHER DETAILS: This function returns a positive integer n such that the Hecke operators $T_1, \ldots, T_n$ acting on *cusp forms* generate the Hecke algebra as a $\mathbf{Z}$-module when the character is trivial or quadratic. Otherwise, $T_1, \ldots, T_n$ generate the Hecke algebra at least as a $\mathbf{Z}[\varepsilon]$-module, where $\mathbf{Z}[\varepsilon]$ is the ring generated by the values of the Dirichlet character ε. Alternatively, this is a bound such that if two cusp forms associated to this space of modular symbols are congruent modulo (λ, q^n), then they are congruent modulo λ.

REFERENCES:

- See the Agashe-Stein appendix to Lario and Schoof, *Some computations with Hecke rings and deformation rings*, Experimental Math., 11 (2002), no. 2, 303-311.
- This result originated in the paper Sturm, *On the congruence of modular forms*, Springer LNM 1240, 275-280, 1987.

REMARK: Kevin Buzzard pointed out to me (William Stein) in Fall 2002 that the above bound is fine for $\Gamma_1(N)$ with character, as one sees by taking a power of f. More precisely, if $f \cong 0 \pmod{p}$ for first s coefficients, then $f^r \cong 0 \pmod{p}$ for first sr coefficients. Since the weight of f^r is $r \cdot k(f)$, it follows that if $s \geq b$, where b is the Sturm bound for $\Gamma_0(N)$ at weight $k(f)$, then f^r has valuation large enough to be forced to be 0 at $r * k(f)$ by Sturm bound (which is valid if we choose r correctly). Thus $f \cong 0 \pmod{p}$. Conclusion: For $\Gamma_1(N)$ with fixed character, the Sturm bound is *exactly* the same as for $\Gamma_0(N)$.

A key point is that we are finding $\mathbf{Z}[\varepsilon]$ generators for the Hecke algebra here, not $\mathbf{Z}$-generators. So if one wants generators for the Hecke algebra over $\mathbf{Z}$, this bound must be suitably modified (and I'm not sure what the modification is).

AUTHORS:

- William Stein

to_even_subgroup()

Return the smallest even subgroup of $SL(2, \mathbf{Z})$ containing self.

EXAMPLES:

```
sage: sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().to_even_
↪subgroup()
Traceback (most recent call last):
...
NotImplementedError: Please implement _contains_sl2 for <class 'sage.modular.
↪arithgroup.arithgroup_generic.ArithmeticSubgroup_with_category'>
```

```
>>> from sage.all import *
>>> sage.modular.arithgroup.arithgroup_generic.ArithmeticSubgroup().to_even_
↪subgroup()
Traceback (most recent call last):
...
NotImplementedError: Please implement _contains_sl2 for <class 'sage.modular.
↪arithgroup.arithgroup_generic.ArithmeticSubgroup_with_category'>
```

todd_coxeter(*G=None*, *on_right=True*)

Compute coset representatives for self \ G and action of standard generators on them via Todd-Coxeter enumeration.

If `G` is `None`, default to `SL2Z`. The method also computes generators of the subgroup at same time.

INPUT:

- `G` – intermediate subgroup (currently not implemented if different from SL(2,Z))
- `on_right` – boolean (default: `True`); if True return right coset enumeration, if False return left one.

This is *extremely* slow in general.

OUTPUT:

- a list of coset representatives
- a list of generators for the group
- `l` – list of integers that correspond to the action of the standard parabolic element [[1,1],[0,1]] of $SL(2,\mathbf{Z})$ on the cosets of self.
- `s` – list of integers that correspond to the action of the standard element of order 2 [[0,-1],[1,0]] on the cosets of self.

EXAMPLES:

```
sage: L = SL2Z([1,1,0,1])
sage: S = SL2Z([0,-1,1,0])

sage: G = Gamma(2)
sage: reps, gens, l, s = G.todd_coxeter()
sage: len(reps) == G.index()
True
sage: all(reps[i] * L * ~reps[l[i]] in G for i in range(6))
True
sage: all(reps[i] * S * ~reps[s[i]] in G for i in range(6))
True

sage: G = Gamma0(7)
sage: reps, gens, l, s = G.todd_coxeter()
sage: len(reps) == G.index()
True
sage: all(reps[i] * L * ~reps[l[i]] in G for i in range(8))
True
sage: all(reps[i] * S * ~reps[s[i]] in G for i in range(8))
True

sage: G = Gamma1(3)
sage: reps, gens, l, s = G.todd_coxeter(on_right=False)
sage: len(reps) == G.index()
True
sage: all(~reps[l[i]] * L * reps[i] in G for i in range(8))
True
sage: all(~reps[s[i]] * S * reps[i] in G for i in range(8))
True

sage: G = Gamma0(5)
sage: reps, gens, l, s = G.todd_coxeter(on_right=False)
sage: len(reps) == G.index()
```

(continues on next page)

(continued from previous page)

```
True
sage: all(~reps[l[i]] * L * reps[i] in G for i in range(6))
True
sage: all(~reps[s[i]] * S * reps[i] in G for i in range(6))
True
```

```
>>> from sage.all import *
>>> L = SL2Z([Integer(1),Integer(1),Integer(0),Integer(1)])
>>> S = SL2Z([Integer(0),-Integer(1),Integer(1),Integer(0)])

>>> G = Gamma(Integer(2))
>>> reps, gens, l, s = G.todd_coxeter()
>>> len(reps) == G.index()
True
>>> all(reps[i] * L * ~reps[l[i]] in G for i in range(Integer(6)))
True
>>> all(reps[i] * S * ~reps[s[i]] in G for i in range(Integer(6)))
True

>>> G = Gamma0(Integer(7))
>>> reps, gens, l, s = G.todd_coxeter()
>>> len(reps) == G.index()
True
>>> all(reps[i] * L * ~reps[l[i]] in G for i in range(Integer(8)))
True
>>> all(reps[i] * S * ~reps[s[i]] in G for i in range(Integer(8)))
True

>>> G = Gamma1(Integer(3))
>>> reps, gens, l, s = G.todd_coxeter(on_right=False)
>>> len(reps) == G.index()
True
>>> all(~reps[l[i]] * L * reps[i] in G for i in range(Integer(8)))
True
>>> all(~reps[s[i]] * S * reps[i] in G for i in range(Integer(8)))
True

>>> G = Gamma0(Integer(5))
>>> reps, gens, l, s = G.todd_coxeter(on_right=False)
>>> len(reps) == G.index()
True
>>> all(~reps[l[i]] * L * reps[i] in G for i in range(Integer(6)))
True
>>> all(~reps[s[i]] * S * reps[i] in G for i in range(Integer(6)))
True
```

`sage.modular.arithgroup.arithgroup_generic.`**`is_ArithmeticSubgroup`**`(x)`

Return `True` if `x` is of type *`ArithmeticSubgroup`*.

EXAMPLES:

```
sage: from sage.modular.arithgroup.all import is_ArithmeticSubgroup
sage: is_ArithmeticSubgroup(GL(2, GF(7)))
doctest:warning...
DeprecationWarning: The function is_ArithmeticSubgroup is deprecated; use
↪'isinstance(..., ArithmeticSubgroup)' instead.
```

(continues on next page)

(continued from previous page)

```
See https://github.com/sagemath/sage/issues/38035 for details.
False
sage: is_ArithmeticSubgroup(Gamma0(4))
True
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.all import is_ArithmeticSubgroup
>>> is_ArithmeticSubgroup(GL(Integer(2), GF(Integer(7))))
doctest:warning...
DeprecationWarning: The function is_ArithmeticSubgroup is deprecated; use
↪'isinstance(..., ArithmeticSubgroup)' instead.
See https://github.com/sagemath/sage/issues/38035 for details.
False
>>> is_ArithmeticSubgroup(Gamma0(Integer(4)))
True
```

CHAPTER

TWO

ARITHMETIC SUBGROUPS DEFINED BY PERMUTATIONS OF COSETS

A subgroup of finite index H of a finitely generated group G is completely described by the action of a set of generators of G on the right cosets $H\backslash G = \{Hg\}_{g\in G}$. After some arbitrary choice of numbering one can identify the action of generators as elements of a symmetric group acting transitively (and satisfying the relations of the relators in G). As $\mathrm{SL}_2(\mathbf{Z})$ has a very simple presentation as a central extension of a free product of cyclic groups, one can easily design algorithms from this point of view.

The generators of $\mathrm{SL}_2(\mathbf{Z})$ used in this module are named as follows s_2, s_3, l, r which are defined by

$$s_2 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \quad s_3 = \begin{pmatrix} 0 & 1 \\ -1 & 1 \end{pmatrix}, \quad l = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad r = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Those generators satisfy the following relations

$$s_2^2 = s_3^3 = -1, \quad r = s_2^{-1}\, l^{-1}\, s_2.$$

In particular not all four are needed to generate the whole group $\mathrm{SL}_2(\mathbf{Z})$. Three couples which generate $\mathrm{SL}_2(\mathbf{Z})$ are of particular interest:

- (l, r) as they are also semigroup generators for the semigroup of matrices in $\mathrm{SL}_2(\mathbf{Z})$ with non-negative entries,
- (l, s_2) as they are closely related to the continued fraction algorithm,
- (s_2, s_3) as the group $\mathrm{PSL}_2(\mathbf{Z})$ is the free product of the finite cyclic groups generated by these two elements.

Part of these functions are based on Chris Kurth's *KFarey* package [Kur2008]. For tests see the file `sage.modular.arithgroup.tests`.

REFERENCES:

- [ASD1971]
- [Gor2009]
- [HL2014]
- [Hsu1996]
- [Hsu1997]
- [KSV2011]
- [Kul1991]
- [Kur2008]
- [KL2008]
- [Ver]

Todo:

- modular Farey symbols
- computation of generators of a modular subgroup with a standard surface group presentation. In other words, compute a presentation of the form

$$\langle x_i, y_i, c_j | \prod_i [x_i, y_i] \prod_j c_j^{\nu_j} = 1 \rangle$$

  where the elements x_i and y_i are hyperbolic and c_j are parabolic ($\nu_j = \infty$) or elliptic elements ($\nu_j < \infty$).
- computation of centralizer.
- generation of modular (even) subgroups of fixed index.

AUTHORS:

- Chris Kurth (2008): created KFarey package
- David Loeffler (2009): adapted functions from KFarey for inclusion into Sage
- Vincent Delecroix (2010): implementation for odd groups, new design, improvements, documentation
- David Loeffler (2011): congruence testing for odd subgroups, enumeration of liftings of projective subgroups
- David Loeffler & Thomas Hamilton (2012): generalised Hsu congruence test for odd subgroups

`sage.modular.arithgroup.arithgroup_perm.`**`ArithmeticSubgroup_Permutation`**(*L=None*, *R=None*, *S2=None*, *S3=None*, *relabel=False*, *check=True*)

Construct a subgroup of $\mathrm{SL}_2(\mathbf{Z})$ from the action of generators on its right cosets.

Return an arithmetic subgroup knowing the action, given by permutations, of at least two standard generators on the its cosets. The generators considered are the following matrices:

$$s_2 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \quad s_3 = \begin{pmatrix} 0 & 1 \\ -1 & 1 \end{pmatrix}, \quad l = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad r = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

An error will be raised if only one permutation is given. If no arguments are given at all, the full modular group $\mathrm{SL}(2, \mathbf{Z})$ is returned.

INPUT:

- `S2`, `S3`, `L`, `R` – permutations; action of matrices on the right cosets (each coset is identified to an element of $\{1, \ldots, n\}$ where 1 is reserved for the identity coset).
- `relabel` – boolean (default: `False`); if True, renumber the cosets in a canonical way.
- `check` – boolean (default: `True`); check that the input is valid (it may be time efficient but less safe to set it to False)

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2)(3,4)",S3="(1,2,3)")
sage: G
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,4)
 S3=(1,2,3)
 L=(1,4,3)
 R=(2,4,3)
sage: G.index()
4

sage: G = ArithmeticSubgroup_Permutation(); G
Arithmetic subgroup with permutations of right cosets
 S2=()
 S3=()
 L=()
 R=()
sage: G == SL2Z
True
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2)(3,4)",S3="(1,2,3)")
>>> G
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,4)
 S3=(1,2,3)
 L=(1,4,3)
 R=(2,4,3)
>>> G.index()
4

>>> G = ArithmeticSubgroup_Permutation(); G
Arithmetic subgroup with permutations of right cosets
 S2=()
 S3=()
 L=()
 R=()
>>> G == SL2Z
True
```

Some invalid inputs:

```
sage: ArithmeticSubgroup_Permutation(S2="(1,2)")
Traceback (most recent call last):
...
ValueError: Need at least two generators
sage: ArithmeticSubgroup_Permutation(S2="(1,2)",S3="(3,4,5)")
Traceback (most recent call last):
...
ValueError: Permutations do not generate a transitive group
sage: ArithmeticSubgroup_Permutation(L="(1,2)",R="(1,2,3)")
Traceback (most recent call last):
...
ValueError: Wrong relations between generators
sage: ArithmeticSubgroup_Permutation(S2="(1,2,3,4)",S3="()")
Traceback (most recent call last):
...
ValueError: S2^2 does not equal to S3^3
```

(continues on next page)

(continued from previous page)

```
sage: ArithmeticSubgroup_Permutation(S2="(1,4,2,5,3)", S3="(1,3,5,2,4)")
Traceback (most recent call last):
...
ValueError: S2^2 = S3^3 must have order 1 or 2
```

```
>>> from sage.all import *
>>> ArithmeticSubgroup_Permutation(S2="(1,2)")
Traceback (most recent call last):
...
ValueError: Need at least two generators
>>> ArithmeticSubgroup_Permutation(S2="(1,2)",S3="(3,4,5)")
Traceback (most recent call last):
...
ValueError: Permutations do not generate a transitive group
>>> ArithmeticSubgroup_Permutation(L="(1,2)",R="(1,2,3)")
Traceback (most recent call last):
...
ValueError: Wrong relations between generators
>>> ArithmeticSubgroup_Permutation(S2="(1,2,3,4)",S3="()")
Traceback (most recent call last):
...
ValueError: S2^2 does not equal to S3^3
>>> ArithmeticSubgroup_Permutation(S2="(1,4,2,5,3)", S3="(1,3,5,2,4)")
Traceback (most recent call last):
...
ValueError: S2^2 = S3^3 must have order 1 or 2
```

The input checks can be disabled for speed:

```
sage: ArithmeticSubgroup_Permutation(S2="(1,2)",S3="(3,4,5)", check=False) # don
↪'t do this!
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)
 S3=(3,4,5)
 L=(1,2)(3,5,4)
 R=(1,2)(3,4,5)
```

```
>>> from sage.all import *
>>> ArithmeticSubgroup_Permutation(S2="(1,2)",S3="(3,4,5)", check=False) # don't␣
↪do this!
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)
 S3=(3,4,5)
 L=(1,2)(3,5,4)
 R=(1,2)(3,4,5)
```

class sage.modular.arithgroup.arithgroup_perm.**ArithmeticSubgroup_Permutation_class**

Bases: *ArithmeticSubgroup*

A subgroup of $\mathrm{SL}_2(\mathbf{Z})$ defined by the action of generators on its coset graph.

The class stores the action of generators s_2, s_3, l, r on right cosets Hg of a finite index subgroup $H < \mathrm{SL}_2(\mathbf{Z})$. In particular the action of $\mathrm{SL}_2(\mathbf{Z})$ on the cosets is on right.

$$s_2 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \quad s_3 = \begin{pmatrix} 0 & 1 \\ -1 & 1 \end{pmatrix}, \quad l = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad r = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

L()

Return the action of the matrix l as a permutation of cosets.

$$l = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2)",S3="(1,2,3)")
sage: G.L()
(1,3)
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2)",S3="(1,2,3)")
>>> G.L()
(1,3)
```

R()

Return the action of the matrix r as a permutation of cosets.

$$r = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2)",S3="(1,2,3)")
sage: G.R()
(2,3)
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2)",S3="(1,2,3)")
>>> G.R()
(2,3)
```

S2()

Return the action of the matrix s_2 as a permutation of cosets.

$$s_2 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2)",S3="(1,2,3)")
sage: G.S2()
(1,2)
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2)",S3="(1,2,3)")
>>> G.S2()
(1,2)
```

S3()

Return the action of the matrix s_3 as a permutation of cosets.

$$s_3 = \begin{pmatrix} 0 & 1 \\ -1 & 1 \end{pmatrix},$$

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2)",S3="(1,2,3)")
sage: G.S3()
(1,2,3)
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2)",S3="(1,2,3)")
>>> G.S3()
(1,2,3)
```

congruence_closure()

Returns the smallest congruence subgroup containing self. If self is congruence, this is just self, but represented as a congruence subgroup data type. If self is not congruence, then it may be larger.

In practice, we use the following criterion: let m be the generalised level of self. If this subgroup is even, let $n = m$, else let $n = 2m$. Then any congruence subgroup containing self contains $\Gamma(n)$ (a generalisation of Wohlfahrt's theorem due to Kiming, Verrill and Schuett). So we compute the image of self modulo n and return the preimage of that.

Note: If you just want to know if the subgroup is congruence or not, it is *much* faster to use `is_congruence()`.

EXAMPLES:

```
sage: Gamma1(3).as_permutation_group().congruence_closure()
Congruence subgroup of SL(2,Z) of level 3, preimage of:
 Matrix group over Ring of integers modulo 3 with 2 generators (
[1 1]  [1 2]
[0 1], [0 1]
)
sage: sage.modular.arithgroup.arithgroup_perm.HsuExample10().congruence_
↪closure()  # long time (11s on sage.math, 2012)
Modular Group SL(2,Z)
```

```
>>> from sage.all import *
>>> Gamma1(Integer(3)).as_permutation_group().congruence_closure()
Congruence subgroup of SL(2,Z) of level 3, preimage of:
 Matrix group over Ring of integers modulo 3 with 2 generators (
[1 1]  [1 2]
[0 1], [0 1]
)
>>> sage.modular.arithgroup.arithgroup_perm.HsuExample10().congruence_
↪closure()  # long time (11s on sage.math, 2012)
Modular Group SL(2,Z)
```

coset_graph(*right_cosets=False*, *s2_edges=True*, *s3_edges=True*, *l_edges=False*, *r_edges=False*, *s2_label='s2'*, *s3_label='s3'*, *l_label='l'*, *r_label='r'*)

Return the right (or left) coset graph.

INPUT:

- `right_cosets` – bool (default: `False`); right or left coset graph
- `s2_edges` – bool (default: `True`); put edges associated to s2
- `s3_edges` – bool (default: `True`); put edges associated to s3
- `l_edges` – bool (default: `False`); put edges associated to l

- `r_edges` – bool (default: `False`); put edges associated to r
- `s2_label`, `s3_label`, `l_label`, `r_label` – the labels to put on the edges corresponding to the generators action. Use `None` for no label.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2)",S3="()")
sage: G
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)
 S3=()
 L=(1,2)
 R=(1,2)
sage: G.index()
2
sage: G.coset_graph()
Looped multi-digraph on 2 vertices
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2)",S3="()")
>>> G
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)
 S3=()
 L=(1,2)
 R=(1,2)
>>> G.index()
2
>>> G.coset_graph()
Looped multi-digraph on 2 vertices
```

generalised_level()

Return the generalised level of this subgroup.

The *generalised level* of a subgroup of the modular group is the least common multiple of the widths of the cusps. It was proven by Wohlfart that for even congruence subgroups, the (conventional) level coincides with the generalised level. For odd congruence subgroups the level is either the generalised level, or twice the generalised level [KSV2011].

EXAMPLES:

```
sage: G = Gamma(2).as_permutation_group()
sage: G.generalised_level()
2
sage: G = Gamma0(3).as_permutation_group()
sage: G.generalised_level()
3
```

```
>>> from sage.all import *
>>> G = Gamma(Integer(2)).as_permutation_group()
>>> G.generalised_level()
2
>>> G = Gamma0(Integer(3)).as_permutation_group()
>>> G.generalised_level()
3
```

index()

Returns the index of this modular subgroup in the full modular group.

EXAMPLES:

```
sage: G = Gamma(2)
sage: P = G.as_permutation_group()
sage: P.index()
6
sage: G.index() == P.index()
True

sage: G = Gamma0(8)
sage: P = G.as_permutation_group()
sage: P.index()
12
sage: G.index() == P.index()
True

sage: G = Gamma1(6)
sage: P = G.as_permutation_group()
sage: P.index()
24
sage: G.index() == P.index()
True
```

```
>>> from sage.all import *
>>> G = Gamma(Integer(2))
>>> P = G.as_permutation_group()
>>> P.index()
6
>>> G.index() == P.index()
True

>>> G = Gamma0(Integer(8))
>>> P = G.as_permutation_group()
>>> P.index()
12
>>> G.index() == P.index()
True

>>> G = Gamma1(Integer(6))
>>> P = G.as_permutation_group()
>>> P.index()
24
>>> G.index() == P.index()
True
```

is_congruence()

Return `True` if this is a congruence subgroup, and `False` otherwise.

ALGORITHM:

Uses Hsu's algorithm [Hsu1996]. Adapted from Chris Kurth's implementation in KFarey [Kur2008].

For *odd* subgroups, Hsu's algorithm still works with minor modifications, using the extension of Wohlfarht's theorem due to Kiming, Schuett and Verrill [KSV2011]. See [HL2014] for details.

The algorithm is as follows. Let G be a finite-index subgroup of $\mathrm{SL}(2, \mathbf{Z})$, and let L and R be the permutations of the cosets of G given by the elements $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ and $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$. Let N be the generalized level of G (if G is even) or twice the generalized level (if G is odd). Then:

- if N is odd, G is congruence if and only if the relation

$$(LR^{-1}L)^2 = (R^2L^{1/2})^3$$

holds, where $1/2$ is understood as the multiplicative inverse of 2 modulo N.

- if N is a power of 2, then G is congruence if and only if the relations

$$\begin{array}{rl} (LR^{-1}L)^{-1}S(LR^{-1}L)S = 1 & (A1) \\ S^{-1}RS = R^{25} & (A2) \\ (LR^{-1}L)^2 = (SR^5LR^{-1}L)^3 & (A3) \end{array}$$

hold, where $S = L^{20}R^{1/5}L^{-4}R^{-1}$, $1/5$ being the inverse of 5 modulo N.

- if N is neither odd nor a power of 2, seven relations (B1-7) hold, for which see [HL2014], or the source code of this function.

If the Sage verbosity flag is set (using `set_verbose()`), then extra output will be produced indicating which of the relations (A1-3) or (B1-7) is not satisfied.

EXAMPLES:

Test if $\mathrm{SL}_2(\mathbf{Z})$ is congruence:

```
sage: a = ArithmeticSubgroup_Permutation(L='',R='')
sage: a.index()
1
sage: a.is_congruence()
True
```

```
>>> from sage.all import *
>>> a = ArithmeticSubgroup_Permutation(L='',R='')
>>> a.index()
1
>>> a.is_congruence()
True
```

This example is congruence – it is $\Gamma_0(3)$ in disguise:

```
sage: S2 = SymmetricGroup(4)
sage: l = S2((2,3,4))
sage: r = S2((1,3,4))
sage: G = ArithmeticSubgroup_Permutation(L=l,R=r)
sage: G
Arithmetic subgroup with permutations of right cosets
S2=(1,2)(3,4)
S3=(1,4,2)
L=(2,3,4)
R=(1,3,4)
sage: G.is_congruence()
True
```

```
>>> from sage.all import *
>>> S2 = SymmetricGroup(Integer(4))
>>> l = S2((Integer(2),Integer(3),Integer(4)))
>>> r = S2((Integer(1),Integer(3),Integer(4)))
>>> G = ArithmeticSubgroup_Permutation(L=l,R=r)
>>> G
Arithmetic subgroup with permutations of right cosets
```

(continues on next page)

(continued from previous page)

```
S2=(1,2)(3,4)
S3=(1,4,2)
L=(2,3,4)
R=(1,3,4)
>>> G.is_congruence()
True
```

This one is noncongruence:

```
sage: import sage.modular.arithgroup.arithgroup_perm as ap
sage: ap.HsuExample10().is_congruence()
False
```

```
>>> from sage.all import *
>>> import sage.modular.arithgroup.arithgroup_perm as ap
>>> ap.HsuExample10().is_congruence()
False
```

The following example (taken from [KSV2011]) shows that a lifting of a congruence subgroup of $\mathrm{PSL}(2,\mathbf{Z})$ to a subgroup of $\mathrm{SL}(2,\mathbf{Z})$ need not necessarily be congruence:

```
sage: S2 = "(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,
↪23)"
sage: S3 = "(1,14,15,13,2,3)(4,5,6,16,17,18)(7,8,9,19,20,21)(10,11,12,22,23,
↪24)"
sage: G = ArithmeticSubgroup_Permutation(S2=S2,S3=S3)
sage: G.is_congruence()
False
sage: G.to_even_subgroup().is_congruence()
True
```

```
>>> from sage.all import *
>>> S2 = "(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,
↪23)"
>>> S3 = "(1,14,15,13,2,3)(4,5,6,16,17,18)(7,8,9,19,20,21)(10,11,12,22,23,24)"
>>> G = ArithmeticSubgroup_Permutation(S2=S2,S3=S3)
>>> G.is_congruence()
False
>>> G.to_even_subgroup().is_congruence()
True
```

In fact G is a lifting to $\mathrm{SL}(2,\mathbf{Z})$ of the group $\bar{\Gamma}_0(6)$:

```
sage: G.to_even_subgroup() == Gamma0(6)
True
```

```
>>> from sage.all import *
>>> G.to_even_subgroup() == Gamma0(Integer(6))
True
```

is_normal()

Test whether the group is normal

EXAMPLES:

```
sage: G = Gamma(2).as_permutation_group()
sage: G.is_normal()
True

sage: G = Gamma1(2).as_permutation_group()
sage: G.is_normal()
False
```

```
>>> from sage.all import *
>>> G = Gamma(Integer(2)).as_permutation_group()
>>> G.is_normal()
True

>>> G = Gamma1(Integer(2)).as_permutation_group()
>>> G.is_normal()
False
```

perm_group()

Return the underlying permutation group.

The permutation group returned is isomorphic to the action of the generators s_2 (element of order two), s_3 (element of order 3), l (parabolic element) and r (parabolic element) on right cosets (the action is on the right).

EXAMPLES:

```
sage: import sage.modular.arithgroup.arithgroup_perm as ap
sage: ap.HsuExample10().perm_group()
Permutation Group with generators [(1,2)(3,4)(5,6)(7,8)(9,10), (1,8,3)(2,4,
↪6)(5,7,10), (1,4)(2,5,9,10,8)(3,7,6), (1,7,9,10,6)(2,3)(4,5,8)]
```

```
>>> from sage.all import *
>>> import sage.modular.arithgroup.arithgroup_perm as ap
>>> ap.HsuExample10().perm_group()
Permutation Group with generators [(1,2)(3,4)(5,6)(7,8)(9,10), (1,8,3)(2,4,
↪6)(5,7,10), (1,4)(2,5,9,10,8)(3,7,6), (1,7,9,10,6)(2,3)(4,5,8)]
```

permutation_action(*x*)

Given an element x of $\mathrm{SL}_2(\mathbf{Z})$, compute the permutation of the cosets of self given by right multiplication by x.

EXAMPLES:

```
sage: import sage.modular.arithgroup.arithgroup_perm as ap
sage: ap.HsuExample10().permutation_action(SL2Z([32, -21, -67, 44]))
(1,4,6,2,10,5,3,7,8,9)
```

```
>>> from sage.all import *
>>> import sage.modular.arithgroup.arithgroup_perm as ap
>>> ap.HsuExample10().permutation_action(SL2Z([Integer(32), -Integer(21), -
↪Integer(67), Integer(44)]))
(1,4,6,2,10,5,3,7,8,9)
```

random_element(*initial_steps=30*)

Returns a random element in this subgroup.

The algorithm uses a random walk on the Cayley graph of $\mathrm{SL}_2(\mathbf{Z})$ stopped at the first time it reaches the subgroup after at least `initial_steps` steps.

INPUT:

- `initial_steps` – positive integer (default: 30)

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2='(1,3)(4,5)',S3='(1,2,5)(3,4,6)')
sage: all(G.random_element() in G for _ in range(10))
True
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2='(1,3)(4,5)',S3='(1,2,5)(3,4,6)')
>>> all(G.random_element() in G for _ in range(Integer(10)))
True
```

relabel(*inplace=True*)

Relabel the cosets of this modular subgroup in a canonical way.

The implementation of modular subgroup by action of generators on cosets depends on the choice of a numbering. This function provides canonical labels in the sense that two equal subgroups with different labels are relabeled the same way. The default implementation relabels the group itself. If you want to get a relabeled copy of your modular subgroup, put to `False` the option `inplace`.

ALGORITHM:

We give an overview of how the canonical labels for the modular subgroup are built. The procedure only uses the permutations $S3$ and $S2$ that define the modular subgroup and can be used to renumber any transitive action of the symmetric group. In other words, the algorithm construct a canonical representative of a transitive subgroup in its conjugacy class in the full symmetric group.

1. The identity is still numbered 0 and set the current vertex to be the identity.

2. Number the cycle of $S3$ the current vertex belongs to: if the current vertex is labeled n then the numbering in such way that the cycle becomes $(n, n+1, \ldots, n+k)$).

3. Find a new current vertex using the permutation $S2$. If all vertices are relabeled then it's done, otherwise go to step 2.

EXAMPLES:

```
sage: S2 = "(1,2)(3,4)(5,6)"; S3 = "(1,2,3)(4,5,6)"
sage: G1 = ArithmeticSubgroup_Permutation(S2=S2,S3=S3); G1
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,4)(5,6)
 S3=(1,2,3)(4,5,6)
 L=(1,4,5,3)
 R=(2,4,6,3)
sage: G1.relabel(); G1
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,4)(5,6)
 S3=(1,2,3)(4,5,6)
 L=(1,4,5,3)
 R=(2,4,6,3)

sage: S2 = "(1,2)(3,5)(4,6)"; S3 = "(1,2,3)(4,5,6)"
sage: G2 = ArithmeticSubgroup_Permutation(S2=S2,S3=S3); G2
Arithmetic subgroup with permutations of right cosets
```

(continues on next page)

(continued from previous page)

```
 S2=(1,2)(3,5)(4,6)
 S3=(1,2,3)(4,5,6)
 L=(1,5,6,3)
 R=(2,5,4,3)
sage: G2.relabel(); G2
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,4)(5,6)
 S3=(1,2,3)(4,5,6)
 L=(1,4,5,3)
 R=(2,4,6,3)

sage: S2 = "(1,2)(3,6)(4,5)"; S3 = "(1,2,3)(4,5,6)"
sage: G3 = ArithmeticSubgroup_Permutation(S2=S2,S3=S3); G3
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,6)(4,5)
 S3=(1,2,3)(4,5,6)
 L=(1,6,4,3)
 R=(2,6,5,3)
sage: G4 = G3.relabel(inplace=False)
sage: G4
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,4)(5,6)
 S3=(1,2,3)(4,5,6)
 L=(1,4,5,3)
 R=(2,4,6,3)
sage: G3 is G4
False
```

```
>>> from sage.all import *
>>> S2 = "(1,2)(3,4)(5,6)"; S3 = "(1,2,3)(4,5,6)"
>>> G1 = ArithmeticSubgroup_Permutation(S2=S2,S3=S3); G1
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,4)(5,6)
 S3=(1,2,3)(4,5,6)
 L=(1,4,5,3)
 R=(2,4,6,3)
>>> G1.relabel(); G1
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,4)(5,6)
 S3=(1,2,3)(4,5,6)
 L=(1,4,5,3)
 R=(2,4,6,3)

>>> S2 = "(1,2)(3,5)(4,6)"; S3 = "(1,2,3)(4,5,6)"
>>> G2 = ArithmeticSubgroup_Permutation(S2=S2,S3=S3); G2
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,5)(4,6)
 S3=(1,2,3)(4,5,6)
 L=(1,5,6,3)
 R=(2,5,4,3)
>>> G2.relabel(); G2
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,4)(5,6)
 S3=(1,2,3)(4,5,6)
 L=(1,4,5,3)
 R=(2,4,6,3)
```

(continues on next page)

(continued from previous page)

```
>>> S2 = "(1,2)(3,6)(4,5)"; S3 = "(1,2,3)(4,5,6)"
>>> G3 = ArithmeticSubgroup_Permutation(S2=S2,S3=S3); G3
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,6)(4,5)
 S3=(1,2,3)(4,5,6)
 L=(1,6,4,3)
 R=(2,6,5,3)
>>> G4 = G3.relabel(inplace=False)
>>> G4
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,4)(5,6)
 S3=(1,2,3)(4,5,6)
 L=(1,4,5,3)
 R=(2,4,6,3)
>>> G3 is G4
False
```

surgroups()

Return an iterator through the non-trivial intermediate groups between $SL(2,\mathbf{Z})$ and this finite index group.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2)(3,4)(5,6)", S3="(1,2,3)(4,
↪5,6)")
sage: H = next(G.surgroups())
sage: H
 Arithmetic subgroup with permutations of right cosets
 S2=(1,2)
 S3=(1,2,3)
 L=(1,3)
 R=(2,3)
sage: G.is_subgroup(H)
True
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2)(3,4)(5,6)", S3="(1,2,3)(4,5,
↪6)")
>>> H = next(G.surgroups())
>>> H
 Arithmetic subgroup with permutations of right cosets
 S2=(1,2)
 S3=(1,2,3)
 L=(1,3)
 R=(2,3)
>>> G.is_subgroup(H)
True
```

The principal congruence group $\Gamma(3)$ has thirteen surgroups:

```
sage: G = Gamma(3).as_permutation_group()
sage: G.index()
24
sage: l = []
sage: for H in G.surgroups():
....:     l.append(H.index())
```

(continues on next page)

(continued from previous page)

```
....:     assert G.is_subgroup(H) and H.is_congruence()
sage: l
[6, 3, 4, 8, 4, 8, 4, 12, 4, 6, 6, 8, 8]
```

```
>>> from sage.all import *
>>> G = Gamma(Integer(3)).as_permutation_group()
>>> G.index()
24
>>> l = []
>>> for H in G.surgroups():
...     l.append(H.index())
...     assert G.is_subgroup(H) and H.is_congruence()
>>> l
[6, 3, 4, 8, 4, 8, 4, 12, 4, 6, 6, 8, 8]
```

class sage.modular.arithgroup.arithgroup_perm.**EvenArithmeticSubgroup_Permutation**(*S2*, *S3*, *L*, *R*, *canonical_labels=False*)

Bases: *ArithmeticSubgroup_Permutation_class*

An arithmetic subgroup of $\mathrm{SL}(2, \mathbf{Z})$ containing -1, represented in terms of the right action of $\mathrm{SL}(2, \mathbf{Z})$ on its cosets.

EXAMPLES:

Construct a noncongruence subgroup of index 7 (the smallest possible):

```
sage: a2 = SymmetricGroup(7)([(1,2),(3,4),(6,7)]); a3 = SymmetricGroup(7)([(1,2,
↪3),(4,5,6)])
sage: G = ArithmeticSubgroup_Permutation(S2=a2, S3=a3); G
Arithmetic subgroup with permutations of right cosets
S2=(1,2)(3,4)(6,7)
S3=(1,2,3)(4,5,6)
L=(1,4,7,6,5,3)
R=(2,4,5,7,6,3)
sage: G.index()
7
sage: G.dimension_cusp_forms(4)
1
sage: G.is_congruence()
False
```

```
>>> from sage.all import *
>>> a2 = SymmetricGroup(Integer(7))([(Integer(1),Integer(2)),(Integer(3),
↪Integer(4)),(Integer(6),Integer(7))]); a3 =␣
↪SymmetricGroup(Integer(7))([(Integer(1),Integer(2),Integer(3)),(Integer(4),
↪Integer(5),Integer(6))])
>>> G = ArithmeticSubgroup_Permutation(S2=a2, S3=a3); G
Arithmetic subgroup with permutations of right cosets
S2=(1,2)(3,4)(6,7)
S3=(1,2,3)(4,5,6)
```

(continues on next page)

(continued from previous page)

```
L=(1,4,7,6,5,3)
R=(2,4,5,7,6,3)
>>> G.index()
7
>>> G.dimension_cusp_forms(Integer(4))
1
>>> G.is_congruence()
False
```

Convert some standard congruence subgroups into permutation form:

```
sage: G = Gamma0(8).as_permutation_group()
sage: G.index()
12
sage: G.is_congruence()
True

sage: G = Gamma0(12).as_permutation_group()
sage: G
Arithmetic subgroup of index 24
sage: G.is_congruence()
True
```

```
>>> from sage.all import *
>>> G = Gamma0(Integer(8)).as_permutation_group()
>>> G.index()
12
>>> G.is_congruence()
True

>>> G = Gamma0(Integer(12)).as_permutation_group()
>>> G
Arithmetic subgroup of index 24
>>> G.is_congruence()
True
```

The following is the unique index 2 even subgroup of $\mathrm{SL}_2(\mathbf{Z})$:

```
sage: w = SymmetricGroup(2)([2,1])
sage: G = ArithmeticSubgroup_Permutation(L=w, R=w)
sage: G.dimension_cusp_forms(6)
1
sage: G.genus()
0
```

```
>>> from sage.all import *
>>> w = SymmetricGroup(Integer(2))([Integer(2),Integer(1)])
>>> G = ArithmeticSubgroup_Permutation(L=w, R=w)
>>> G.dimension_cusp_forms(Integer(6))
1
>>> G.genus()
0
```

coset_reps()

Return coset representatives.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2)(3,4)",S3="(1,2,3)")
sage: c = G.coset_reps()
sage: len(c)
4
sage: [g in G for g in c]
[True, False, False, False]
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2)(3,4)",S3="(1,2,3)")
>>> c = G.coset_reps()
>>> len(c)
4
>>> [g in G for g in c]
[True, False, False, False]
```

cusp_widths(*exp=False*)

Return the list of cusp widths of the group.

EXAMPLES:

```
sage: G = Gamma(2).as_permutation_group()
sage: G.cusp_widths()
[2, 2, 2]
sage: G.cusp_widths(exp=True)
{2: 3}

sage: S2 = "(1,2)(3,4)(5,6)"
sage: S3 = "(1,2,3)(4,5,6)"
sage: G = ArithmeticSubgroup_Permutation(S2=S2,S3=S3)
sage: G.cusp_widths()
[1, 1, 4]
sage: G.cusp_widths(exp=True)
{1: 2, 4: 1}

sage: S2 = "(1,2)(3,4)(5,6)"
sage: S3 = "(1,3,5)(2,4,6)"
sage: G = ArithmeticSubgroup_Permutation(S2=S2,S3=S3)
sage: G.cusp_widths()
[6]
sage: G.cusp_widths(exp=True)
{6: 1}
```

```
>>> from sage.all import *
>>> G = Gamma(Integer(2)).as_permutation_group()
>>> G.cusp_widths()
[2, 2, 2]
>>> G.cusp_widths(exp=True)
{2: 3}

>>> S2 = "(1,2)(3,4)(5,6)"
>>> S3 = "(1,2,3)(4,5,6)"
>>> G = ArithmeticSubgroup_Permutation(S2=S2,S3=S3)
>>> G.cusp_widths()
[1, 1, 4]
>>> G.cusp_widths(exp=True)
{1: 2, 4: 1}
```

(continues on next page)

(continued from previous page)

```
>>> S2 = "(1,2)(3,4)(5,6)"
>>> S3 = "(1,3,5)(2,4,6)"
>>> G = ArithmeticSubgroup_Permutation(S2=S2,S3=S3)
>>> G.cusp_widths()
[6]
>>> G.cusp_widths(exp=True)
{6: 1}
```

is_even()

Returns True if this subgroup contains the matrix $-Id$.

EXAMPLES:

```
sage: G = Gamma(2).as_permutation_group()
sage: G.is_even()
True
```

```
>>> from sage.all import *
>>> G = Gamma(Integer(2)).as_permutation_group()
>>> G.is_even()
True
```

is_odd()

Returns True if this subgroup does not contain the matrix $-Id$.

EXAMPLES:

```
sage: G = Gamma(2).as_permutation_group()
sage: G.is_odd()
False
```

```
>>> from sage.all import *
>>> G = Gamma(Integer(2)).as_permutation_group()
>>> G.is_odd()
False
```

ncusps()

Return the number of cusps of this arithmetic subgroup.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2)(3,4)(5,6)",S3="(1,2,3)(4,5,
↪6)")
sage: G.ncusps()
3
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2)(3,4)(5,6)",S3="(1,2,3)(4,5,6)
↪")
>>> G.ncusps()
3
```

nu2()

Returns the number of orbits of elliptic points of order 2 for this arithmetic subgroup.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,4)(2)(3)",S3="(1,2,3)(4)")
sage: G.nu2()
2
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,4)(2)(3)",S3="(1,2,3)(4)")
>>> G.nu2()
2
```

nu3()

Returns the number of orbits of elliptic points of order 3 for this arithmetic subgroup.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,4)(2)(3)",S3="(1,2,3)(4)")
sage: G.nu3()
1
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,4)(2)(3)",S3="(1,2,3)(4)")
>>> G.nu3()
1
```

odd_subgroups()

Return a list of the odd subgroups of index 2 in Γ, where Γ is this subgroup. (Equivalently, return the liftings of $\bar{\Gamma} \leq \mathrm{PSL}(2, \mathbf{Z})$ to $\mathrm{SL}(2, \mathbf{Z})$.) This can take rather a long time if the index of this subgroup is large.

See also:

one_odd_subgroup(), which returns just one of the odd subgroups (which is much quicker than enumerating them all).

ALGORITHM:

- If Γ has an element of order 4, then there are no index 2 odd subgroups, so return the empty set.
- If Γ has no elements of order 4, then the permutation S_2 is a combination of 2-cycles with no fixed points on $\{1, \ldots, N\}$. We construct the permutation $\tilde{S}_2$ of $\{1, \ldots, 2N\}$ which has a 4-cycle $(a, b, a + N, b + N)$ for each 2-cycle (a, b) in `S2`. Similarly, we construct a permutation $\tilde{S}_3$ which has a 6-cycle $(a, b, c, a + N, b + N, c + N)$ for each 3-cycle (a, b, c) in S_3, and a 2-cycle $(a, a + N)$ for each fixed point a of S_3.

  Then the permutations $\tilde{S}_2$ and $\tilde{S}_3$ satisfy $\tilde{S}_2^2 = \tilde{S}_3^3 = \iota$ where ι is the order 2 permutation interchanging a and $a+N$ for each a. So the subgroup corresponding to these permutations is an index 2 odd subgroup of Γ.
- The other index 2 odd subgroups of Γ are obtained from the pairs $\tilde{S}_2, \tilde{S}_3^\sigma$ where σ is an element of the group generated by the 2-cycles $(a, a + N)$.

Studying the permutations in the first example below gives a good illustration of the algorithm.

EXAMPLES:

```
sage: G = sage.modular.arithgroup.arithgroup_perm.HsuExample10()
sage: [G.S2(), G.S3()]
[(1,2)(3,4)(5,6)(7,8)(9,10), (1,8,3)(2,4,6)(5,7,10)]
sage: X = G.odd_subgroups()
sage: for u in X: print([u.S2(), u.S3()])
[(1,2,11,12)(3,4,13,14)(5,6,15,16)(7,8,17,18)(9,10,19,20), (1,8,3,11,18,13)(2,
```

(continues on next page)

(continued from previous page)

```
↪4,6,12,14,16)(5,7,10,15,17,20)(9,19)]
[(1,2,11,12)(3,4,13,14)(5,6,15,16)(7,8,17,18)(9,10,19,20), (1,18,13,11,8,3)(2,
↪4,6,12,14,16)(5,7,10,15,17,20)(9,19)]
[(1,2,11,12)(3,4,13,14)(5,6,15,16)(7,8,17,18)(9,10,19,20), (1,8,13,11,18,3)(2,
↪4,6,12,14,16)(5,7,10,15,17,20)(9,19)]
[(1,2,11,12)(3,4,13,14)(5,6,15,16)(7,8,17,18)(9,10,19,20), (1,18,3,11,8,13)(2,
↪4,6,12,14,16)(5,7,10,15,17,20)(9,19)]
```

```
>>> from sage.all import *
>>> G = sage.modular.arithgroup.arithgroup_perm.HsuExample10()
>>> [G.S2(), G.S3()]
[(1,2)(3,4)(5,6)(7,8)(9,10), (1,8,3)(2,4,6)(5,7,10)]
>>> X = G.odd_subgroups()
>>> for u in X: print([u.S2(), u.S3()])
[(1,2,11,12)(3,4,13,14)(5,6,15,16)(7,8,17,18)(9,10,19,20), (1,8,3,11,18,13)(2,
↪4,6,12,14,16)(5,7,10,15,17,20)(9,19)]
[(1,2,11,12)(3,4,13,14)(5,6,15,16)(7,8,17,18)(9,10,19,20), (1,18,13,11,8,3)(2,
↪4,6,12,14,16)(5,7,10,15,17,20)(9,19)]
[(1,2,11,12)(3,4,13,14)(5,6,15,16)(7,8,17,18)(9,10,19,20), (1,8,13,11,18,3)(2,
↪4,6,12,14,16)(5,7,10,15,17,20)(9,19)]
[(1,2,11,12)(3,4,13,14)(5,6,15,16)(7,8,17,18)(9,10,19,20), (1,18,3,11,8,13)(2,
↪4,6,12,14,16)(5,7,10,15,17,20)(9,19)]
```

A projective congruence subgroup may have noncongruence liftings, as the example of $\bar{\Gamma}_0(6)$ illustrates (see [KSV2011]):

```
sage: X = Gamma0(6).as_permutation_group().odd_subgroups(); Sequence([[u.S2(),
↪ u.S3()] for u in X],cr=True)
[
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,2,
↪3,13,14,15)(4,5,6,16,17,18)(7,8,9,19,20,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,14,
↪15,13,2,3)(4,5,6,16,17,18)(7,8,9,19,20,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,2,
↪3,13,14,15)(4,17,6,16,5,18)(7,8,9,19,20,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,14,
↪15,13,2,3)(4,17,6,16,5,18)(7,8,9,19,20,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,2,
↪3,13,14,15)(4,5,6,16,17,18)(7,20,9,19,8,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,14,
↪15,13,2,3)(4,5,6,16,17,18)(7,20,9,19,8,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,2,
↪3,13,14,15)(4,17,6,16,5,18)(7,20,9,19,8,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,14,
↪15,13,2,3)(4,17,6,16,5,18)(7,20,9,19,8,21)(10,11,12,22,23,24)]
]
sage: [u.is_congruence() for u in X]
[True, False, False, True, True, False, False, True]
```

```
>>> from sage.all import *
>>> X = Gamma0(Integer(6)).as_permutation_group().odd_subgroups();␣
↪Sequence([[u.S2(), u.S3()] for u in X],cr=True)
[
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,2,
↪3,13,14,15)(4,5,6,16,17,18)(7,8,9,19,20,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,14,
```

(continues on next page)

(continued from previous page)

```
↪15,13,2,3)(4,5,6,16,17,18)(7,8,9,19,20,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,2,
↪3,13,14,15)(4,17,6,16,5,18)(7,8,9,19,20,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,14,
↪15,13,2,3)(4,17,6,16,5,18)(7,8,9,19,20,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,2,
↪3,13,14,15)(4,5,6,16,17,18)(7,20,9,19,8,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,14,
↪15,13,2,3)(4,5,6,16,17,18)(7,20,9,19,8,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,2,
↪3,13,14,15)(4,17,6,16,5,18)(7,20,9,19,8,21)(10,11,12,22,23,24)],
[(1,3,13,15)(2,4,14,16)(5,7,17,19)(6,10,18,22)(8,12,20,24)(9,11,21,23), (1,14,
↪15,13,2,3)(4,17,6,16,5,18)(7,20,9,19,8,21)(10,11,12,22,23,24)]
]
>>> [u.is_congruence() for u in X]
[True, False, False, True, True, False, False, True]
```

one_odd_subgroup(*random=False*)

Return an odd subgroup of index 2 in Γ, where Γ is this subgroup. If the optional argument `random` is False (the default), this returns an arbitrary but consistent choice from the set of index 2 odd subgroups. If `random` is True, then it will choose one of these at random.

For details of the algorithm used, see the docstring for the related function `odd_subgroups()`, which returns a list of all the index 2 odd subgroups.

EXAMPLES:

Starting from $\Gamma(4)$ we get back $\Gamma(4)$:

```
sage: G = Gamma(4).as_permutation_group()
sage: G.is_odd(), G.index()
(True, 48)
sage: Ge = G.to_even_subgroup()
sage: Go = Ge.one_odd_subgroup()
sage: Go.is_odd(), Go.index()
(True, 48)
sage: Go == G
True
```

```
>>> from sage.all import *
>>> G = Gamma(Integer(4)).as_permutation_group()
>>> G.is_odd(), G.index()
(True, 48)
>>> Ge = G.to_even_subgroup()
>>> Go = Ge.one_odd_subgroup()
>>> Go.is_odd(), Go.index()
(True, 48)
>>> Go == G
True
```

Strating from $\Gamma(6)$ we get a different group:

```
sage: G = Gamma(6).as_permutation_group()
sage: G.is_odd(), G.index()
(True, 144)
sage: Ge = G.to_even_subgroup()
sage: Go = Ge.one_odd_subgroup()
```

(continues on next page)

(continued from previous page)

```
sage: Go.is_odd(), Go.index()
(True, 144)
sage: Go == G
False
```

```
>>> from sage.all import *
>>> G = Gamma(Integer(6)).as_permutation_group()
>>> G.is_odd(), G.index()
(True, 144)
>>> Ge = G.to_even_subgroup()
>>> Go = Ge.one_odd_subgroup()
>>> Go.is_odd(), Go.index()
(True, 144)
>>> Go == G
False
```

An error will be raised if there are no such subgroups, which occurs if and only if the group contains an element of order 4:

```
sage: Gamma0(10).as_permutation_group().one_odd_subgroup()
Traceback (most recent call last):
...
ValueError: Group contains an element of order 4, hence no index 2 odd␣
↪subgroups
```

```
>>> from sage.all import *
>>> Gamma0(Integer(10)).as_permutation_group().one_odd_subgroup()
Traceback (most recent call last):
...
ValueError: Group contains an element of order 4, hence no index 2 odd␣
↪subgroups
```

Testing randomness:

```
sage: G = Gamma(6).as_permutation_group().to_even_subgroup()
sage: G1 = G.one_odd_subgroup(random=True) # random
sage: G1.is_subgroup(G)
True
```

```
>>> from sage.all import *
>>> G = Gamma(Integer(6)).as_permutation_group().to_even_subgroup()
>>> G1 = G.one_odd_subgroup(random=True) # random
>>> G1.is_subgroup(G)
True
```

to_even_subgroup(*relabel=True*)

Return the subgroup generated by self and `-Id`. Since self is even, this is just self. Provided for compatibility.

EXAMPLES:

```
sage: G = Gamma0(4).as_permutation_group()
sage: H = G.to_even_subgroup()
sage: H == G
True
```

```
>>> from sage.all import *
>>> G = Gamma0(Integer(4)).as_permutation_group()
>>> H = G.to_even_subgroup()
>>> H == G
True
```

todd_coxeter()

Returns a 4-tuple `(coset_reps, gens, l, s2)` where `coset_reps` is a list of coset representatives of the subgroup, `gens` a list of generators, `l` and `s2` are list that corresponds to the action of the matrix $S2$ and L on the cosets.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2='(1,2)(3,4)',S3='(1,2,3)')
sage: reps,gens,l,s=G.todd_coxeter_l_s2()
sage: reps
[
[1 0]  [ 0 -1]  [1 2]  [1 1]
[0 1], [ 1  0], [0 1], [0 1]
]
sage: len(gens)
3
sage: Matrix(2, 2, [1, 3, 0, 1]) in gens
True
sage: Matrix(2, 2, [1, 0, -1, 1]) in gens
True
sage: Matrix(2, 2, [1, -3, 1, -2]) in gens or Matrix(2, 2, [2, -3, 1, -1]) in␣
↪gens
True
sage: l
[3, 1, 0, 2]
sage: s
[1, 0, 3, 2]
sage: S2 = SL2Z([0,-1,1,0])
sage: L = SL2Z([1,1,0,1])
sage: reps[0] == SL2Z([1,0,0,1])
True
sage: all(reps[i]*S2*~reps[s[i]] in G for i in range(4))
True
sage: all(reps[i]*L*~reps[l[i]] in G for i in range(4))
True
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2='(1,2)(3,4)',S3='(1,2,3)')
>>> reps,gens,l,s=G.todd_coxeter_l_s2()
>>> reps
[
[1 0]  [ 0 -1]  [1 2]  [1 1]
[0 1], [ 1  0], [0 1], [0 1]
]
>>> len(gens)
3
>>> Matrix(Integer(2), Integer(2), [Integer(1), Integer(3), Integer(0),␣
↪Integer(1)]) in gens
True
>>> Matrix(Integer(2), Integer(2), [Integer(1), Integer(0), -Integer(1),␣
↪Integer(1)]) in gens
```

(continues on next page)

(continued from previous page)

```
True
>>> Matrix(Integer(2), Integer(2), [Integer(1), -Integer(3), Integer(1), -
↪Integer(2)]) in gens or Matrix(Integer(2), Integer(2), [Integer(2), -
↪Integer(3), Integer(1), -Integer(1)]) in gens
True
>>> l
[3, 1, 0, 2]
>>> s
[1, 0, 3, 2]
>>> S2 = SL2Z([Integer(0),-Integer(1),Integer(1),Integer(0)])
>>> L = SL2Z([Integer(1),Integer(1),Integer(0),Integer(1)])
>>> reps[Integer(0)] == SL2Z([Integer(1),Integer(0),Integer(0),Integer(1)])
True
>>> all(reps[i]*S2*~reps[s[i]] in G for i in range(Integer(4)))
True
>>> all(reps[i]*L*~reps[l[i]] in G for i in range(Integer(4)))
True
```

todd_coxeter_l_s2()

Returns a 4-tuple `(coset_reps, gens, l, s2)` where `coset_reps` is a list of coset representatives of the subgroup, `gens` a list of generators, `l` and `s2` are list that corresponds to the action of the matrix $S2$ and L on the cosets.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2='(1,2)(3,4)',S3='(1,2,3)')
sage: reps,gens,l,s=G.todd_coxeter_l_s2()
sage: reps
[
[1 0]  [ 0 -1]  [1 2]  [1 1]
[0 1], [ 1  0], [0 1], [0 1]
]
sage: len(gens)
3
sage: Matrix(2, 2, [1, 3, 0, 1]) in gens
True
sage: Matrix(2, 2, [1, 0, -1, 1]) in gens
True
sage: Matrix(2, 2, [1, -3, 1, -2]) in gens or Matrix(2, 2, [2, -3, 1, -1]) in␣
↪gens
True
sage: l
[3, 1, 0, 2]
sage: s
[1, 0, 3, 2]
sage: S2 = SL2Z([0,-1,1,0])
sage: L = SL2Z([1,1,0,1])
sage: reps[0] == SL2Z([1,0,0,1])
True
sage: all(reps[i]*S2*~reps[s[i]] in G for i in range(4))
True
sage: all(reps[i]*L*~reps[l[i]] in G for i in range(4))
True
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2='(1,2)(3,4)',S3='(1,2,3)')
```

(continues on next page)

(continued from previous page)

```
>>> reps,gens,l,s=G.todd_coxeter_l_s2()
>>> reps
[
[1 0]  [ 0 -1]  [1 2]  [1 1]
[0 1], [ 1  0], [0 1], [0 1]
]
>>> len(gens)
3
>>> Matrix(Integer(2), Integer(2), [Integer(1), Integer(3), Integer(0),␣
↪Integer(1)]) in gens
True
>>> Matrix(Integer(2), Integer(2), [Integer(1), Integer(0), -Integer(1),␣
↪Integer(1)]) in gens
True
>>> Matrix(Integer(2), Integer(2), [Integer(1), -Integer(3), Integer(1), -
↪Integer(2)]) in gens or Matrix(Integer(2), Integer(2), [Integer(2), -
↪Integer(3), Integer(1), -Integer(1)]) in gens
True
>>> l
[3, 1, 0, 2]
>>> s
[1, 0, 3, 2]
>>> S2 = SL2Z([Integer(0),-Integer(1),Integer(1),Integer(0)])
>>> L = SL2Z([Integer(1),Integer(1),Integer(0),Integer(1)])
>>> reps[Integer(0)] == SL2Z([Integer(1),Integer(0),Integer(0),Integer(1)])
True
>>> all(reps[i]*S2*~reps[s[i]] in G for i in range(Integer(4)))
True
>>> all(reps[i]*L*~reps[l[i]] in G for i in range(Integer(4)))
True
```

todd_coxeter_s2_s3()

Returns a 4-tuple `(coset_reps, gens, s2, s3)` where `coset_reps` are coset representatives of the subgroup, `gens` is a list of generators, `s2` and `s3` are the action of the matrices $S2$ and $S3$ on the list of cosets.

The so called *Todd-Coxeter algorithm* is a general method for coset enumeration for a subgroup of a group given by generators and relations.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2='(1,2)(3,4)',S3='(1,2,3)')
sage: G.genus()
0
sage: reps,gens,s2,s3=G.todd_coxeter_s2_s3()
sage: g1,g2 = gens
sage: g1 in G and g2 in G
True
sage: Matrix(2, 2, [-1, 3, -1, 2]) in gens
True
sage: Matrix(2, 2, [-1, 0, 1, -1]) in gens or Matrix(2, 2, [1, 0, 1, 1]) in␣
↪gens
True
sage: S2 = SL2Z([0,-1,1,0])
sage: S3 = SL2Z([0,1,-1,1])
sage: reps[0] == SL2Z([1,0,0,1])
True
```

(continues on next page)

(continued from previous page)

```
sage: all(reps[i]*S2*~reps[s2[i]] in G for i in range(4))
True
sage: all(reps[i]*S3*~reps[s3[i]] in G for i in range(4))
True
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2='(1,2)(3,4)',S3='(1,2,3)')
>>> G.genus()
0
>>> reps,gens,s2,s3=G.todd_coxeter_s2_s3()
>>> g1,g2 = gens
>>> g1 in G and g2 in G
True
>>> Matrix(Integer(2), Integer(2), [-Integer(1), Integer(3), -Integer(1),␣
↪Integer(2)]) in gens
True
>>> Matrix(Integer(2), Integer(2), [-Integer(1), Integer(0), Integer(1), -
↪Integer(1)]) in gens or Matrix(Integer(2), Integer(2), [Integer(1),␣
↪Integer(0), Integer(1), Integer(1)]) in gens
True
>>> S2 = SL2Z([Integer(0),-Integer(1),Integer(1),Integer(0)])
>>> S3 = SL2Z([Integer(0),Integer(1),-Integer(1),Integer(1)])
>>> reps[Integer(0)] == SL2Z([Integer(1),Integer(0),Integer(0),Integer(1)])
True
>>> all(reps[i]*S2*~reps[s2[i]] in G for i in range(Integer(4)))
True
>>> all(reps[i]*S3*~reps[s3[i]] in G for i in range(Integer(4)))
True
```

`sage.modular.arithgroup.arithgroup_perm.`**`HsuExample10`**`()`

An example of an index 10 arithmetic subgroup studied by Tim Hsu.

EXAMPLES:

```
sage: import sage.modular.arithgroup.arithgroup_perm as ap
sage: ap.HsuExample10()
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,4)(5,6)(7,8)(9,10)
 S3=(1,8,3)(2,4,6)(5,7,10)
 L=(1,4)(2,5,9,10,8)(3,7,6)
 R=(1,7,9,10,6)(2,3)(4,5,8)
```

```
>>> from sage.all import *
>>> import sage.modular.arithgroup.arithgroup_perm as ap
>>> ap.HsuExample10()
Arithmetic subgroup with permutations of right cosets
 S2=(1,2)(3,4)(5,6)(7,8)(9,10)
 S3=(1,8,3)(2,4,6)(5,7,10)
 L=(1,4)(2,5,9,10,8)(3,7,6)
 R=(1,7,9,10,6)(2,3)(4,5,8)
```

`sage.modular.arithgroup.arithgroup_perm.`**`HsuExample18`**`()`

An example of an index 18 arithmetic subgroup studied by Tim Hsu.

EXAMPLES:

```
sage: import sage.modular.arithgroup.arithgroup_perm as ap
sage: ap.HsuExample18()
Arithmetic subgroup with permutations of right cosets
 S2=(1,5)(2,11)(3,10)(4,15)(6,18)(7,12)(8,14)(9,16)(13,17)
 S3=(1,7,11)(2,18,5)(3,9,15)(4,14,10)(6,17,12)(8,13,16)
 L=(1,2)(3,4)(5,6,7)(8,9,10)(11,12,13,14,15,16,17,18)
 R=(1,12,18)(2,6,13,9,4,8,17,7)(3,16,14)(5,11)(10,15)
```

```
>>> from sage.all import *
>>> import sage.modular.arithgroup.arithgroup_perm as ap
>>> ap.HsuExample18()
Arithmetic subgroup with permutations of right cosets
 S2=(1,5)(2,11)(3,10)(4,15)(6,18)(7,12)(8,14)(9,16)(13,17)
 S3=(1,7,11)(2,18,5)(3,9,15)(4,14,10)(6,17,12)(8,13,16)
 L=(1,2)(3,4)(5,6,7)(8,9,10)(11,12,13,14,15,16,17,18)
 R=(1,12,18)(2,6,13,9,4,8,17,7)(3,16,14)(5,11)(10,15)
```

class sage.modular.arithgroup.arithgroup_perm.**OddArithmeticSubgroup_Permutation**(*S2*, *S3*, *L*, *R*, *canonical_labels=False*)

Bases: *ArithmeticSubgroup_Permutation_class*

An arithmetic subgroup of $\mathrm{SL}(2, \mathbf{Z})$ not containing -1, represented in terms of the right action of $\mathrm{SL}(2, \mathbf{Z})$ on its cosets.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2,3,4)",S3="(1,3)(2,4)")
sage: G
Arithmetic subgroup with permutations of right cosets
S2=(1,2,3,4)
S3=(1,3)(2,4)
L=(1,2,3,4)
R=(1,4,3,2)
sage: type(G)
<class 'sage.modular.arithgroup.arithgroup_perm.OddArithmeticSubgroup_Permutation_
↪with_category'>
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2,3,4)",S3="(1,3)(2,4)")
>>> G
Arithmetic subgroup with permutations of right cosets
S2=(1,2,3,4)
S3=(1,3)(2,4)
L=(1,2,3,4)
R=(1,4,3,2)
>>> type(G)
<class 'sage.modular.arithgroup.arithgroup_perm.OddArithmeticSubgroup_Permutation_
↪with_category'>
```

cusp_widths(*exp=False*)

Return the list of cusp widths.

INPUT:

- `exp` – boolean (default: `False`) - if True, return a dictionary with keys the possible widths and with values the number of cusp with that width.

EXAMPLES:

```
sage: G = Gamma1(5).as_permutation_group()
sage: G.cusp_widths()
[1, 1, 5, 5]
sage: G.cusp_widths(exp=True)
{1: 2, 5: 2}
```

```
>>> from sage.all import *
>>> G = Gamma1(Integer(5)).as_permutation_group()
>>> G.cusp_widths()
[1, 1, 5, 5]
>>> G.cusp_widths(exp=True)
{1: 2, 5: 2}
```

is_even()

Test whether the group is even.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,6,4,3)(2,7,5,8)",S3="(1,2,3,4,
↪5,6)(7,8)")
sage: G.is_even()
False
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,6,4,3)(2,7,5,8)",S3="(1,2,3,4,5,
↪6)(7,8)")
>>> G.is_even()
False
```

is_odd()

Test whether the group is odd.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,6,4,3)(2,7,5,8)",S3="(1,2,3,4,
↪5,6)(7,8)")
sage: G.is_odd()
True
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,6,4,3)(2,7,5,8)",S3="(1,2,3,4,5,
↪6)(7,8)")
>>> G.is_odd()
True
```

ncusps()

Returns the number of cusps.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2,3,4)",S3="(1,3)(2,4)")
sage: G.ncusps()
1

sage: G = Gamma1(3).as_permutation_group()
sage: G.ncusps()
2
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2,3,4)",S3="(1,3)(2,4)")
>>> G.ncusps()
1

>>> G = Gamma1(Integer(3)).as_permutation_group()
>>> G.ncusps()
2
```

nirregcusps()

Return the number of irregular cusps.

The cusps are associated to cycles of the permutations L or R. The irregular cusps are the one which are stabilised by $-Id$.

EXAMPLES:

```
sage: S2 = "(1,3,2,4)(5,7,6,8)(9,11,10,12)"
sage: S3 = "(1,3,5,2,4,6)(7,9,11,8,10,12)"
sage: G = ArithmeticSubgroup_Permutation(S2=S2,S3=S3)
sage: G.nirregcusps()
3
```

```
>>> from sage.all import *
>>> S2 = "(1,3,2,4)(5,7,6,8)(9,11,10,12)"
>>> S3 = "(1,3,5,2,4,6)(7,9,11,8,10,12)"
>>> G = ArithmeticSubgroup_Permutation(S2=S2,S3=S3)
>>> G.nirregcusps()
3
```

nregcusps()

Return the number of regular cusps of the group.

The cusps are associated to cycles of L or R. The irregular cusps correspond to the ones which are not stabilised by $-Id$.

EXAMPLES:

```
sage: G = Gamma1(3).as_permutation_group()
sage: G.nregcusps()
2
```

```
>>> from sage.all import *
>>> G = Gamma1(Integer(3)).as_permutation_group()
>>> G.nregcusps()
2
```

nu2()

Return the number of elliptic points of order 2.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2,3,4)",S3="(1,3)(2,4)")
sage: G.nu2()
0

sage: G = Gamma1(2).as_permutation_group()
sage: G.nu2()
1
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2,3,4)",S3="(1,3)(2,4)")
>>> G.nu2()
0

>>> G = Gamma1(Integer(2)).as_permutation_group()
>>> G.nu2()
1
```

nu3()

Return the number of elliptic points of order 3.

EXAMPLES:

```
sage: G = ArithmeticSubgroup_Permutation(S2="(1,2,3,4)",S3="(1,3)(2,4)")
sage: G.nu3()
2

sage: G = Gamma1(3).as_permutation_group()
sage: G.nu3()
1
```

```
>>> from sage.all import *
>>> G = ArithmeticSubgroup_Permutation(S2="(1,2,3,4)",S3="(1,3)(2,4)")
>>> G.nu3()
2

>>> G = Gamma1(Integer(3)).as_permutation_group()
>>> G.nu3()
1
```

to_even_subgroup(*relabel=True*)

Returns the group with $-Id$ added in it.

EXAMPLES:

```
sage: G = Gamma1(3).as_permutation_group()
sage: G.to_even_subgroup()
Arithmetic subgroup with permutations of right cosets
 S2=(1,3)(2,4)
 S3=(1,2,3)
 L=(2,3,4)
 R=(1,4,2)

sage: H = ArithmeticSubgroup_Permutation(S2 = '(1,4,11,14)(2,7,12,17)(3,5,13,
↪15)(6,9,16,19)(8,10,18,20)', S3 = '(1,2,3,11,12,13)(4,5,6,14,15,16)(7,8,9,
↪17,18,19)(10,20)')
```

(continues on next page)

(continued from previous page)

```
sage: G = H.to_even_subgroup(relabel=False); G
Arithmetic subgroup with permutations of right cosets
 S2=(1,4)(2,7)(3,5)(6,9)(8,10)
 S3=(1,2,3)(4,5,6)(7,8,9)
 L=(1,5)(2,4,9,10,8)(3,7,6)
 R=(1,7,10,8,6)(2,5,9)(3,4)
sage: H.is_subgroup(G)
True
```

```
>>> from sage.all import *
>>> G = Gamma1(Integer(3)).as_permutation_group()
>>> G.to_even_subgroup()
Arithmetic subgroup with permutations of right cosets
 S2=(1,3)(2,4)
 S3=(1,2,3)
 L=(2,3,4)
 R=(1,4,2)

>>> H = ArithmeticSubgroup_Permutation(S2 = '(1,4,11,14)(2,7,12,17)(3,5,13,
↪15)(6,9,16,19)(8,10,18,20)', S3 = '(1,2,3,11,12,13)(4,5,6,14,15,16)(7,8,9,
↪17,18,19)(10,20)')
>>> G = H.to_even_subgroup(relabel=False); G
Arithmetic subgroup with permutations of right cosets
 S2=(1,4)(2,7)(3,5)(6,9)(8,10)
 S3=(1,2,3)(4,5,6)(7,8,9)
 L=(1,5)(2,4,9,10,8)(3,7,6)
 R=(1,7,10,8,6)(2,5,9)(3,4)
>>> H.is_subgroup(G)
True
```

`sage.modular.arithgroup.arithgroup_perm.`**`eval_sl2z_word`**`(w)`

Given a word in the format output by *`sl2z_word_problem()`*, convert it back into an element of $\mathrm{SL}_2(\mathbf{Z})$.

EXAMPLES:

```
sage: from sage.modular.arithgroup.arithgroup_perm import eval_sl2z_word
sage: eval_sl2z_word([(0, 1), (1, -1), (0, 0), (1, 3), (0, 2), (1, 9), (0, -1)])
[ 66 -59]
[ 47 -42]
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.arithgroup_perm import eval_sl2z_word
>>> eval_sl2z_word([(Integer(0), Integer(1)), (Integer(1), -Integer(1)),␣
↪(Integer(0), Integer(0)), (Integer(1), Integer(3)), (Integer(0), Integer(2)),␣
↪(Integer(1), Integer(9)), (Integer(0), -Integer(1))])
[ 66 -59]
[ 47 -42]
```

`sage.modular.arithgroup.arithgroup_perm.`**`sl2z_word_problem`**`(A)`

Given an element of $\mathrm{SL}_2(\mathbf{Z})$, express it as a word in the generators L = [1,1,0,1] and R = [1,0,1,1].

The return format is a list of pairs `(a,b)`, where `a = 0` or `1` denoting `L` or `R` respectively, and `b` is an integer exponent.

See also the function *`eval_sl2z_word()`*.

EXAMPLES:

```
sage: from sage.modular.arithgroup.arithgroup_perm import eval_sl2z_word, sl2z_
↪word_problem
sage: m = SL2Z([1,0,0,1])
sage: eval_sl2z_word(sl2z_word_problem(m)) == m
True
sage: m = SL2Z([0,-1,1,0])
sage: eval_sl2z_word(sl2z_word_problem(m)) == m
True
sage: m = SL2Z([7,8,-50,-57])
sage: eval_sl2z_word(sl2z_word_problem(m)) == m
True
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.arithgroup_perm import eval_sl2z_word, sl2z_word_
↪problem
>>> m = SL2Z([Integer(1),Integer(0),Integer(0),Integer(1)])
>>> eval_sl2z_word(sl2z_word_problem(m)) == m
True
>>> m = SL2Z([Integer(0),-Integer(1),Integer(1),Integer(0)])
>>> eval_sl2z_word(sl2z_word_problem(m)) == m
True
>>> m = SL2Z([Integer(7),Integer(8),-Integer(50),-Integer(57)])
>>> eval_sl2z_word(sl2z_word_problem(m)) == m
True
```

`sage.modular.arithgroup.arithgroup_perm.`**`word_of_perms`**$(w, p1, p2)$

Given a word w as a list of 2-tuples `(index,power)` and permutations `p1` and `p2` return the product of `p1` and `p2` that corresponds to `w`.

EXAMPLES:

```
sage: import sage.modular.arithgroup.arithgroup_perm as ap
sage: S2 = SymmetricGroup(4)
sage: p1 = S2('(1,2)(3,4)')
sage: p2 = S2('(1,2,3,4)')
sage: ap.word_of_perms([(1,1),(0,1)], p1, p2) ==  p2 * p1
True
sage: ap.word_of_perms([(0,1),(1,1)], p1, p2) == p1 * p2
True
```

```
>>> from sage.all import *
>>> import sage.modular.arithgroup.arithgroup_perm as ap
>>> S2 = SymmetricGroup(Integer(4))
>>> p1 = S2('(1,2)(3,4)')
>>> p2 = S2('(1,2,3,4)')
>>> ap.word_of_perms([(Integer(1),Integer(1)),(Integer(0),Integer(1))], p1, p2)␣
↪==  p2 * p1
True
>>> ap.word_of_perms([(Integer(0),Integer(1)),(Integer(1),Integer(1))], p1, p2)␣
↪== p1 * p2
True
```

CHAPTER

THREE

ELEMENTS OF ARITHMETIC SUBGROUPS

class sage.modular.arithgroup.arithgroup_element.**ArithmeticSubgroupElement**

Bases: MultiplicativeGroupElement

An element of the group $\mathrm{SL}_2(\mathbf{Z})$, i.e. a 2x2 integer matrix of determinant 1.

a()

Return the upper left entry of `self`.

EXAMPLES:

```
sage: Gamma0(13)([7,1,13,2]).a()
7
```

```
>>> from sage.all import *
>>> Gamma0(Integer(13))([Integer(7),Integer(1),Integer(13),Integer(2)]).a()
7
```

acton(*z*)

Return the result of the action of `self` on z as a fractional linear transformation.

EXAMPLES:

```
sage: G = Gamma0(15)
sage: g = G([1, 2, 15, 31])
```

```
>>> from sage.all import *
>>> G = Gamma0(Integer(15))
>>> g = G([Integer(1), Integer(2), Integer(15), Integer(31)])
```

An example of g acting on a symbolic variable:

```
sage: z = var('z')                                                    #␣
↪needs sage.symbolic
sage: g.acton(z)                                                      #␣
↪needs sage.symbolic
(z + 2)/(15*z + 31)
```

```
>>> from sage.all import *
>>> z = var('z')                                                    #␣
↪needs sage.symbolic
>>> g.acton(z)                                                      #␣
↪needs sage.symbolic
(z + 2)/(15*z + 31)
```

An example involving the Gaussian numbers:

```
sage: # needs sage.rings.number_field
sage: x = polygen(ZZ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: g.acton(i)
1/1186*i + 77/1186
```

```
>>> from sage.all import *
>>> # needs sage.rings.number_field
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._
↪first_ngens(1)
>>> g.acton(i)
1/1186*i + 77/1186
```

An example with complex numbers:

```
sage: C.<i> = ComplexField()
sage: g.acton(i)
0.0649241146711636 + 0.000843170320404721*I
```

```
>>> from sage.all import *
>>> C = ComplexField(names=('i',)); (i,) = C._first_ngens(1)
>>> g.acton(i)
0.0649241146711636 + 0.000843170320404721*I
```

An example with the cusp infinity:

```
sage: g.acton(infinity)
1/15
```

```
>>> from sage.all import *
>>> g.acton(infinity)
1/15
```

An example which maps a finite cusp to infinity:

```
sage: g.acton(-31/15)
+Infinity
```

```
>>> from sage.all import *
>>> g.acton(-Integer(31)/Integer(15))
+Infinity
```

Note that when acting on instances of cusps the return value is still a rational number or infinity (Note the presence of '+', which does not show up for cusp instances):

```
sage: g.acton(Cusp(-31/15))
+Infinity
```

```
>>> from sage.all import *
>>> g.acton(Cusp(-Integer(31)/Integer(15)))
+Infinity
```

b()

Return the upper right entry of `self`.

EXAMPLES:

```
sage: Gamma0(13)([7,1,13,2]).b()
1
```

```
>>> from sage.all import *
>>> Gamma0(Integer(13))([Integer(7),Integer(1),Integer(13),Integer(2)]).b()
1
```

c()

Return the lower left entry of `self`.

EXAMPLES:

```
sage: Gamma0(13)([7,1,13,2]).c()
13
```

```
>>> from sage.all import *
>>> Gamma0(Integer(13))([Integer(7),Integer(1),Integer(13),Integer(2)]).c()
13
```

d()

Return the lower right entry of `self`.

EXAMPLES:

```
sage: Gamma0(13)([7,1,13,2]).d()
2
```

```
>>> from sage.all import *
>>> Gamma0(Integer(13))([Integer(7),Integer(1),Integer(13),Integer(2)]).d()
2
```

det()

Return the determinant of `self`, which is always 1.

EXAMPLES:

```
sage: Gamma1(11)([12,11,-11,-10]).det()
1
```

```
>>> from sage.all import *
>>> Gamma1(Integer(11))([Integer(12),Integer(11),-Integer(11),-Integer(10)]).
↪det()
1
```

determinant()

Return the determinant of `self`, which is always 1.

EXAMPLES:

```
sage: Gamma0(691)([1,0,691,1]).determinant()
1
```

```
>>> from sage.all import *
>>> Gamma0(Integer(691))([Integer(1),Integer(0),Integer(691),Integer(1)]).
↪determinant()
1
```

matrix()

Return the matrix corresponding to `self`.

EXAMPLES:

```
sage: x = Gamma1(3)([4,5,3,4]) ; x
[4 5]
[3 4]
sage: x.matrix()
[4 5]
[3 4]
sage: type(x.matrix())
<class 'sage.matrix.matrix_integer_dense.Matrix_integer_dense'>
```

```
>>> from sage.all import *
>>> x = Gamma1(Integer(3))([Integer(4),Integer(5),Integer(3),Integer(4)]) ; x
[4 5]
[3 4]
>>> x.matrix()
[4 5]
[3 4]
>>> type(x.matrix())
<class 'sage.matrix.matrix_integer_dense.Matrix_integer_dense'>
```

multiplicative_order()

Return the multiplicative order of this element.

EXAMPLES:

```
sage: SL2Z.one().multiplicative_order()
1
sage: SL2Z([-1,0,0,-1]).multiplicative_order()
2
sage: s,t = SL2Z.gens()
sage: ((t^3*s*t^2) * s * ~(t^3*s*t^2)).multiplicative_order()
4
sage: (t^3 * s * t * t^-3).multiplicative_order()
6
sage: (t^3 * s * t * s * t^-2).multiplicative_order()
3
sage: SL2Z([2,1,1,1]).multiplicative_order()
+Infinity
sage: SL2Z([-2,1,1,-1]).multiplicative_order()
+Infinity
```

```
>>> from sage.all import *
>>> SL2Z.one().multiplicative_order()
1
>>> SL2Z([-Integer(1),Integer(0),Integer(0),-Integer(1)]).multiplicative_
↪order()
2
>>> s,t = SL2Z.gens()
```

(continues on next page)

(continued from previous page)

```
>>> ((t**Integer(3)*s*t**Integer(2)) * s * ~(t**Integer(3)*s*t**Integer(2))).
↪multiplicative_order()
4
>>> (t**Integer(3) * s * t * t**-Integer(3)).multiplicative_order()
6
>>> (t**Integer(3) * s * t * s * t**-Integer(2)).multiplicative_order()
3
>>> SL2Z([Integer(2),Integer(1),Integer(1),Integer(1)]).multiplicative_order()
+Infinity
>>> SL2Z([-Integer(2),Integer(1),Integer(1),-Integer(1)]).multiplicative_
↪order()
+Infinity
```

CHAPTER

FOUR

CONGRUENCE ARITHMETIC SUBGROUPS OF $\mathrm{SL}_2(\mathbf{Z})$

Sage can compute extensively with the standard congruence subgroups $\Gamma_0(N)$, $\Gamma_1(N)$, and $\Gamma_H(N)$.

AUTHORS:

- William Stein
- David Loeffler (2009, 10) – modifications to work with more general arithmetic subgroups

class sage.modular.arithgroup.congroup_generic.**CongruenceSubgroup**(**args*, ***kwds*)

Bases: *CongruenceSubgroupFromGroup*

One of the "standard" congruence subgroups $\Gamma_0(N)$, $\Gamma_1(N)$, $\Gamma(N)$, or $\Gamma_H(N)$ (for some H).

This class is not intended to be instantiated directly. Derived subclasses must override `_contains_sl2`, `_repr_`, and `image_mod_n`.

image_mod_n()

Raise an error: all derived subclasses should override this function.

EXAMPLES:

```
sage: sage.modular.arithgroup.congroup_generic.CongruenceSubgroup(5).image_
↪mod_n()
Traceback (most recent call last):
...
NotImplementedError
```

```
>>> from sage.all import *
>>> sage.modular.arithgroup.congroup_generic.CongruenceSubgroup(Integer(5)).
↪image_mod_n()
Traceback (most recent call last):
...
NotImplementedError
```

modular_abelian_variety()

Return the modular abelian variety corresponding to the congruence subgroup self.

EXAMPLES:

```
sage: Gamma0(11).modular_abelian_variety()
Abelian variety J0(11) of dimension 1
sage: Gamma1(11).modular_abelian_variety()
Abelian variety J1(11) of dimension 1
sage: GammaH(11,[3]).modular_abelian_variety()
Abelian variety JH(11,[3]) of dimension 1
```

```
>>> from sage.all import *
>>> Gamma0(Integer(11)).modular_abelian_variety()
Abelian variety J0(11) of dimension 1
>>> Gamma1(Integer(11)).modular_abelian_variety()
Abelian variety J1(11) of dimension 1
>>> GammaH(Integer(11),[Integer(3)]).modular_abelian_variety()
Abelian variety JH(11,[3]) of dimension 1
```

modular_symbols(*sign=0*, *weight=2*, *base_ring=Rational Field*)

Return the space of modular symbols of the specified weight and sign on the congruence subgroup self.

EXAMPLES:

```
sage: G = Gamma0(23)
sage: G.modular_symbols()
Modular Symbols space of dimension 5 for Gamma_0(23) of weight 2 with sign 0␣
↪over Rational Field
sage: G.modular_symbols(weight=4)
Modular Symbols space of dimension 12 for Gamma_0(23) of weight 4 with sign 0␣
↪over Rational Field
sage: G.modular_symbols(base_ring=GF(7))
Modular Symbols space of dimension 5 for Gamma_0(23) of weight 2 with sign 0␣
↪over Finite Field of size 7
sage: G.modular_symbols(sign=1)
Modular Symbols space of dimension 3 for Gamma_0(23) of weight 2 with sign 1␣
↪over Rational Field
```

```
>>> from sage.all import *
>>> G = Gamma0(Integer(23))
>>> G.modular_symbols()
Modular Symbols space of dimension 5 for Gamma_0(23) of weight 2 with sign 0␣
↪over Rational Field
>>> G.modular_symbols(weight=Integer(4))
Modular Symbols space of dimension 12 for Gamma_0(23) of weight 4 with sign 0␣
↪over Rational Field
>>> G.modular_symbols(base_ring=GF(Integer(7)))
Modular Symbols space of dimension 5 for Gamma_0(23) of weight 2 with sign 0␣
↪over Finite Field of size 7
>>> G.modular_symbols(sign=Integer(1))
Modular Symbols space of dimension 3 for Gamma_0(23) of weight 2 with sign 1␣
↪over Rational Field
```

class sage.modular.arithgroup.congroup_generic.**CongruenceSubgroupBase**(*level*)

Bases: *ArithmeticSubgroup*

Create a congruence subgroup with given level.

EXAMPLES:

```
sage: Gamma0(500)
Congruence Subgroup Gamma0(500)
```

```
>>> from sage.all import *
>>> Gamma0(Integer(500))
Congruence Subgroup Gamma0(500)
```

is_congruence()

Return True, since this is a congruence subgroup.

EXAMPLES:

```
sage: Gamma0(7).is_congruence()
True
```

```
>>> from sage.all import *
>>> Gamma0(Integer(7)).is_congruence()
True
```

level()

Return the level of this congruence subgroup.

EXAMPLES:

```
sage: SL2Z.level()
1
sage: Gamma0(20).level()
20
sage: Gamma1(11).level()
11
sage: GammaH(14, [5]).level()
14
```

```
>>> from sage.all import *
>>> SL2Z.level()
1
>>> Gamma0(Integer(20)).level()
20
>>> Gamma1(Integer(11)).level()
11
>>> GammaH(Integer(14), [Integer(5)]).level()
14
```

class sage.modular.arithgroup.congroup_generic.**CongruenceSubgroupFromGroup**(G)

Bases: *CongruenceSubgroupBase*

A congruence subgroup, defined by the data of an integer N and a subgroup G of the finite group $SL(2, \mathbf{Z}/N\mathbf{Z})$; the congruence subgroup consists of all the matrices in $SL(2, \mathbf{Z})$ whose reduction modulo N lies in G.

This class should not be instantiated directly, but created using the factory function *CongruenceSubgroup_constructor()*, which accepts much more flexible input, and checks the input to make sure it is valid.

image_mod_n()

Return the subgroup of $SL(2, \mathbf{Z}/N\mathbf{Z})$ of which this is the preimage, where N is the level of self.

EXAMPLES:

```
sage: G = MatrixGroup([matrix(Zmod(2), 2, [1,1,1,0])])
sage: H = sage.modular.arithgroup.congroup_generic.
↪CongruenceSubgroupFromGroup(G); H.image_mod_n()
Matrix group over Ring of integers modulo 2 with 1 generators (
[1 1]
[1 0]
)
```

(continues on next page)

(continued from previous page)

```
sage: H.image_mod_n() == G
True
```

```
>>> from sage.all import *
>>> G = MatrixGroup([matrix(Zmod(Integer(2)), Integer(2), [Integer(1),
↪Integer(1),Integer(1),Integer(0)])])
>>> H = sage.modular.arithgroup.congroup_generic.
↪CongruenceSubgroupFromGroup(G); H.image_mod_n()
Matrix group over Ring of integers modulo 2 with 1 generators (
[1 1]
[1 0]
)
>>> H.image_mod_n() == G
True
```

index()

Return the index of self in the full modular group. This is equal to the index in $SL(2, \mathbf{Z}/N\mathbf{Z})$ of the image of this group modulo $\Gamma(N)$.

EXAMPLES:

```
sage: sage.modular.arithgroup.congroup_generic.
↪CongruenceSubgroupFromGroup(MatrixGroup([matrix(Zmod(2), 2, [1,1,1,0])])).
↪index()
2
```

```
>>> from sage.all import *
>>> sage.modular.arithgroup.congroup_generic.
↪CongruenceSubgroupFromGroup(MatrixGroup([matrix(Zmod(Integer(2)),␣
↪Integer(2), [Integer(1),Integer(1),Integer(1),Integer(0)])])).index()
2
```

to_even_subgroup()

Return the smallest even subgroup of $SL(2, \mathbf{Z})$ containing self.

EXAMPLES:

```
sage: G = Gamma(3)
sage: G.to_even_subgroup()
Congruence subgroup of SL(2,Z) of level 3, preimage of:
 Matrix group over Ring of integers modulo 3 with 1 generators (
[2 0]
[0 2]
)
```

```
>>> from sage.all import *
>>> G = Gamma(Integer(3))
>>> G.to_even_subgroup()
Congruence subgroup of SL(2,Z) of level 3, preimage of:
 Matrix group over Ring of integers modulo 3 with 1 generators (
[2 0]
[0 2]
)
```

sage.modular.arithgroup.congroup_generic.**CongruenceSubgroup_constructor**(**args*)

Attempt to create a congruence subgroup from the given data.

The allowed inputs are as follows:

- A `MatrixGroup` object. This must be a group of matrices over $\mathbf{Z}/N\mathbf{Z}$ for some N, with determinant 1, in which case the function will return the group of matrices in $SL(2, \mathbf{Z})$ whose reduction mod N is in the given group.
- A list of matrices over $\mathbf{Z}/N\mathbf{Z}$ for some N. The function will then compute the subgroup of $SL(2, \mathbf{Z})$ generated by these matrices, and proceed as above.
- An integer N and a list of matrices (over any ring coercible to $\mathbf{Z}/N\mathbf{Z}$, e.g. over $\mathbf{Z}$). The matrices will then be coerced to $\mathbf{Z}/N\mathbf{Z}$.

The function checks that the input G is valid. It then tests to see if G is the preimage mod N of some group of matrices modulo a proper divisor M of N, in which case it replaces G with this group before continuing.

EXAMPLES:

```
sage: from sage.modular.arithgroup.congroup_generic import CongruenceSubgroup_
↪constructor as CS
sage: CS(2, [[1,1,0,1]])
Congruence subgroup of SL(2,Z) of level 2, preimage of:
 Matrix group over Ring of integers modulo 2 with 1 generators (
[1 1]
[0 1]
)
sage: CS([matrix(Zmod(2), 2, [1,1,0,1])])
Congruence subgroup of SL(2,Z) of level 2, preimage of:
 Matrix group over Ring of integers modulo 2 with 1 generators (
[1 1]
[0 1]
)
sage: CS(MatrixGroup([matrix(Zmod(2), 2, [1,1,0,1])]))
Congruence subgroup of SL(2,Z) of level 2, preimage of:
 Matrix group over Ring of integers modulo 2 with 1 generators (
[1 1]
[0 1]
)
sage: CS(SL(2, 2))
Modular Group SL(2,Z)
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.congroup_generic import CongruenceSubgroup_
↪constructor as CS
>>> CS(Integer(2), [[Integer(1),Integer(1),Integer(0),Integer(1)]])
Congruence subgroup of SL(2,Z) of level 2, preimage of:
 Matrix group over Ring of integers modulo 2 with 1 generators (
[1 1]
[0 1]
)
>>> CS([matrix(Zmod(Integer(2)), Integer(2), [Integer(1),Integer(1),Integer(0),
↪Integer(1)])])
Congruence subgroup of SL(2,Z) of level 2, preimage of:
 Matrix group over Ring of integers modulo 2 with 1 generators (
[1 1]
[0 1]
)
>>> CS(MatrixGroup([matrix(Zmod(Integer(2)), Integer(2), [Integer(1),Integer(1),
↪Integer(0),Integer(1)])]))
Congruence subgroup of SL(2,Z) of level 2, preimage of:
```

(continues on next page)

(continued from previous page)

```
 Matrix group over Ring of integers modulo 2 with 1 generators (
[1 1]
[0 1]
)
>>> CS(SL(Integer(2), Integer(2)))
Modular Group SL(2,Z)
```

Some invalid inputs:

```
sage: CS(SU(2, 7))
Traceback (most recent call last):
...
TypeError: Ring of definition must be Z / NZ for some N
```

```
>>> from sage.all import *
>>> CS(SU(Integer(2), Integer(7)))
Traceback (most recent call last):
...
TypeError: Ring of definition must be Z / NZ for some N
```

`sage.modular.arithgroup.congroup_generic.`**`is_CongruenceSubgroup`**(x)

Return True if x is of type CongruenceSubgroup.

Note that this may be False even if x really is a congruence subgroup – it tests whether x is "obviously" congruence, i.e.~whether it has a congruence subgroup datatype. To test whether or not an arithmetic subgroup of $SL(2, \mathbf{Z})$ is congruence, use the `is_congruence()` method instead.

EXAMPLES:

```
sage: from sage.modular.arithgroup.congroup_generic import is_CongruenceSubgroup
sage: is_CongruenceSubgroup(SL2Z)
doctest:warning...
DeprecationWarning: The function is_CongruenceSubgroup is deprecated; use
↪'isinstance(..., CongruenceSubgroupBase)' instead.
See https://github.com/sagemath/sage/issues/38035 for details.
True
sage: is_CongruenceSubgroup(Gamma0(13))
True
sage: is_CongruenceSubgroup(Gamma1(6))
True
sage: is_CongruenceSubgroup(GammaH(11, [3]))
True
sage: G = ArithmeticSubgroup_Permutation(L = "(1, 2)", R = "(1, 2)"); is_
↪CongruenceSubgroup(G)
False
sage: G.is_congruence()
True
sage: is_CongruenceSubgroup(SymmetricGroup(3))
False
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.congroup_generic import is_CongruenceSubgroup
>>> is_CongruenceSubgroup(SL2Z)
doctest:warning...
DeprecationWarning: The function is_CongruenceSubgroup is deprecated; use
↪'isinstance(..., CongruenceSubgroupBase)' instead.
```

(continues on next page)

(continued from previous page)

```
See https://github.com/sagemath/sage/issues/38035 for details.
True
>>> is_CongruenceSubgroup(Gamma0(Integer(13)))
True
>>> is_CongruenceSubgroup(Gamma1(Integer(6)))
True
>>> is_CongruenceSubgroup(GammaH(Integer(11), [Integer(3)]))
True
>>> G = ArithmeticSubgroup_Permutation(L = "(1, 2)", R = "(1, 2)"); is_
↪CongruenceSubgroup(G)
False
>>> G.is_congruence()
True
>>> is_CongruenceSubgroup(SymmetricGroup(Integer(3)))
False
```

CHAPTER

FIVE

CONGRUENCE SUBGROUP $\Gamma_H(N)$

AUTHORS:

- Jordi Quer
- David Loeffler

class sage.modular.arithgroup.congroup_gammaH.**GammaH_class**(*level*, *H*, *Hlist=None*)

Bases: *CongruenceSubgroup*

The congruence subgroup $\Gamma_H(N)$ for some subgroup $H \trianglelefteq (\mathbf{Z}/N\mathbf{Z})^\times$, which is the subgroup of $\mathrm{SL}_2(\mathbf{Z})$ consisting of matrices of the form $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ with $N \mid c$ and $a, d \in H$.

atkin_lehner_matrix(Q)

Return the matrix of the Atkin–Lehner–Li operator W_Q associated to an exact divisor Q of N, where N is the level of this group; that is, $gcd(Q, N/Q) = 1$.

Note: We follow the conventions of [AL1978] here, so W_Q is given by the action of any matrix of the form $\begin{pmatrix} Qx & y \\ Nz & Qw \end{pmatrix}$ where x, y, z, w are integers such that $y = 1 \bmod Q$, $x = 1 \bmod N/Q$, and $det(W_Q) = Q$. For convenience, we actually always choose $x = y = 1$.

INPUT:

- Q (integer): an integer dividing N, where N is the level of this group. If this divisor does not satisfy $gcd(Q, N/Q) = 1$, it will be replaced by the unique integer with this property having the same prime factors as Q.

EXAMPLES:

```
sage: Gamma1(994).atkin_lehner_matrix(71)
[  71    1]
[4970   71]
sage: Gamma1(996).atkin_lehner_matrix(2)
[   4    1]
[-996 -248]
sage: Gamma1(15).atkin_lehner_matrix(7)
Traceback (most recent call last):
...
ValueError: Q must divide the level
```

```
>>> from sage.all import *
>>> Gamma1(Integer(994)).atkin_lehner_matrix(Integer(71))
```

(continues on next page)

(continued from previous page)

```
[  71    1]
[4970   71]
>>> Gamma1(Integer(996)).atkin_lehner_matrix(Integer(2))
[   4    1]
[-996 -248]
>>> Gamma1(Integer(15)).atkin_lehner_matrix(Integer(7))
Traceback (most recent call last):
...
ValueError: Q must divide the level
```

characters_mod_H(*sign=None*, *galois_orbits=False*)

Return the characters of $(\mathbf{Z}/N\mathbf{Z})^*$, of the specified sign, which are trivial on H.

INPUT:

- `sign` (default: None): if not None, return only characters of the given sign
- `galois_orbits` (default: `False`): if True, return only one character from each Galois orbit.

EXAMPLES:

```
sage: GammaH(5, [-1]).characters_mod_H()
[Dirichlet character modulo 5 of conductor 5 mapping 2 |--> -1,
 Dirichlet character modulo 5 of conductor 1 mapping 2 |--> 1]
sage: Gamma1(31).characters_mod_H(galois_orbits=True,sign=-1)
[Dirichlet character modulo 31 of conductor 31 mapping 3 |--> zeta30,
 Dirichlet character modulo 31 of conductor 31 mapping 3 |--> zeta30^3,
 Dirichlet character modulo 31 of conductor 31 mapping 3 |--> zeta30^5,
 Dirichlet character modulo 31 of conductor 31 mapping 3 |--> -1]
sage: GammaH(31, [-1]).characters_mod_H(sign=-1)
[]
```

```
>>> from sage.all import *
>>> GammaH(Integer(5), [-Integer(1)]).characters_mod_H()
[Dirichlet character modulo 5 of conductor 5 mapping 2 |--> -1,
 Dirichlet character modulo 5 of conductor 1 mapping 2 |--> 1]
>>> Gamma1(Integer(31)).characters_mod_H(galois_orbits=True,sign=-Integer(1))
[Dirichlet character modulo 31 of conductor 31 mapping 3 |--> zeta30,
 Dirichlet character modulo 31 of conductor 31 mapping 3 |--> zeta30^3,
 Dirichlet character modulo 31 of conductor 31 mapping 3 |--> zeta30^5,
 Dirichlet character modulo 31 of conductor 31 mapping 3 |--> -1]
>>> GammaH(Integer(31), [-Integer(1)]).characters_mod_H(sign=-Integer(1))
[]
```

coset_reps()

Return a set of coset representatives for self \ SL2Z.

EXAMPLES:

```
sage: list(Gamma1(3).coset_reps())
[
[1 0]  [-1  0]  [ 0 -1]  [ 0  1]  [1 0]  [-1  0]  [ 0 -1]  [ 0  1]
[0 1], [ 0 -1], [ 1  0], [-1  0], [1 1], [-1 -1], [ 1  2], [-1 -2]
]
sage: len(list(Gamma1(31).coset_reps())) == 31**2 - 1
True
```

```
>>> from sage.all import *
>>> list(Gamma1(Integer(3)).coset_reps())
[
[1 0]  [-1  0]  [ 0 -1]  [ 0  1]  [1 0]  [-1  0]  [ 0 -1]  [ 0  1]
[0 1], [ 0 -1], [ 1  0], [-1  0], [1 1], [-1 -1], [ 1  2], [-1 -2]
]
>>> len(list(Gamma1(Integer(31)).coset_reps())) == Integer(31)**Integer(2) -␣
↪Integer(1)
True
```

dimension_cusp_forms(*k=2*)

Return the dimension of the space of weight k cusp forms for this group. For $k \geq 2$, this is given by a standard formula in terms of k and various invariants of the group; see Diamond + Shurman, "A First Course in Modular Forms", section 3.5 and 3.6. If k is not given, default to k = 2.

For dimensions of spaces of cusp forms with character for Gamma1, use the dimension_cusp_forms method of the Gamma1 class, or the standalone function dimension_cusp_forms().

For weight 1 cusp forms, there is no simple formula for the dimensions, so we first try to rule out nonzero cusp forms existing via Riemann-Roch, and if this fails, we trigger computation of the cusp form space using Schaeffer's algorithm; this can be quite expensive in large levels.

EXAMPLES:

```
sage: GammaH(31, [23]).dimension_cusp_forms(10)
69
sage: GammaH(31, [7]).dimension_cusp_forms(1)
1
```

```
>>> from sage.all import *
>>> GammaH(Integer(31), [Integer(23)]).dimension_cusp_forms(Integer(10))
69
>>> GammaH(Integer(31), [Integer(7)]).dimension_cusp_forms(Integer(1))
1
```

dimension_new_cusp_forms(*k=2, p=0*)

Return the dimension of the space of new (or p-new) weight k cusp forms for this congruence subgroup.

INPUT:

- k – an integer (default: 2), the weight. Not fully implemented for k = 1.
- p – integer (default: 0); if nonzero, compute the p-new subspace.

OUTPUT: Integer

EXAMPLES:

```
sage: GammaH(33,[2]).dimension_new_cusp_forms()
3
sage: Gamma1(4*25).dimension_new_cusp_forms(2, p=5)
225
sage: Gamma1(33).dimension_new_cusp_forms(2)
19
sage: Gamma1(33).dimension_new_cusp_forms(2,p=11)
21
```

```
>>> from sage.all import *
>>> GammaH(Integer(33),[Integer(2)]).dimension_new_cusp_forms()
3
>>> Gamma1(Integer(4)*Integer(25)).dimension_new_cusp_forms(Integer(2),␣
↪p=Integer(5))
225
>>> Gamma1(Integer(33)).dimension_new_cusp_forms(Integer(2))
19
>>> Gamma1(Integer(33)).dimension_new_cusp_forms(Integer(2),p=Integer(11))
21
```

divisor_subgroups()

Given this congruence subgroup $\Gamma_H(N)$, return all subgroups $\Gamma_G(M)$ for M a divisor of N and such that G is equal to the image of H modulo M.

EXAMPLES:

```
sage: G = GammaH(33,[2]); G
Congruence Subgroup Gamma_H(33) with H generated by [2]
sage: G._list_of_elements_in_H()
[1, 2, 4, 8, 16, 17, 25, 29, 31, 32]
sage: G.divisor_subgroups()
[Modular Group SL(2,Z),
 Congruence Subgroup Gamma0(3),
 Congruence Subgroup Gamma0(11),
 Congruence Subgroup Gamma_H(33) with H generated by [2]]
```

```
>>> from sage.all import *
>>> G = GammaH(Integer(33),[Integer(2)]); G
Congruence Subgroup Gamma_H(33) with H generated by [2]
>>> G._list_of_elements_in_H()
[1, 2, 4, 8, 16, 17, 25, 29, 31, 32]
>>> G.divisor_subgroups()
[Modular Group SL(2,Z),
 Congruence Subgroup Gamma0(3),
 Congruence Subgroup Gamma0(11),
 Congruence Subgroup Gamma_H(33) with H generated by [2]]
```

extend(M)

Return the subgroup of $\Gamma_0(M)$, for M a multiple of N, obtained by taking the preimage of this group under the reduction map; in other words, the intersection of this group with $\Gamma_0(M)$.

EXAMPLES:

```
sage: G = GammaH(33, [2])
sage: G.extend(99)
Congruence Subgroup Gamma_H(99) with H generated by [2, 17, 68]
sage: G.extend(11)
Traceback (most recent call last):
...
ValueError: M (=11) must be a multiple of the level (33) of self
```

```
>>> from sage.all import *
>>> G = GammaH(Integer(33), [Integer(2)])
>>> G.extend(Integer(99))
Congruence Subgroup Gamma_H(99) with H generated by [2, 17, 68]
```

(continues on next page)

(continued from previous page)

```
>>> G.extend(Integer(11))
Traceback (most recent call last):
...
ValueError: M (=11) must be a multiple of the level (33) of self
```

gamma0_coset_reps()

Return a set of coset representatives for self \ Gamma0(N), where N is the level of self.

EXAMPLES:

```
sage: GammaH(108, [1,-1]).gamma0_coset_reps()
[
[1 0]  [-43  -2]  [ 31   2]  [-49  -5]  [ 25   3]  [-19  -3]
[0 1], [108   5], [108   7], [108  11], [108  13], [108  17],

[-17  -3]  [ 47  10]  [ 13   3]  [ 41  11]  [  7   2]  [-37 -12]
[108  19], [108  23], [108  25], [108  29], [108  31], [108  35],

[-35 -12]  [ 29  11]  [ -5  -2]  [ 23  10]  [-11  -5]  [ 53  26]
[108  37], [108  41], [108  43], [108  47], [108  49], [108  53]
]
```

```
>>> from sage.all import *
>>> GammaH(Integer(108), [Integer(1),-Integer(1)]).gamma0_coset_reps()
[
[1 0]  [-43  -2]  [ 31   2]  [-49  -5]  [ 25   3]  [-19  -3]
[0 1], [108   5], [108   7], [108  11], [108  13], [108  17],
<BLANKLINE>
[-17  -3]  [ 47  10]  [ 13   3]  [ 41  11]  [  7   2]  [-37 -12]
[108  19], [108  23], [108  25], [108  29], [108  31], [108  35],
<BLANKLINE>
[-35 -12]  [ 29  11]  [ -5  -2]  [ 23  10]  [-11  -5]  [ 53  26]
[108  37], [108  41], [108  43], [108  47], [108  49], [108  53]
]
```

generators(*algorithm='farey'*)

Return generators for this congruence subgroup. The result is cached.

INPUT:

- `algorithm` (string): either `farey` (default) or `todd-coxeter`.

If `algorithm` is set to `"farey"`, then the generators will be calculated using Farey symbols, which will always return a *minimal* generating set. See `farey_symbol` for more information.

If `algorithm` is set to `"todd-coxeter"`, a simpler algorithm based on Todd-Coxeter enumeration will be used. This tends to return far larger sets of generators.

EXAMPLES:

```
sage: GammaH(7, [2]).generators()
[
[1 1]  [ 2 -1]  [ 4 -3]
[0 1], [ 7 -3], [ 7 -5]
]
sage: GammaH(7, [2]).generators(algorithm="todd-coxeter")
[
```

(continues on next page)

(continued from previous page)

```
[1 1]  [-13   4]  [ 15   4]  [-3 -1]  [ 1 -1]  [1 0]  [1 1]  [-3 -1]
[0 1], [ 42 -13], [-49 -13], [ 7  2], [ 0  1], [7 1], [0 1], [ 7  2],

[-13   4]  [-5 -1]  [-5 -2]  [-10   3]  [ 1  0]  [ 2 -1]  [1 0]
[ 42 -13], [21  4], [28 11], [ 63 -19], [-7  1], [ 7 -3], [7 1],

[-3 -1]  [ 15  -4]  [ 2 -1]  [-5  1]  [  8  -3]  [11  5]  [-13  -4]
[ 7  2], [ 49 -13], [ 7 -3], [14 -3], [-21   8], [35 16], [-42 -13]
]
```

```
>>> from sage.all import *
>>> GammaH(Integer(7), [Integer(2)]).generators()
[
[1 1]  [ 2 -1]  [ 4 -3]
[0 1], [ 7 -3], [ 7 -5]
]
>>> GammaH(Integer(7), [Integer(2)]).generators(algorithm="todd-coxeter")
[
[1 1]  [-13   4]  [ 15   4]  [-3 -1]  [ 1 -1]  [1 0]  [1 1]  [-3 -1]
[0 1], [ 42 -13], [-49 -13], [ 7  2], [ 0  1], [7 1], [0 1], [ 7  2],
<BLANKLINE>
[-13   4]  [-5 -1]  [-5 -2]  [-10   3]  [ 1  0]  [ 2 -1]  [1 0]
[ 42 -13], [21  4], [28 11], [ 63 -19], [-7  1], [ 7 -3], [7 1],
<BLANKLINE>
[-3 -1]  [ 15  -4]  [ 2 -1]  [-5  1]  [  8  -3]  [11  5]  [-13  -4]
[ 7  2], [ 49 -13], [ 7 -3], [14 -3], [-21   8], [35 16], [-42 -13]
]
```

image_mod_n()

Return the image of this group in $SL(2, \mathbf{Z}/N\mathbf{Z})$.

EXAMPLES:

```
sage: Gamma0(3).image_mod_n()
Matrix group over Ring of integers modulo 3 with 2 generators (
[2 0]  [1 1]
[0 2], [0 1]
)
```

```
>>> from sage.all import *
>>> Gamma0(Integer(3)).image_mod_n()
Matrix group over Ring of integers modulo 3 with 2 generators (
[2 0]  [1 1]
[0 2], [0 1]
)
```

index()

Return the index of `self` in SL2Z.

EXAMPLES:

```
sage: [G.index() for G in Gamma0(40).gamma_h_subgroups()]  # optional - gap_
↪package_polycyclic
[72, 144, 144, 144, 144, 288, 288, 288, 288, 144, 288, 288, 576, 576, 144,␣
↪288, 288, 576, 576, 144, 288, 288, 576, 576, 288, 576, 1152]
```

```
>>> from sage.all import *
>>> [G.index() for G in Gamma0(Integer(40)).gamma_h_subgroups()]  # optional -
↪ gap_package_polycyclic
[72, 144, 144, 144, 144, 288, 288, 288, 288, 144, 288, 288, 576, 576, 144,␣
↪288, 288, 576, 576, 144, 288, 288, 576, 576, 288, 576, 1152]
```

is_even()

Return True precisely if this subgroup contains the matrix -1.

EXAMPLES:

```
sage: GammaH(10, [3]).is_even()
True
sage: GammaH(14, [1]).is_even()
False
```

```
>>> from sage.all import *
>>> GammaH(Integer(10), [Integer(3)]).is_even()
True
>>> GammaH(Integer(14), [Integer(1)]).is_even()
False
```

is_subgroup(*other*)

Return True if self is a subgroup of right, and False otherwise.

EXAMPLES:

```
sage: GammaH(24,[7]).is_subgroup(SL2Z)
True
sage: GammaH(24,[7]).is_subgroup(Gamma0(8))
True
sage: GammaH(24, []).is_subgroup(GammaH(24, [7]))
True
sage: GammaH(24, []).is_subgroup(Gamma1(24))
True
sage: GammaH(24, [17]).is_subgroup(GammaH(24, [7]))
False
sage: GammaH(1371, [169]).is_subgroup(GammaH(457, [169]))
True
```

```
>>> from sage.all import *
>>> GammaH(Integer(24),[Integer(7)]).is_subgroup(SL2Z)
True
>>> GammaH(Integer(24),[Integer(7)]).is_subgroup(Gamma0(Integer(8)))
True
>>> GammaH(Integer(24), []).is_subgroup(GammaH(Integer(24), [Integer(7)]))
True
>>> GammaH(Integer(24), []).is_subgroup(Gamma1(Integer(24)))
True
>>> GammaH(Integer(24), [Integer(17)]).is_subgroup(GammaH(Integer(24),␣
↪[Integer(7)]))
False
>>> GammaH(Integer(1371), [Integer(169)]).is_subgroup(GammaH(Integer(457),␣
↪[Integer(169)]))
True
```

ncusps()

Return the number of orbits of cusps (regular or otherwise) for this subgroup.

EXAMPLES:

```
sage: GammaH(33,[2]).ncusps()
8
sage: GammaH(32079, [21676]).ncusps()
28800
```

```
>>> from sage.all import *
>>> GammaH(Integer(33),[Integer(2)]).ncusps()
8
>>> GammaH(Integer(32079), [Integer(21676)]).ncusps()
28800
```

AUTHORS:

- Jordi Quer

nirregcusps()

Return the number of irregular cusps for this subgroup.

EXAMPLES:

```
sage: GammaH(3212, [2045, 2773]).nirregcusps()
720
```

```
>>> from sage.all import *
>>> GammaH(Integer(3212), [Integer(2045), Integer(2773)]).nirregcusps()
720
```

nregcusps()

Return the number of orbits of regular cusps for this subgroup. A cusp is regular if we may find a parabolic element generating the stabiliser of that cusp whose eigenvalues are both +1 rather than -1. If G contains -1, all cusps are regular.

EXAMPLES:

```
sage: GammaH(20, [17]).nregcusps()
4
sage: GammaH(20, [17]).nirregcusps()
2
sage: GammaH(3212, [2045, 2773]).nregcusps()
1440
sage: GammaH(3212, [2045, 2773]).nirregcusps()
720
```

```
>>> from sage.all import *
>>> GammaH(Integer(20), [Integer(17)]).nregcusps()
4
>>> GammaH(Integer(20), [Integer(17)]).nirregcusps()
2
>>> GammaH(Integer(3212), [Integer(2045), Integer(2773)]).nregcusps()
1440
>>> GammaH(Integer(3212), [Integer(2045), Integer(2773)]).nirregcusps()
720
```

AUTHOR:

- Jordi Quer

nu2()

Return the number of orbits of elliptic points of order 2 for this group.

EXAMPLES:

```
sage: [H.nu2() for n in [1..10] for H in Gamma0(n).gamma_h_subgroups()]  #␣
↪optional - gap_package_polycyclic
[1, 1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0,
↪ 0]
sage: GammaH(33,[2]).nu2()
0
sage: GammaH(5,[2]).nu2()
2
```

```
>>> from sage.all import *
>>> [H.nu2() for n in (ellipsis_range(Integer(1),Ellipsis,Integer(10))) for H␣
↪in Gamma0(n).gamma_h_subgroups()]  # optional - gap_package_polycyclic
[1, 1, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0,
↪ 0]
>>> GammaH(Integer(33),[Integer(2)]).nu2()
0
>>> GammaH(Integer(5),[Integer(2)]).nu2()
2
```

AUTHORS:

- Jordi Quer

nu3()

Return the number of orbits of elliptic points of order 3 for this group.

EXAMPLES:

```
sage: [H.nu3() for n in [1..10] for H in Gamma0(n).gamma_h_subgroups()]  #␣
↪optional - gap_package_polycyclic
[1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↪ 0]
sage: GammaH(33,[2]).nu3()
0
sage: GammaH(7,[2]).nu3()
2
```

```
>>> from sage.all import *
>>> [H.nu3() for n in (ellipsis_range(Integer(1),Ellipsis,Integer(10))) for H␣
↪in Gamma0(n).gamma_h_subgroups()]  # optional - gap_package_polycyclic
[1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
↪ 0]
>>> GammaH(Integer(33),[Integer(2)]).nu3()
0
>>> GammaH(Integer(7),[Integer(2)]).nu3()
2
```

AUTHORS:

- Jordi Quer

reduce_cusp(c)

Compute a minimal representative for the given cusp c. Returns a cusp c' which is equivalent to the given cusp, and is in lowest terms with minimal positive denominator, and minimal positive numerator for that denominator.

Two cusps u_1/v_1 and u_2/v_2 are equivalent modulo $\Gamma_H(N)$ if and only if

$$v_1 = hv_2 \bmod N \quad \text{and} \quad u_1 = h^{-1}u_2 \bmod \gcd(v_1, N)$$

or

$$v_1 = -hv_2 \bmod N \quad \text{and} \quad u_1 = -h^{-1}u_2 \bmod \gcd(v_1, N)$$

for some $h \in H$.

EXAMPLES:

```
sage: GammaH(6,[5]).reduce_cusp(5/3)
1/3
sage: GammaH(12,[5]).reduce_cusp(Cusp(8,9))
1/3
sage: GammaH(12,[5]).reduce_cusp(5/12)
Infinity
sage: GammaH(12,[]).reduce_cusp(Cusp(5,12))
5/12
sage: GammaH(21,[5]).reduce_cusp(Cusp(-9/14))
1/7
sage: Gamma1(5).reduce_cusp(oo)
Infinity
sage: Gamma1(5).reduce_cusp(0)
0
```

```
>>> from sage.all import *
>>> GammaH(Integer(6),[Integer(5)]).reduce_cusp(Integer(5)/Integer(3))
1/3
>>> GammaH(Integer(12),[Integer(5)]).reduce_cusp(Cusp(Integer(8),Integer(9)))
1/3
>>> GammaH(Integer(12),[Integer(5)]).reduce_cusp(Integer(5)/Integer(12))
Infinity
>>> GammaH(Integer(12),[]).reduce_cusp(Cusp(Integer(5),Integer(12)))
5/12
>>> GammaH(Integer(21),[Integer(5)]).reduce_cusp(Cusp(-Integer(9)/
↪Integer(14)))
1/7
>>> Gamma1(Integer(5)).reduce_cusp(oo)
Infinity
>>> Gamma1(Integer(5)).reduce_cusp(Integer(0))
0
```

restrict(M)

Return the subgroup of $\Gamma_0(M)$, for M a divisor of N, obtained by taking the image of this group under reduction modulo N.

EXAMPLES:

```
sage: G = GammaH(33,[2])
sage: G.restrict(11)
Congruence Subgroup Gamma0(11)
```

(continues on next page)

(continued from previous page)

```
sage: G.restrict(1)
Modular Group SL(2,Z)
sage: G.restrict(15)
Traceback (most recent call last):
...
ValueError: M (=15) must be a divisor of the level (33) of self
```

```
>>> from sage.all import *
>>> G = GammaH(Integer(33),[Integer(2)])
>>> G.restrict(Integer(11))
Congruence Subgroup Gamma0(11)
>>> G.restrict(Integer(1))
Modular Group SL(2,Z)
>>> G.restrict(Integer(15))
Traceback (most recent call last):
...
ValueError: M (=15) must be a divisor of the level (33) of self
```

to_even_subgroup()

Return the smallest even subgroup of $SL(2, \mathbf{Z})$ containing self.

EXAMPLES:

```
sage: GammaH(11, [4]).to_even_subgroup()
Congruence Subgroup Gamma0(11)
sage: Gamma1(11).to_even_subgroup()
Congruence Subgroup Gamma_H(11) with H generated by [10]
```

```
>>> from sage.all import *
>>> GammaH(Integer(11), [Integer(4)]).to_even_subgroup()
Congruence Subgroup Gamma0(11)
>>> Gamma1(Integer(11)).to_even_subgroup()
Congruence Subgroup Gamma_H(11) with H generated by [10]
```

sage.modular.arithgroup.congroup_gammaH.**GammaH_constructor**(*level*, *H*)

Return the congruence subgroup $\Gamma_H(N)$, which is the subgroup of $SL_2(\mathbf{Z})$ consisting of matrices of the form $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ with $N|c$ and $a, d \in H$, for H a specified subgroup of $(\mathbf{Z}/N\mathbf{Z})^\times$.

INPUT:

- level – an integer
- **H – either 0, 1, or a list**
  - If H is a list, return $\Gamma_H(N)$, where H is the subgroup of $(\mathbf{Z}/N\mathbf{Z})^*$ **generated** by the elements of the list.
  - If H = 0, returns $\Gamma_0(N)$.
  - If H = 1, returns $\Gamma_1(N)$.

EXAMPLES:

```
sage: GammaH(11,0) # indirect doctest
Congruence Subgroup Gamma0(11)
sage: GammaH(11,1)
Congruence Subgroup Gamma1(11)
```

(continues on next page)

(continued from previous page)

```
sage: GammaH(11,[10])
Congruence Subgroup Gamma_H(11) with H generated by [10]
sage: GammaH(11,[10,1])
Congruence Subgroup Gamma_H(11) with H generated by [10]
sage: GammaH(14,[10])
Traceback (most recent call last):
...
ArithmeticError: The generators [10] must be units modulo 14
```

```
>>> from sage.all import *
>>> GammaH(Integer(11),Integer(0)) # indirect doctest
Congruence Subgroup Gamma0(11)
>>> GammaH(Integer(11),Integer(1))
Congruence Subgroup Gamma1(11)
>>> GammaH(Integer(11),[Integer(10)])
Congruence Subgroup Gamma_H(11) with H generated by [10]
>>> GammaH(Integer(11),[Integer(10),Integer(1)])
Congruence Subgroup Gamma_H(11) with H generated by [10]
>>> GammaH(Integer(14),[Integer(10)])
Traceback (most recent call last):
...
ArithmeticError: The generators [10] must be units modulo 14
```

sage.modular.arithgroup.congroup_gammaH.**is_GammaH**(*x*)

Return True if x is a congruence subgroup of type GammaH.

EXAMPLES:

```
sage: from sage.modular.arithgroup.all import is_GammaH
sage: is_GammaH(GammaH(13, [2]))
doctest:warning...
DeprecationWarning: The function is_GammaH is deprecated; use 'isinstance(...,␣
↪GammaH_class)' instead.
See https://github.com/sagemath/sage/issues/38035 for details.
True
sage: is_GammaH(Gamma0(6))
True
sage: is_GammaH(Gamma1(6))
True
sage: is_GammaH(sage.modular.arithgroup.congroup_generic.CongruenceSubgroup(5))
False
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.all import is_GammaH
>>> is_GammaH(GammaH(Integer(13), [Integer(2)]))
doctest:warning...
DeprecationWarning: The function is_GammaH is deprecated; use 'isinstance(...,␣
↪GammaH_class)' instead.
See https://github.com/sagemath/sage/issues/38035 for details.
True
>>> is_GammaH(Gamma0(Integer(6)))
True
>>> is_GammaH(Gamma1(Integer(6)))
True
>>> is_GammaH(sage.modular.arithgroup.congroup_generic.
↪CongruenceSubgroup(Integer(5)))
```

(continues on next page)

(continued from previous page)

```
False
```

`sage.modular.arithgroup.congroup_gammaH.mumu(N)`

Return 0 if any cube divides N. Otherwise return $(-2)^v$ where v is the number of primes that exactly divide N.

This is similar to the Möbius function.

INPUT:

- `N` – an integer at least 1

OUTPUT: Integer

EXAMPLES:

```
sage: from sage.modular.arithgroup.congroup_gammaH import mumu
sage: mumu(27)
0
sage: mumu(6*25)
4
sage: mumu(7*9*25)
-2
sage: mumu(9*25)
1
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.congroup_gammaH import mumu
>>> mumu(Integer(27))
0
>>> mumu(Integer(6)*Integer(25))
4
>>> mumu(Integer(7)*Integer(9)*Integer(25))
-2
>>> mumu(Integer(9)*Integer(25))
1
```

CHAPTER

SIX

CONGRUENCE SUBGROUP $\Gamma_1(N)$

class sage.modular.arithgroup.congroup_gamma1.**Gamma1_class**(*level*)

Bases: *GammaH_class*

The congruence subgroup $\Gamma_1(N)$.

dimension_cusp_forms(*k=2*, *eps=None*, *algorithm='CohenOesterle'*)

Return the dimension of the space of cusp forms for `self`, or the dimension of the subspace corresponding to the given character if one is supplied.

INPUT:

- `k` – an integer (default: 2), the weight.
- `eps` – either None or a Dirichlet character modulo N, where N is the level of this group. If this is None, then the dimension of the whole space is returned; otherwise, the dimension of the subspace of forms of character eps.
- `algorithm` – either "CohenOesterle" (the default) or "Quer". This specifies the method to use in the case of nontrivial character: either the Cohen–Oesterle formula as described in Stein's book, or by Möbius inversion using the subgroups GammaH (a method due to Jordi Quer). Ignored for weight 1.

EXAMPLES:

We compute the same dimension in two different ways

```
sage: # needs sage.rings.number_field
sage: K = CyclotomicField(3)
sage: eps = DirichletGroup(7*43,K).0^2
sage: G = Gamma1(7*43)
```

```
>>> from sage.all import *
>>> # needs sage.rings.number_field
>>> K = CyclotomicField(Integer(3))
>>> eps = DirichletGroup(Integer(7)*Integer(43),K).gen(0)**Integer(2)
>>> G = Gamma1(Integer(7)*Integer(43))
```

Via Cohen–Oesterle:

```
sage: Gamma1(7*43).dimension_cusp_forms(2, eps)                          #␣
↪needs sage.rings.number_field
28
```

```
>>> from sage.all import *
>>> Gamma1(Integer(7)*Integer(43)).dimension_cusp_forms(Integer(2), eps)      ␣
```

(continues on next page)

(continued from previous page)

```
↪                          # needs sage.rings.number_field
28
```

Via Quer's method:

```
sage: Gamma1(7*43).dimension_cusp_forms(2, eps, algorithm="Quer")            #␣
↪needs sage.rings.number_field
28
```

```
>>> from sage.all import *
>>> Gamma1(Integer(7)*Integer(43)).dimension_cusp_forms(Integer(2), eps,␣
↪algorithm="Quer")            # needs sage.rings.number_field
28
```

Some more examples:

```
sage: G.<eps> = DirichletGroup(9)
sage: [Gamma1(9).dimension_cusp_forms(k, eps) for k in [1..10]]
[0, 0, 1, 0, 3, 0, 5, 0, 7, 0]
sage: [Gamma1(9).dimension_cusp_forms(k, eps^2) for k in [1..10]]
[0, 0, 0, 2, 0, 4, 0, 6, 0, 8]
```

```
>>> from sage.all import *
>>> G = DirichletGroup(Integer(9), names=('eps',)); (eps,) = G._first_ngens(1)
>>> [Gamma1(Integer(9)).dimension_cusp_forms(k, eps) for k in (ellipsis_
↪range(Integer(1),Ellipsis,Integer(10)))]
[0, 0, 1, 0, 3, 0, 5, 0, 7, 0]
>>> [Gamma1(Integer(9)).dimension_cusp_forms(k, eps**Integer(2)) for k in␣
↪(ellipsis_range(Integer(1),Ellipsis,Integer(10)))]
[0, 0, 0, 2, 0, 4, 0, 6, 0, 8]
```

In weight 1, we can sometimes rule out cusp forms existing via Riemann-Roch, but if this does not work, we trigger computation of the cusp forms space via Schaeffer's algorithm:

```
sage: chi = [u for u in DirichletGroup(40) if u(-1) == -1 and u(21) == 1][0]
sage: Gamma1(40).dimension_cusp_forms(1, chi)
0
sage: G = DirichletGroup(57); chi = (G.0) * (G.1)^6
sage: Gamma1(57).dimension_cusp_forms(1, chi)
1
```

```
>>> from sage.all import *
>>> chi = [u for u in DirichletGroup(Integer(40)) if u(-Integer(1)) == -
↪Integer(1) and u(Integer(21)) == Integer(1)][Integer(0)]
>>> Gamma1(Integer(40)).dimension_cusp_forms(Integer(1), chi)
0
>>> G = DirichletGroup(Integer(57)); chi = (G.gen(0)) * (G.gen(1))**Integer(6)
>>> Gamma1(Integer(57)).dimension_cusp_forms(Integer(1), chi)
1
```

dimension_eis(*k=2*, *eps=None*, *algorithm='CohenOesterle'*)

Return the dimension of the space of Eisenstein series forms for self, or the dimension of the subspace corresponding to the given character if one is supplied.

INPUT:

- `k` – an integer (default: 2), the weight.
- `eps` – either None or a Dirichlet character modulo N, where N is the level of this group. If this is None, then the dimension of the whole space is returned; otherwise, the dimension of the subspace of Eisenstein series of character eps.
- `algorithm` – either "CohenOesterle" (the default) or "Quer". This specifies the method to use in the case of nontrivial character: either the Cohen–Oesterle formula as described in Stein's book, or by Möbius inversion using the subgroups GammaH (a method due to Jordi Quer).

AUTHORS:

- William Stein - Cohen–Oesterle algorithm
- Jordi Quer - algorithm based on GammaH subgroups
- David Loeffler (2009) - code refactoring

EXAMPLES:

The following two computations use different algorithms:

```
sage: [Gamma1(36).dimension_eis(1,eps) for eps in DirichletGroup(36)]
[0, 4, 3, 0, 0, 2, 6, 0, 0, 2, 3, 0]
sage: [Gamma1(36).dimension_eis(1,eps,algorithm="Quer") for eps in␣
↪DirichletGroup(36)]
[0, 4, 3, 0, 0, 2, 6, 0, 0, 2, 3, 0]
```

```
>>> from sage.all import *
>>> [Gamma1(Integer(36)).dimension_eis(Integer(1),eps) for eps in␣
↪DirichletGroup(Integer(36))]
[0, 4, 3, 0, 0, 2, 6, 0, 0, 2, 3, 0]
>>> [Gamma1(Integer(36)).dimension_eis(Integer(1),eps,algorithm="Quer") for␣
↪eps in DirichletGroup(Integer(36))]
[0, 4, 3, 0, 0, 2, 6, 0, 0, 2, 3, 0]
```

So do these:

```
sage: [Gamma1(48).dimension_eis(3,eps) for eps in DirichletGroup(48)]
[0, 12, 0, 4, 0, 8, 0, 4, 12, 0, 4, 0, 8, 0, 4, 0]
sage: [Gamma1(48).dimension_eis(3,eps,algorithm="Quer") for eps in␣
↪DirichletGroup(48)]
[0, 12, 0, 4, 0, 8, 0, 4, 12, 0, 4, 0, 8, 0, 4, 0]
```

```
>>> from sage.all import *
>>> [Gamma1(Integer(48)).dimension_eis(Integer(3),eps) for eps in␣
↪DirichletGroup(Integer(48))]
[0, 12, 0, 4, 0, 8, 0, 4, 12, 0, 4, 0, 8, 0, 4, 0]
>>> [Gamma1(Integer(48)).dimension_eis(Integer(3),eps,algorithm="Quer") for␣
↪eps in DirichletGroup(Integer(48))]
[0, 12, 0, 4, 0, 8, 0, 4, 12, 0, 4, 0, 8, 0, 4, 0]
```

dimension_modular_forms (*k=2*, *eps=None*, *algorithm='CohenOesterle'*)

Return the dimension of the space of modular forms for self, or the dimension of the subspace corresponding to the given character if one is supplied.

INPUT:

- `k` – an integer (default: 2), the weight.

- `eps` – either None or a Dirichlet character modulo N, where N is the level of this group. If this is None, then the dimension of the whole space is returned; otherwise, the dimension of the subspace of forms of character eps.
- `algorithm` – either "CohenOesterle" (the default) or "Quer". This specifies the method to use in the case of nontrivial character: either the Cohen–Oesterle formula as described in Stein's book, or by Möbius inversion using the subgroups GammaH (a method due to Jordi Quer).

EXAMPLES:

```
sage: # needs sage.rings.number_field
sage: K = CyclotomicField(3)
sage: eps = DirichletGroup(7*43,K).0^2
sage: G = Gamma1(7*43)
sage: G.dimension_modular_forms(2, eps)
32
sage: G.dimension_modular_forms(2, eps, algorithm="Quer")
32
```

```
>>> from sage.all import *
>>> # needs sage.rings.number_field
>>> K = CyclotomicField(Integer(3))
>>> eps = DirichletGroup(Integer(7)*Integer(43),K).gen(0)**Integer(2)
>>> G = Gamma1(Integer(7)*Integer(43))
>>> G.dimension_modular_forms(Integer(2), eps)
32
>>> G.dimension_modular_forms(Integer(2), eps, algorithm="Quer")
32
```

dimension_new_cusp_forms(*k=2, eps=None, p=0, algorithm='CohenOesterle'*)

Dimension of the new subspace (or p-new subspace) of cusp forms of weight k and character ε.

INPUT:

- `k` – an integer (default: 2)
- `eps` – a Dirichlet character
- `p` – a prime (default: 0); just the p-new subspace if given
- `algorithm` – either "CohenOesterle" (the default) or "Quer". This specifies the method to use in the case of nontrivial character: either the Cohen–Oesterle formula as described in Stein's book, or by Möbius inversion using the subgroups GammaH (a method due to Jordi Quer).

EXAMPLES:

```
sage: G = DirichletGroup(9)
sage: eps = G.0^3
sage: eps.conductor()
3
sage: [Gamma1(9).dimension_new_cusp_forms(k, eps) for k in [2..10]]
[0, 0, 0, 2, 0, 2, 0, 2, 0]
sage: [Gamma1(9).dimension_cusp_forms(k, eps) for k in [2..10]]
[0, 0, 0, 2, 0, 4, 0, 6, 0]
sage: [Gamma1(9).dimension_new_cusp_forms(k, eps, 3) for k in [2..10]]
[0, 0, 0, 2, 0, 2, 0, 2, 0]
```

```
>>> from sage.all import *
>>> G = DirichletGroup(Integer(9))
```

(continues on next page)

(continued from previous page)

```
>>> eps = G.gen(0)**Integer(3)
>>> eps.conductor()
3
>>> [Gamma1(Integer(9)).dimension_new_cusp_forms(k, eps) for k in (ellipsis_
↪range(Integer(2),Ellipsis,Integer(10)))]
[0, 0, 0, 2, 0, 2, 0, 2, 0]
>>> [Gamma1(Integer(9)).dimension_cusp_forms(k, eps) for k in (ellipsis_
↪range(Integer(2),Ellipsis,Integer(10)))]
[0, 0, 0, 2, 0, 4, 0, 6, 0]
>>> [Gamma1(Integer(9)).dimension_new_cusp_forms(k, eps, Integer(3)) for k in␣
↪(ellipsis_range(Integer(2),Ellipsis,Integer(10)))]
[0, 0, 0, 2, 0, 2, 0, 2, 0]
```

Double check using modular symbols (independent calculation):

```
sage: [ModularSymbols(eps,k,sign=1).cuspidal_subspace().new_subspace().
↪dimension()  for k in [2..10]]
[0, 0, 0, 2, 0, 2, 0, 2, 0]
sage: [ModularSymbols(eps,k,sign=1).cuspidal_subspace().new_subspace(3).
↪dimension()  for k in [2..10]]
[0, 0, 0, 2, 0, 2, 0, 2, 0]
```

```
>>> from sage.all import *
>>> [ModularSymbols(eps,k,sign=Integer(1)).cuspidal_subspace().new_subspace().
↪dimension()  for k in (ellipsis_range(Integer(2),Ellipsis,Integer(10)))]
[0, 0, 0, 2, 0, 2, 0, 2, 0]
>>> [ModularSymbols(eps,k,sign=Integer(1)).cuspidal_subspace().new_
↪subspace(Integer(3)).dimension()  for k in (ellipsis_range(Integer(2),
↪Ellipsis,Integer(10)))]
[0, 0, 0, 2, 0, 2, 0, 2, 0]
```

Another example at level 33:

```
sage: G = DirichletGroup(33)
sage: eps = G.1
sage: eps.conductor()
11
sage: [Gamma1(33).dimension_new_cusp_forms(k, G.1) for k in [2..4]]
[0, 4, 0]
sage: [Gamma1(33).dimension_new_cusp_forms(k, G.1, algorithm="Quer") for k in␣
↪[2..4]]
[0, 4, 0]
sage: [Gamma1(33).dimension_new_cusp_forms(k, G.1^2) for k in [2..4]]
[2, 0, 6]
sage: [Gamma1(33).dimension_new_cusp_forms(k, G.1^2, p=3) for k in [2..4]]
[2, 0, 6]
```

```
>>> from sage.all import *
>>> G = DirichletGroup(Integer(33))
>>> eps = G.gen(1)
>>> eps.conductor()
11
>>> [Gamma1(Integer(33)).dimension_new_cusp_forms(k, G.gen(1)) for k in␣
↪(ellipsis_range(Integer(2),Ellipsis,Integer(4)))]
[0, 4, 0]
>>> [Gamma1(Integer(33)).dimension_new_cusp_forms(k, G.gen(1), algorithm="Quer
```

(continues on next page)

(continued from previous page)

```
↪") for k in (ellipsis_range(Integer(2),Ellipsis,Integer(4)))]
[0, 4, 0]
>>> [Gamma1(Integer(33)).dimension_new_cusp_forms(k, G.gen(1)**Integer(2))␣
↪for k in (ellipsis_range(Integer(2),Ellipsis,Integer(4)))]
[2, 0, 6]
>>> [Gamma1(Integer(33)).dimension_new_cusp_forms(k, G.gen(1)**Integer(2),␣
↪p=Integer(3)) for k in (ellipsis_range(Integer(2),Ellipsis,Integer(4)))]
[2, 0, 6]
```

generators(*algorithm='farey'*)

Return generators for this congruence subgroup. The result is cached.

INPUT:

- `algorithm` (string): either `farey` (default) or `todd-coxeter`.

If `algorithm` is set to `"farey"`, then the generators will be calculated using Farey symbols, which will always return a *minimal* generating set. See `farey_symbol` for more information.

If `algorithm` is set to `"todd-coxeter"`, a simpler algorithm based on Todd-Coxeter enumeration will be used. This tends to return far larger sets of generators.

EXAMPLES:

```
sage: Gamma1(3).generators()
[
[1 1]  [ 1 -1]
[0 1], [ 3 -2]
]
sage: Gamma1(3).generators(algorithm="todd-coxeter")
[
[1 1]  [-2  1]  [1 1]  [ 1 -1]  [1 0]  [1 1]  [-5  2]  [ 1  0]
[0 1], [-3  1], [0 1], [ 0  1], [3 1], [0 1], [12 -5], [-3  1],

[ 1 -1]  [ 1 -1]  [ 4 -1]  [ -5   3]
[ 3 -2], [ 3 -2], [ 9 -2], [-12   7]
]
```

```
>>> from sage.all import *
>>> Gamma1(Integer(3)).generators()
[
[1 1]  [ 1 -1]
[0 1], [ 3 -2]
]
>>> Gamma1(Integer(3)).generators(algorithm="todd-coxeter")
[
[1 1]  [-2  1]  [1 1]  [ 1 -1]  [1 0]  [1 1]  [-5  2]  [ 1  0]
[0 1], [-3  1], [0 1], [ 0  1], [3 1], [0 1], [12 -5], [-3  1],
<BLANKLINE>
[ 1 -1]  [ 1 -1]  [ 4 -1]  [ -5   3]
[ 3 -2], [ 3 -2], [ 9 -2], [-12   7]
]
```

index()

Return the index of self in the full modular group. This is given by the formula

$$N^2 \prod_{\substack{p|N \\ p \text{ prime}}} \left(1 - \frac{1}{p^2}\right).$$

EXAMPLES:

```
sage: Gamma1(180).index()
20736
sage: [Gamma1(n).projective_index() for n in [1..16]]
[1, 3, 4, 6, 12, 12, 24, 24, 36, 36, 60, 48, 84, 72, 96, 96]
```

```
>>> from sage.all import *
>>> Gamma1(Integer(180)).index()
20736
>>> [Gamma1(n).projective_index() for n in (ellipsis_range(Integer(1),
↪Ellipsis,Integer(16)))]
[1, 3, 4, 6, 12, 12, 24, 24, 36, 36, 60, 48, 84, 72, 96, 96]
```

is_even()

Return True precisely if this subgroup contains the matrix -1.

EXAMPLES:

```
sage: Gamma1(1).is_even()
True
sage: Gamma1(2).is_even()
True
sage: Gamma1(15).is_even()
False
```

```
>>> from sage.all import *
>>> Gamma1(Integer(1)).is_even()
True
>>> Gamma1(Integer(2)).is_even()
True
>>> Gamma1(Integer(15)).is_even()
False
```

is_subgroup(*right*)

Return True if self is a subgroup of right.

EXAMPLES:

```
sage: Gamma1(3).is_subgroup(SL2Z)
True
sage: Gamma1(3).is_subgroup(Gamma1(5))
False
sage: Gamma1(3).is_subgroup(Gamma1(6))
False
sage: Gamma1(6).is_subgroup(Gamma1(3))
True
sage: Gamma1(6).is_subgroup(Gamma0(2))
True
sage: Gamma1(80).is_subgroup(GammaH(40, []))
True
```

(continues on next page)

(continued from previous page)

```
sage: Gamma1(80).is_subgroup(GammaH(40, [21]))
True
```

```
>>> from sage.all import *
>>> Gamma1(Integer(3)).is_subgroup(SL2Z)
True
>>> Gamma1(Integer(3)).is_subgroup(Gamma1(Integer(5)))
False
>>> Gamma1(Integer(3)).is_subgroup(Gamma1(Integer(6)))
False
>>> Gamma1(Integer(6)).is_subgroup(Gamma1(Integer(3)))
True
>>> Gamma1(Integer(6)).is_subgroup(Gamma0(Integer(2)))
True
>>> Gamma1(Integer(80)).is_subgroup(GammaH(Integer(40), []))
True
>>> Gamma1(Integer(80)).is_subgroup(GammaH(Integer(40), [Integer(21)]))
True
```

ncusps ()

Return the number of cusps of this subgroup $\Gamma_1(N)$.

EXAMPLES:

```
sage: [Gamma1(n).ncusps() for n in [1..15]]
[1, 2, 2, 3, 4, 4, 6, 6, 8, 8, 10, 10, 12, 12, 16]
sage: [Gamma1(n).ncusps() for n in prime_range(2, 100)]
[2, 2, 4, 6, 10, 12, 16, 18, 22, 28, 30, 36, 40, 42, 46, 52, 58, 60, 66, 70,␣
↪72, 78, 82, 88, 96]
```

```
>>> from sage.all import *
>>> [Gamma1(n).ncusps() for n in (ellipsis_range(Integer(1),Ellipsis,
↪Integer(15)))]
[1, 2, 2, 3, 4, 4, 6, 6, 8, 8, 10, 10, 12, 12, 16]
>>> [Gamma1(n).ncusps() for n in prime_range(Integer(2), Integer(100))]
[2, 2, 4, 6, 10, 12, 16, 18, 22, 28, 30, 36, 40, 42, 46, 52, 58, 60, 66, 70,␣
↪72, 78, 82, 88, 96]
```

nu2 ()

Calculate the number of orbits of elliptic points of order 2 for this subgroup $\Gamma_1(N)$. This is known to be 0 if N > 2.

EXAMPLES:

```
sage: Gamma1(2).nu2()
1
sage: Gamma1(457).nu2()
0
sage: [Gamma1(n).nu2() for n in [1..16]]
[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
>>> from sage.all import *
>>> Gamma1(Integer(2)).nu2()
1
>>> Gamma1(Integer(457)).nu2()
```

(continues on next page)

(continued from previous page)

```
0
>>> [Gamma1(n).nu2() for n in (ellipsis_range(Integer(1),Ellipsis,
↪Integer(16)))]
[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

nu3()

Calculate the number of orbits of elliptic points of order 3 for this subgroup $\Gamma_1(N)$. This is known to be 0 if N > 3.

EXAMPLES:

```
sage: Gamma1(2).nu3()
0
sage: Gamma1(3).nu3()
1
sage: Gamma1(457).nu3()
0
sage: [Gamma1(n).nu3() for n in [1..10]]
[1, 0, 1, 0, 0, 0, 0, 0, 0, 0]
```

```
>>> from sage.all import *
>>> Gamma1(Integer(2)).nu3()
0
>>> Gamma1(Integer(3)).nu3()
1
>>> Gamma1(Integer(457)).nu3()
0
>>> [Gamma1(n).nu3() for n in (ellipsis_range(Integer(1),Ellipsis,
↪Integer(10)))]
[1, 0, 1, 0, 0, 0, 0, 0, 0, 0]
```

sage.modular.arithgroup.congroup_gamma1.**Gamma1_constructor**(N)

Return the congruence subgroup $\Gamma_1(N)$.

EXAMPLES:

```
sage: Gamma1(5) # indirect doctest
Congruence Subgroup Gamma1(5)
sage: G = Gamma1(23)
sage: G is Gamma1(23)
True
sage: G is GammaH(23, [1])
True
sage: TestSuite(G).run()
sage: G is loads(dumps(G))
True
```

```
>>> from sage.all import *
>>> Gamma1(Integer(5)) # indirect doctest
Congruence Subgroup Gamma1(5)
>>> G = Gamma1(Integer(23))
>>> G is Gamma1(Integer(23))
True
>>> G is GammaH(Integer(23), [Integer(1)])
True
>>> TestSuite(G).run()
```

(continues on next page)

(continued from previous page)

```
>>> G is loads(dumps(G))
True
```

`sage.modular.arithgroup.congroup_gamma1.is_Gamma1(x)`

Return True if x is a congruence subgroup of type Gamma1.

EXAMPLES:

```
sage: from sage.modular.arithgroup.all import is_Gamma1
sage: is_Gamma1(SL2Z)
doctest:warning...
DeprecationWarning: The function is_Gamma1 is deprecated; use 'isinstance(...,␣
↪Gamma1_class)' instead.
See https://github.com/sagemath/sage/issues/38035 for details.
False
sage: is_Gamma1(Gamma1(13))
True
sage: is_Gamma1(Gamma0(6))
False
sage: is_Gamma1(GammaH(12, [])) # trick question!
True
sage: is_Gamma1(GammaH(12, [5]))
False
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.all import is_Gamma1
>>> is_Gamma1(SL2Z)
doctest:warning...
DeprecationWarning: The function is_Gamma1 is deprecated; use 'isinstance(...,␣
↪Gamma1_class)' instead.
See https://github.com/sagemath/sage/issues/38035 for details.
False
>>> is_Gamma1(Gamma1(Integer(13)))
True
>>> is_Gamma1(Gamma0(Integer(6)))
False
>>> is_Gamma1(GammaH(Integer(12), [])) # trick question!
True
>>> is_Gamma1(GammaH(Integer(12), [Integer(5)]))
False
```

CHAPTER

SEVEN

CONGRUENCE SUBGROUP $\Gamma_0(N)$

class sage.modular.arithgroup.congroup_gamma0.**Gamma0_class**(*level*)

Bases: *GammaH_class*

The congruence subgroup $\Gamma_0(N)$.

coset_reps()

Return representatives for the right cosets of this congruence subgroup in $\mathrm{SL}_2(\mathbf{Z})$ as a generator object.

Use `list(self.coset_reps())` to obtain coset reps as a list.

EXAMPLES:

```
sage: list(Gamma0(5).coset_reps())
[
[1 0]  [ 0 -1]  [1 0]  [ 0 -1]  [ 0 -1]  [ 0 -1]
[0 1], [ 1  0], [1 1], [ 1  2], [ 1  3], [ 1  4]
]
sage: list(Gamma0(4).coset_reps())
[
[1 0]  [ 0 -1]  [1 0]  [ 0 -1]  [ 0 -1]  [1 0]
[0 1], [ 1  0], [1 1], [ 1  2], [ 1  3], [2 1]
]
sage: list(Gamma0(1).coset_reps())
[
[1 0]
[0 1]
]
```

```
>>> from sage.all import *
>>> list(Gamma0(Integer(5)).coset_reps())
[
[1 0]  [ 0 -1]  [1 0]  [ 0 -1]  [ 0 -1]  [ 0 -1]
[0 1], [ 1  0], [1 1], [ 1  2], [ 1  3], [ 1  4]
]
>>> list(Gamma0(Integer(4)).coset_reps())
[
[1 0]  [ 0 -1]  [1 0]  [ 0 -1]  [ 0 -1]  [1 0]
[0 1], [ 1  0], [1 1], [ 1  2], [ 1  3], [2 1]
]
>>> list(Gamma0(Integer(1)).coset_reps())
[
[1 0]
[0 1]
]
```

dimension_new_cusp_forms(*k=2, p=0*)

Return the dimension of the space of new (or p-new) weight k cusp forms for this congruence subgroup.

INPUT:

- k – an integer (default: 2), the weight. Not fully implemented for $k = 1$.
- p – integer (default: 0); if nonzero, compute the p-new subspace.

OUTPUT: Integer

ALGORITHM:

This comes from the formula given in Theorem 1 of http://www.math.ubc.ca/~gerg/papers/downloads/DSCFN.pdf

EXAMPLES:

```
sage: Gamma0(11000).dimension_new_cusp_forms()
240
sage: Gamma0(11000).dimension_new_cusp_forms(k=1)
0
sage: Gamma0(22).dimension_new_cusp_forms(k=4)
3
sage: Gamma0(389).dimension_new_cusp_forms(k=2,p=17)
32
```

```
>>> from sage.all import *
>>> Gamma0(Integer(11000)).dimension_new_cusp_forms()
240
>>> Gamma0(Integer(11000)).dimension_new_cusp_forms(k=Integer(1))
0
>>> Gamma0(Integer(22)).dimension_new_cusp_forms(k=Integer(4))
3
>>> Gamma0(Integer(389)).dimension_new_cusp_forms(k=Integer(2),p=Integer(17))
32
```

divisor_subgroups()

Return the subgroups of SL2Z of the form Gamma0(M) that contain this subgroup, i.e. those for M a divisor of N.

EXAMPLES:

```
sage: Gamma0(24).divisor_subgroups()
[Modular Group SL(2,Z),
 Congruence Subgroup Gamma0(2),
 Congruence Subgroup Gamma0(3),
 Congruence Subgroup Gamma0(4),
 Congruence Subgroup Gamma0(6),
 Congruence Subgroup Gamma0(8),
 Congruence Subgroup Gamma0(12),
 Congruence Subgroup Gamma0(24)]
```

```
>>> from sage.all import *
>>> Gamma0(Integer(24)).divisor_subgroups()
[Modular Group SL(2,Z),
 Congruence Subgroup Gamma0(2),
 Congruence Subgroup Gamma0(3),
 Congruence Subgroup Gamma0(4),
```

(continues on next page)

(continued from previous page)

```
 Congruence Subgroup Gamma0(6),
 Congruence Subgroup Gamma0(8),
 Congruence Subgroup Gamma0(12),
 Congruence Subgroup Gamma0(24)]
```

gamma_h_subgroups()

Return the subgroups of the form $\Gamma_H(N)$ contained in self, where N is the level of self.

EXAMPLES:

```
sage: G = Gamma0(11)
sage: G.gamma_h_subgroups()  # optional - gap_package_polycyclic
[Congruence Subgroup Gamma0(11),
 Congruence Subgroup Gamma_H(11) with H generated by [3],
 Congruence Subgroup Gamma_H(11) with H generated by [10],
 Congruence Subgroup Gamma1(11)]
sage: G = Gamma0(12)
sage: G.gamma_h_subgroups()  # optional - gap_package_polycyclic
[Congruence Subgroup Gamma0(12),
 Congruence Subgroup Gamma_H(12) with H generated by [7],
 Congruence Subgroup Gamma_H(12) with H generated by [11],
 Congruence Subgroup Gamma_H(12) with H generated by [5],
 Congruence Subgroup Gamma1(12)]
```

```
>>> from sage.all import *
>>> G = Gamma0(Integer(11))
>>> G.gamma_h_subgroups()  # optional - gap_package_polycyclic
[Congruence Subgroup Gamma0(11),
 Congruence Subgroup Gamma_H(11) with H generated by [3],
 Congruence Subgroup Gamma_H(11) with H generated by [10],
 Congruence Subgroup Gamma1(11)]
>>> G = Gamma0(Integer(12))
>>> G.gamma_h_subgroups()  # optional - gap_package_polycyclic
[Congruence Subgroup Gamma0(12),
 Congruence Subgroup Gamma_H(12) with H generated by [7],
 Congruence Subgroup Gamma_H(12) with H generated by [11],
 Congruence Subgroup Gamma_H(12) with H generated by [5],
 Congruence Subgroup Gamma1(12)]
```

generators(*algorithm='farey'*)

Return generators for this congruence subgroup.

INPUT:

- `algorithm` (string): either `"farey"` (default) or `"todd-coxeter"`.

If `algorithm` is set to `"farey"`, then the generators will be calculated using Farey symbols, which will always return a *minimal* generating set. See `farey_symbol` for more information.

If `algorithm` is set to `"todd-coxeter"`, a simpler algorithm based on Todd-Coxeter enumeration will be used. This tends to return far larger sets of generators.

EXAMPLES:

```
sage: Gamma0(3).generators()
[
[1 1]  [-1  1]
```

(continues on next page)

(continued from previous page)

```
[0 1], [-3  2]
]
sage: Gamma0(3).generators(algorithm="todd-coxeter")
[
[1 1]  [-1  0]  [ 1 -1]  [1 0]  [1 1]  [-1  0]  [ 1  0]
[0 1], [ 0 -1], [ 0  1], [3 1], [0 1], [ 3 -1], [-3  1]
]
sage: SL2Z.gens()
(
[ 0 -1]  [1 1]
[ 1  0], [0 1]
)
```

```
>>> from sage.all import *
>>> Gamma0(Integer(3)).generators()
[
[1 1]  [-1  1]
[0 1], [-3  2]
]
>>> Gamma0(Integer(3)).generators(algorithm="todd-coxeter")
[
[1 1]  [-1  0]  [ 1 -1]  [1 0]  [1 1]  [-1  0]  [ 1  0]
[0 1], [ 0 -1], [ 0  1], [3 1], [0 1], [ 3 -1], [-3  1]
]
>>> SL2Z.gens()
(
[ 0 -1]  [1 1]
[ 1  0], [0 1]
)
```

index()

Return the index of self in the full modular group.

This is given by

$$N \prod_{\substack{p \mid N \\ p \text{ prime}}} \left(1 + \frac{1}{p}\right).$$

EXAMPLES:

```
sage: [Gamma0(n).index() for n in [1..19]]
[1, 3, 4, 6, 6, 12, 8, 12, 12, 18, 12, 24, 14, 24, 24, 24, 18, 36, 20]
sage: Gamma0(32041).index()
32220
```

```
>>> from sage.all import *
>>> [Gamma0(n).index() for n in (ellipsis_range(Integer(1),Ellipsis,
↪Integer(19)))]
[1, 3, 4, 6, 6, 12, 8, 12, 12, 18, 12, 24, 14, 24, 24, 24, 18, 36, 20]
>>> Gamma0(Integer(32041)).index()
32220
```

is_even()

Return True precisely if this subgroup contains the matrix -1.

Since $\Gamma 0(N)$ always contains the matrix -1, this always returns True.

EXAMPLES:

```
sage: Gamma0(12).is_even()
True
sage: SL2Z.is_even()
True
```

```
>>> from sage.all import *
>>> Gamma0(Integer(12)).is_even()
True
>>> SL2Z.is_even()
True
```

is_subgroup(*right*)

Return True if self is a subgroup of right.

EXAMPLES:

```
sage: G = Gamma0(20)
sage: G.is_subgroup(SL2Z)
True
sage: G.is_subgroup(Gamma0(4))
True
sage: G.is_subgroup(Gamma0(20))
True
sage: G.is_subgroup(Gamma0(7))
False
sage: G.is_subgroup(Gamma1(20))
False
sage: G.is_subgroup(GammaH(40, []))
False
sage: Gamma0(80).is_subgroup(GammaH(40, [31, 21, 17]))
True
sage: Gamma0(2).is_subgroup(Gamma1(2))
True
```

```
>>> from sage.all import *
>>> G = Gamma0(Integer(20))
>>> G.is_subgroup(SL2Z)
True
>>> G.is_subgroup(Gamma0(Integer(4)))
True
>>> G.is_subgroup(Gamma0(Integer(20)))
True
>>> G.is_subgroup(Gamma0(Integer(7)))
False
>>> G.is_subgroup(Gamma1(Integer(20)))
False
>>> G.is_subgroup(GammaH(Integer(40), []))
False
>>> Gamma0(Integer(80)).is_subgroup(GammaH(Integer(40), [Integer(31),␣
↪Integer(21), Integer(17)]))
True
>>> Gamma0(Integer(2)).is_subgroup(Gamma1(Integer(2)))
True
```

ncusps()

Return the number of cusps of this subgroup $\Gamma_0(N)$.

EXAMPLES:

```
sage: [Gamma0(n).ncusps() for n in [1..19]]
[1, 2, 2, 3, 2, 4, 2, 4, 4, 4, 2, 6, 2, 4, 4, 6, 2, 8, 2]
sage: [Gamma0(n).ncusps() for n in prime_range(2,100)]
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

```
>>> from sage.all import *
>>> [Gamma0(n).ncusps() for n in (ellipsis_range(Integer(1),Ellipsis,
↪Integer(19)))]
[1, 2, 2, 3, 2, 4, 2, 4, 4, 4, 2, 6, 2, 4, 4, 6, 2, 8, 2]
>>> [Gamma0(n).ncusps() for n in prime_range(Integer(2),Integer(100))]
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

nu2()

Return the number of elliptic points of order 2 for this congruence subgroup $\Gamma_0(N)$.

The number of these is given by a standard formula: 0 if N is divisible by 4 or any prime congruent to -1 mod 4, and otherwise 2^d where d is the number of odd primes dividing N.

EXAMPLES:

```
sage: Gamma0(2).nu2()
1
sage: Gamma0(4).nu2()
0
sage: Gamma0(21).nu2()
0
sage: Gamma0(1105).nu2()
8
sage: [Gamma0(n).nu2() for n in [1..19]]
[1, 1, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 2, 0, 0]
```

```
>>> from sage.all import *
>>> Gamma0(Integer(2)).nu2()
1
>>> Gamma0(Integer(4)).nu2()
0
>>> Gamma0(Integer(21)).nu2()
0
>>> Gamma0(Integer(1105)).nu2()
8
>>> [Gamma0(n).nu2() for n in (ellipsis_range(Integer(1),Ellipsis,
↪Integer(19)))]
[1, 1, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 2, 0, 0]
```

nu3()

Return the number of elliptic points of order 3 for this congruence subgroup $\Gamma_0(N)$. The number of these is given by a standard formula: 0 if N is divisible by 9 or any prime congruent to -1 mod 3, and otherwise 2^d where d is the number of primes other than 3 dividing N.

EXAMPLES:

```
sage: Gamma0(2).nu3()
0
sage: Gamma0(3).nu3()
1
```

(continues on next page)

(continued from previous page)

```
sage: Gamma0(9).nu3()
0
sage: Gamma0(7).nu3()
2
sage: Gamma0(21).nu3()
2
sage: Gamma0(1729).nu3()
8
```

```
>>> from sage.all import *
>>> Gamma0(Integer(2)).nu3()
0
>>> Gamma0(Integer(3)).nu3()
1
>>> Gamma0(Integer(9)).nu3()
0
>>> Gamma0(Integer(7)).nu3()
2
>>> Gamma0(Integer(21)).nu3()
2
>>> Gamma0(Integer(1729)).nu3()
8
```

sage.modular.arithgroup.congroup_gamma0.**Gamma0_constructor**(*N*)

Return the congruence subgroup Gamma0(N).

EXAMPLES:

```
sage: G = Gamma0(51) ; G # indirect doctest
Congruence Subgroup Gamma0(51)
sage: G == Gamma0(51)
True
sage: G is Gamma0(51)
True
```

```
>>> from sage.all import *
>>> G = Gamma0(Integer(51)) ; G # indirect doctest
Congruence Subgroup Gamma0(51)
>>> G == Gamma0(Integer(51))
True
>>> G is Gamma0(Integer(51))
True
```

sage.modular.arithgroup.congroup_gamma0.**is_Gamma0**(*x*)

Return True if x is a congruence subgroup of type Gamma0.

EXAMPLES:

```
sage: from sage.modular.arithgroup.all import is_Gamma0
sage: is_Gamma0(SL2Z)
doctest:warning...
DeprecationWarning: The function is_Gamma0 is deprecated; use 'isinstance(...,␣
↪Gamma0_class)' instead.
See https://github.com/sagemath/sage/issues/38035 for details.
True
sage: is_Gamma0(Gamma0(13))
```

(continues on next page)

(continued from previous page)

```
True
sage: is_Gamma0(Gamma1(6))
False
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.all import is_Gamma0
>>> is_Gamma0(SL2Z)
doctest:warning...
DeprecationWarning: The function is_Gamma0 is deprecated; use 'isinstance(...,␣
↪Gamma0_class)' instead.
See https://github.com/sagemath/sage/issues/38035 for details.
True
>>> is_Gamma0(Gamma0(Integer(13)))
True
>>> is_Gamma0(Gamma1(Integer(6)))
False
```

CHAPTER

EIGHT

CONGRUENCE SUBGROUP $\Gamma(N)$

class sage.modular.arithgroup.congroup_gamma.**Gamma_class**(**args*, ***kwds*)

Bases: *CongruenceSubgroup*

The principal congruence subgroup $\Gamma(N)$.

are_equivalent(*x*, *y*, *trans=False*)

Check if the cusps x and y are equivalent under the action of this group.

ALGORITHM: The cusps u_1/v_1 and u_2/v_2 are equivalent modulo $\Gamma(N)$ if and only if $(u_1, v_1) = \pm(u_2, v_2) \bmod N$.

EXAMPLES:

```
sage: Gamma(7).are_equivalent(Cusp(2/3), Cusp(5/4))
True
```

```
>>> from sage.all import *
>>> Gamma(Integer(7)).are_equivalent(Cusp(Integer(2)/Integer(3)),␣
↪Cusp(Integer(5)/Integer(4)))
True
```

image_mod_n()

Return the image of this group modulo N, as a subgroup of $SL(2, \mathbf{Z}/N\mathbf{Z})$. This is just the trivial subgroup.

EXAMPLES:

```
sage: Gamma(3).image_mod_n()
Matrix group over Ring of integers modulo 3 with 1 generators (
[1 0]
[0 1]
)
```

```
>>> from sage.all import *
>>> Gamma(Integer(3)).image_mod_n()
Matrix group over Ring of integers modulo 3 with 1 generators (
[1 0]
[0 1]
)
```

index()

Return the index of self in the full modular group. This is given by

$$\prod_{\substack{p|N \\ p \text{ prime}}} \left(p^{3e} - p^{3e-2}\right).$$

EXAMPLES:

```
sage: [Gamma(n).index() for n in [1..19]]
[1, 6, 24, 48, 120, 144, 336, 384, 648, 720, 1320, 1152, 2184, 2016, 2880,␣
↪3072, 4896, 3888, 6840]
sage: Gamma(32041).index()
32893086819240
```

```
>>> from sage.all import *
>>> [Gamma(n).index() for n in (ellipsis_range(Integer(1),Ellipsis,
↪Integer(19)))]
[1, 6, 24, 48, 120, 144, 336, 384, 648, 720, 1320, 1152, 2184, 2016, 2880,␣
↪3072, 4896, 3888, 6840]
>>> Gamma(Integer(32041)).index()
32893086819240
```

ncusps()

Return the number of cusps of this subgroup $\Gamma(N)$.

EXAMPLES:

```
sage: [Gamma(n).ncusps() for n in [1..19]]
[1, 3, 4, 6, 12, 12, 24, 24, 36, 36, 60, 48, 84, 72, 96, 96, 144, 108, 180]
sage: Gamma(30030).ncusps()
278691840
sage: Gamma(2^30).ncusps()
432345564227567616
```

```
>>> from sage.all import *
>>> [Gamma(n).ncusps() for n in (ellipsis_range(Integer(1),Ellipsis,
↪Integer(19)))]
[1, 3, 4, 6, 12, 12, 24, 24, 36, 36, 60, 48, 84, 72, 96, 96, 144, 108, 180]
>>> Gamma(Integer(30030)).ncusps()
278691840
>>> Gamma(Integer(2)**Integer(30)).ncusps()
432345564227567616
```

nirregcusps()

Return the number of irregular cusps of self. For principal congruence subgroups this is always 0.

EXAMPLES:

```
sage: Gamma(17).nirregcusps()
0
```

```
>>> from sage.all import *
>>> Gamma(Integer(17)).nirregcusps()
0
```

nu3()

Return the number of elliptic points of order 3 for this arithmetic subgroup. Since this subgroup is $\Gamma(N)$ for $N \geq 2$, there are no such points, so we return 0.

EXAMPLES:

```
sage: Gamma(89).nu3()
0
```

```
>>> from sage.all import *
>>> Gamma(Integer(89)).nu3()
0
```

reduce_cusp(c)

Calculate the unique reduced representative of the equivalence of the cusp c modulo this group. The reduced representative of an equivalence class is the unique cusp in the class of the form u/v with $u, v \geq 0$ coprime, v minimal, and u minimal for that v.

EXAMPLES:

```
sage: Gamma(5).reduce_cusp(1/5)
Infinity
sage: Gamma(5).reduce_cusp(7/8)
3/2
sage: Gamma(6).reduce_cusp(4/3)
2/3
```

```
>>> from sage.all import *
>>> Gamma(Integer(5)).reduce_cusp(Integer(1)/Integer(5))
Infinity
>>> Gamma(Integer(5)).reduce_cusp(Integer(7)/Integer(8))
3/2
>>> Gamma(Integer(6)).reduce_cusp(Integer(4)/Integer(3))
2/3
```

sage.modular.arithgroup.congroup_gamma.**Gamma_constructor**(N)

Return the congruence subgroup $\Gamma(N)$.

EXAMPLES:

```
sage: Gamma(5) # indirect doctest
Congruence Subgroup Gamma(5)
sage: G = Gamma(23)
sage: G is Gamma(23)
True
sage: TestSuite(G).run()
```

```
>>> from sage.all import *
>>> Gamma(Integer(5)) # indirect doctest
Congruence Subgroup Gamma(5)
>>> G = Gamma(Integer(23))
>>> G is Gamma(Integer(23))
True
>>> TestSuite(G).run()
```

Test global uniqueness:

```
sage: G = Gamma(17)
sage: G is loads(dumps(G))
True
sage: G2 = sage.modular.arithgroup.congroup_gamma.Gamma_class(17)
sage: G == G2
True
sage: G is G2
False
```

```
>>> from sage.all import *
>>> G = Gamma(Integer(17))
>>> G is loads(dumps(G))
True
>>> G2 = sage.modular.arithgroup.congroup_gamma.Gamma_class(Integer(17))
>>> G == G2
True
>>> G is G2
False
```

`sage.modular.arithgroup.congroup_gamma.`**`is_Gamma`**`(x)`

Return True if x is a congruence subgroup of type Gamma.

EXAMPLES:

```
sage: from sage.modular.arithgroup.all import Gamma_class
sage: isinstance(Gamma0(13), Gamma_class)
False
sage: isinstance(Gamma(4), Gamma_class)
True
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.all import Gamma_class
>>> isinstance(Gamma0(Integer(13)), Gamma_class)
False
>>> isinstance(Gamma(Integer(4)), Gamma_class)
True
```

CHAPTER

NINE

THE MODULAR GROUP $\mathrm{SL}_2(\mathbf{Z})$

AUTHORS:

- Niles Johnson (2010-08): Issue #3893: `random_element()` should pass on `*args` and `**kwds`.

class `sage.modular.arithgroup.congroup_sl2z.`**`SL2Z_class`**

Bases: *Gamma0_class*

The full modular group $\mathrm{SL}_2(\mathbf{Z})$, regarded as a congruence subgroup of itself.

`is_subgroup`(*right*)

Return True if self is a subgroup of right.

EXAMPLES:

```
sage: SL2Z.is_subgroup(SL2Z)
True
sage: SL2Z.is_subgroup(Gamma1(1))
True
sage: SL2Z.is_subgroup(Gamma0(6))
False
```

```
>>> from sage.all import *
>>> SL2Z.is_subgroup(SL2Z)
True
>>> SL2Z.is_subgroup(Gamma1(Integer(1)))
True
>>> SL2Z.is_subgroup(Gamma0(Integer(6)))
False
```

`random_element`(*bound=100*, **args*, ***kwds*)

Return a random element of $\mathrm{SL}_2(\mathbf{Z})$ with entries whose absolute value is strictly less than bound (default 100). Additional arguments and keywords are passed to the random_element method of ZZ.

(Algorithm: Generate a random pair of integers at most bound. If they are not coprime, throw them away and start again. If they are, find an element of $\mathrm{SL}_2(\mathbf{Z})$ whose bottom row is that, and left-multiply it by $\begin{pmatrix} 1 & w \\ 0 & 1 \end{pmatrix}$ for an integer w randomly chosen from a small enough range that the answer still has entries at most bound.)

It is, unfortunately, not true that all elements of SL2Z with entries < bound appear with equal probability; those with larger bottom rows are favoured, because there are fewer valid possibilities for w.

EXAMPLES:

```
sage: s = SL2Z.random_element()
sage: s.parent() is SL2Z
True
sage: all(a in range(-99, 100) for a in list(s))
True
sage: S = set()
sage: while len(S) < 180:
....:     s = SL2Z.random_element(5)
....:     assert all(a in range(-4, 5) for a in list(s))
....:     assert s.parent() is SL2Z
....:     assert s in SL2Z
....:     S.add(s)
```

```
>>> from sage.all import *
>>> s = SL2Z.random_element()
>>> s.parent() is SL2Z
True
>>> all(a in range(-Integer(99), Integer(100)) for a in list(s))
True
>>> S = set()
>>> while len(S) < Integer(180):
...     s = SL2Z.random_element(Integer(5))
...     assert all(a in range(-Integer(4), Integer(5)) for a in list(s))
...     assert s.parent() is SL2Z
...     assert s in SL2Z
...     S.add(s)
```

Passes extra positional or keyword arguments through:

```
sage: SL2Z.random_element(5, distribution='1/n').parent() is SL2Z
True
```

```
>>> from sage.all import *
>>> SL2Z.random_element(Integer(5), distribution='1/n').parent() is SL2Z
True
```

reduce_cusp(*c*)

Return the unique reduced cusp equivalent to c under the action of self. Always returns Infinity, since there is only one equivalence class of cusps for $SL_2(Z)$.

EXAMPLES:

```
sage: SL2Z.reduce_cusp(Cusps(-1/4))
Infinity
```

```
>>> from sage.all import *
>>> SL2Z.reduce_cusp(Cusps(-Integer(1)/Integer(4)))
Infinity
```

sage.modular.arithgroup.congroup_sl2z.**is_SL2Z**(*x*)

Return True if x is the modular group $\mathrm{SL}_2(\mathbf{Z})$.

EXAMPLES:

```
sage: from sage.modular.arithgroup.all import is_SL2Z
sage: is_SL2Z(SL2Z)
```

(continues on next page)

(continued from previous page)

```
doctest:warning...
DeprecationWarning: The function is_SL2Z is deprecated; use 'isinstance(..., SL2Z_
↪class)' instead.
See https://github.com/sagemath/sage/issues/38035 for details.
True
sage: is_SL2Z(Gamma0(6))
False
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.all import is_SL2Z
>>> is_SL2Z(SL2Z)
doctest:warning...
DeprecationWarning: The function is_SL2Z is deprecated; use 'isinstance(..., SL2Z_
↪class)' instead.
See https://github.com/sagemath/sage/issues/38035 for details.
True
>>> is_SL2Z(Gamma0(Integer(6)))
False
```

CHAPTER

TEN

FAREY SYMBOL FOR ARITHMETIC SUBGROUPS OF $\mathrm{PSL}_2(\mathbf{Z})$

AUTHORS:

- Hartmut Monien (08 - 2011)

based on the *KFarey* package by Chris Kurth. Implemented as C++ module for speed.

class `sage.modular.arithgroup.farey_symbol.`**`Farey`**

Bases: `object`

A class for calculating Farey symbols of arithmetics subgroups of $\mathrm{PSL}_2(\mathbf{Z})$.

The arithmetic subgroup can be either any of the congruence subgroups implemented in Sage, i.e. Gamma, Gamma0, Gamma1 and GammaH or a subgroup of $\mathrm{PSL}_2(\mathbf{Z})$ which is given by a user written helper class defining membership in that group.

REFERENCES:

- Ravi S. Kulkarni, "An arithmetic-geometric method in the study of the subgroups of the modular group", Amer. J. Math., 113(6):1053–1133, 1991.

INPUT:

- G – an arithmetic subgroup of $\mathrm{PSL}_2(\mathbf{Z})$

EXAMPLES:

Create a Farey symbol for the group $\Gamma_0(11)$:

```
sage: f = FareySymbol(Gamma0(11)); f
FareySymbol(Congruence Subgroup Gamma0(11))
```

```
>>> from sage.all import *
>>> f = FareySymbol(Gamma0(Integer(11))); f
FareySymbol(Congruence Subgroup Gamma0(11))
```

Calculate the generators:

```
sage: f.generators()
[
[1 1]  [ 7 -2]  [ 8 -3]  [-1  0]
[0 1], [11 -3], [11 -4], [ 0 -1]
]
```

```
>>> from sage.all import *
>>> f.generators()
[
```

(continues on next page)

(continued from previous page)

```
[1 1]  [ 7 -2]  [ 8 -3]  [-1  0]
[0 1], [11 -3], [11 -4], [ 0 -1]
]
```

Pickling the FareySymbol and recovering it:

```
sage: f == loads(dumps(f))
True
```

```
>>> from sage.all import *
>>> f == loads(dumps(f))
True
```

Calculate the index of $\Gamma_H(33, [2, 5])$ in $\mathrm{PSL}_2(\mathbf{Z})$ via FareySymbol:

```
sage: FareySymbol(GammaH(33, [2, 5])).index()
48
```

```
>>> from sage.all import *
>>> FareySymbol(GammaH(Integer(33), [Integer(2), Integer(5)])).index()
48
```

Calculate the generators of $\Gamma_1(4)$:

```
sage: FareySymbol(Gamma1(4)).generators()
[
[1 1]  [-3  1]
[0 1], [-4  1]
]
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma1(Integer(4))).generators()
[
[1 1]  [-3  1]
[0 1], [-4  1]
]
```

Calculate the generators of the *example* of an index 10 arithmetic subgroup given by Tim Hsu:

```
sage: from sage.modular.arithgroup.arithgroup_perm import HsuExample10
sage: FareySymbol(HsuExample10()).generators()
[
[1 2]  [-2  1]  [ 4 -3]
[0 1], [-7  3], [ 3 -2]
]
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.arithgroup_perm import HsuExample10
>>> FareySymbol(HsuExample10()).generators()
[
[1 2]  [-2  1]  [ 4 -3]
[0 1], [-7  3], [ 3 -2]
]
```

Calculate the generators of the group $\Gamma' = \Gamma_0(8) \cap \Gamma_1(4)$ using a helper class to define group membership:

```
sage: class GPrime:
....:     def __contains__(self, M):
....:         return M in Gamma0(8) and M in Gamma1(4)

sage: FareySymbol(GPrime()).generators()
[
[1 1]  [ 5 -1]  [ 5 -2]
[0 1], [16 -3], [ 8 -3]
]
```

```
>>> from sage.all import *
>>> class GPrime:
...     def __contains__(self, M):
...         return M in Gamma0(Integer(8)) and M in Gamma1(Integer(4))

>>> FareySymbol(GPrime()).generators()
[
[1 1]  [ 5 -1]  [ 5 -2]
[0 1], [16 -3], [ 8 -3]
]
```

Calculate cusps of arithmetic subgroup defined via permutation group:

```
sage: L = SymmetricGroup(4)('(1, 2, 3)')

sage: R = SymmetricGroup(4)('(1, 2, 4)')

sage: FareySymbol(ArithmeticSubgroup_Permutation(L, R)).cusps()
[-1, Infinity]
```

```
>>> from sage.all import *
>>> L = SymmetricGroup(Integer(4))('(1, 2, 3)')

>>> R = SymmetricGroup(Integer(4))('(1, 2, 4)')

>>> FareySymbol(ArithmeticSubgroup_Permutation(L, R)).cusps()
[-1, Infinity]
```

Calculate the left coset representation of $\Gamma_H(8, [3])$:

```
sage: FareySymbol(GammaH(8, [3])).coset_reps()
[
[1 0]  [ 4 -1]  [ 3 -1]  [ 2 -1]  [ 1 -1]  [ 3 -1]  [ 2 -1]  [-1  0]
[0 1], [ 1  0], [ 1  0], [ 1  0], [ 1  0], [ 4 -1], [ 3 -1], [ 3 -1],
[ 1 -1]  [-1  0]  [ 0 -1]  [-1  0]
[ 2 -1], [ 2 -1], [ 1 -1], [ 1 -1]
]
```

```
>>> from sage.all import *
>>> FareySymbol(GammaH(Integer(8), [Integer(3)])).coset_reps()
[
[1 0]  [ 4 -1]  [ 3 -1]  [ 2 -1]  [ 1 -1]  [ 3 -1]  [ 2 -1]  [-1  0]
[0 1], [ 1  0], [ 1  0], [ 1  0], [ 1  0], [ 4 -1], [ 3 -1], [ 3 -1],
[ 1 -1]  [-1  0]  [ 0 -1]  [-1  0]
[ 2 -1], [ 2 -1], [ 1 -1], [ 1 -1]
]
```

coset_reps()

Left coset of the arithmetic group of the FareySymbol.

EXAMPLES:

Calculate the left coset of $\Gamma_0(6)$:

```
sage: FareySymbol(Gamma0(6)).coset_reps()
[
[1 0]  [ 3 -1]  [ 2 -1]  [ 1 -1]  [ 2 -1]  [ 3 -2]  [ 1 -1]  [-1  0]
[0 1], [ 1  0], [ 1  0], [ 1  0], [ 3 -1], [ 2 -1], [ 2 -1], [ 2 -1],
[ 1 -1]  [ 0 -1]  [-1  0]  [-2  1]
[ 3 -2], [ 1 -1], [ 1 -1], [ 1 -1]
]
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(6))).coset_reps()
[
[1 0]  [ 3 -1]  [ 2 -1]  [ 1 -1]  [ 2 -1]  [ 3 -2]  [ 1 -1]  [-1  0]
[0 1], [ 1  0], [ 1  0], [ 1  0], [ 3 -1], [ 2 -1], [ 2 -1], [ 2 -1],
[ 1 -1]  [ 0 -1]  [-1  0]  [-2  1]
[ 3 -2], [ 1 -1], [ 1 -1], [ 1 -1]
]
```

cusp_class(c)

Cusp class of a cusp in the FareySymbol.

INPUT:

- c – a cusp

EXAMPLES:

```
sage: FareySymbol(Gamma0(12)).cusp_class(Cusp(1, 12))
5
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(12))).cusp_class(Cusp(Integer(1), Integer(12)))
5
```

cusp_widths()

Cusps widths of the FareySymbol.

EXAMPLES:

```
sage: FareySymbol(Gamma0(6)).cusp_widths()
[6, 2, 3, 1]
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(6))).cusp_widths()
[6, 2, 3, 1]
```

cusps()

Cusps of the FareySymbol.

EXAMPLES:

```
sage: FareySymbol(Gamma0(6)).cusps()
[0, 1/3, 1/2, Infinity]
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(6))).cusps()
[0, 1/3, 1/2, Infinity]
```

fractions()

Fractions of the FareySymbol.

EXAMPLES:

```
sage: FareySymbol(Gamma(4)).fractions()
[0, 1/2, 1, 3/2, 2, 5/2, 3, 7/2, 4]
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma(Integer(4))).fractions()
[0, 1/2, 1, 3/2, 2, 5/2, 3, 7/2, 4]
```

fundamental_domain(*alpha=1*, *fill=True*, *thickness=1*, *color='lightgray'*, *color_even='white'*, *zorder=2*, *linestyle='solid'*, *show_pairing=True*, *tesselation='Dedekind'*, *ymax=1*, ***options*)

Plot a fundamental domain of an arithmetic subgroup of $\mathrm{PSL}_2(\mathbf{Z})$ corresponding to the Farey symbol.

OPTIONS:

- `fill` – boolean (default `True`) fill the fundamental domain
- `linestyle` – string (default: 'solid') The style of the line, which is one of 'dashed', 'dotted', 'solid', 'dashdot', or '–', ':', '-', '-.', respectively
- `color` – (default: 'lightgray') fill color; fill color for odd part of Dedekind tesselation.
- `show_pairing` – boolean (default: `True`) flag for pairing
- `tesselation` – (default: 'Dedekind') The type of hyperbolic tesselation which is one of 'coset', 'Dedekind' or `None` respectively
- `color_even` – fill color for even parts of Dedekind tesselation (default 'white'); ignored for other tesselations
- `thickness` – float (default: 1) the thickness of the line
- `ymax` – float (default: 1) maximal height

EXAMPLES:

For example, to plot the fundamental domain of $\Gamma_0(11)$ with pairings use the following command:

```
sage: FareySymbol(Gamma0(11)).fundamental_domain()                    #␣
↪needs sage.plot sage.symbolic
Graphics object consisting of 54 graphics primitives
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(11))).fundamental_domain()              ␣
↪     # needs sage.plot sage.symbolic
Graphics object consisting of 54 graphics primitives
```

indicating that side 1 is paired with side 3 and side 2 is paired with side 4, see also `paired_sides()`.

To plot the fundamental domain of $\Gamma(3)$ without pairings use the following command:

```
sage: FareySymbol(Gamma(3)).fundamental_domain(show_pairing=False)          #␣
↪needs sage.plot sage.symbolic
Graphics object consisting of 48 graphics primitives
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma(Integer(3))).fundamental_domain(show_pairing=False)    ␣
↪     # needs sage.plot sage.symbolic
Graphics object consisting of 48 graphics primitives
```

Plot the fundamental domain of $\Gamma_0(23)$ showing the left coset representatives:

```
sage: FareySymbol(Gamma0(23)).fundamental_domain(tesselation='coset')       #␣
↪needs sage.plot sage.symbolic
Graphics object consisting of 58 graphics primitives
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(23))).fundamental_domain(tesselation='coset') ␣
↪     # needs sage.plot sage.symbolic
Graphics object consisting of 58 graphics primitives
```

The same as above but with a custom linestyle:

```
sage: FareySymbol(Gamma0(23)).fundamental_domain(tesselation='coset',       #␣
↪needs sage.plot sage.symbolic
....:                                                 linestyle=':',
....:                                                 thickness='2')
Graphics object consisting of 58 graphics primitives
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(23))).fundamental_domain(tesselation='coset', ␣
↪     # needs sage.plot sage.symbolic
...                                                 linestyle=':',
...                                                 thickness='2')
Graphics object consisting of 58 graphics primitives
```

generators()

Minimal set of generators of the group of the FareySymbol.

EXAMPLES:

Calculate the generators of $\Gamma_0(6)$:

```
sage: FareySymbol(Gamma0(6)).generators()
[
[1 1]  [ 5 -1]  [ 7 -3]  [-1  0]
[0 1], [ 6 -1], [12 -5], [ 0 -1]
]
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(6))).generators()
[
[1 1]  [ 5 -1]  [ 7 -3]  [-1  0]
[0 1], [ 6 -1], [12 -5], [ 0 -1]
]
```

Calculate the generators of $\mathrm{SL}_2(\mathbf{Z})$:

```
sage: FareySymbol(SL2Z).generators()
[
[ 0 -1]  [ 0 -1]
[ 1  0], [ 1 -1]
]
```

```
>>> from sage.all import *
>>> FareySymbol(SL2Z).generators()
[
[ 0 -1]  [ 0 -1]
[ 1  0], [ 1 -1]
]
```

The unique index 2 even subgroup and index 4 odd subgroup each get handled correctly:

```
sage: # needs sage.groups
sage: FareySymbol(ArithmeticSubgroup_Permutation(S2="(1,2)", S3="()")).
↪generators()
[
[ 0  1]  [-1  1]
[-1 -1], [-1  0]
]
sage: FareySymbol(ArithmeticSubgroup_Permutation(S2="(1,2, 3, 4)", S3="(1,
↪3)(2,4)")).generators()
[
[ 0  1]  [-1  1]
[-1 -1], [-1  0]
]
```

```
>>> from sage.all import *
>>> # needs sage.groups
>>> FareySymbol(ArithmeticSubgroup_Permutation(S2="(1,2)", S3="()")).
↪generators()
[
[ 0  1]  [-1  1]
[-1 -1], [-1  0]
]
>>> FareySymbol(ArithmeticSubgroup_Permutation(S2="(1,2, 3, 4)", S3="(1,3)(2,
↪4)")).generators()
[
[ 0  1]  [-1  1]
[-1 -1], [-1  0]
]
```

genus()

Return the genus of the arithmetic group of the FareySymbol.

EXAMPLES:

```
sage: [FareySymbol(Gamma0(n)).genus() for n in range(16, 32)]
[0, 1, 0, 1, 1, 1, 2, 2, 1, 0, 2, 1, 2, 2, 3, 2]
```

```
>>> from sage.all import *
>>> [FareySymbol(Gamma0(n)).genus() for n in range(Integer(16), Integer(32))]
[0, 1, 0, 1, 1, 1, 2, 2, 1, 0, 2, 1, 2, 2, 3, 2]
```

index()

Return the index of the arithmetic group of the FareySymbol in $\mathrm{PSL}_2(\mathbf{Z})$.

EXAMPLES:

```
sage: [FareySymbol(Gamma0(n)).index() for n in range(1, 16)]
[1, 3, 4, 6, 6, 12, 8, 12, 12, 18, 12, 24, 14, 24, 24]
```

```
>>> from sage.all import *
>>> [FareySymbol(Gamma0(n)).index() for n in range(Integer(1), Integer(16))]
[1, 3, 4, 6, 6, 12, 8, 12, 12, 18, 12, 24, 14, 24, 24]
```

level()

Return the level of the arithmetic group of the FareySymbol.

EXAMPLES:

```
sage: [FareySymbol(Gamma0(n)).level() for n in range(1, 16)]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

```
>>> from sage.all import *
>>> [FareySymbol(Gamma0(n)).level() for n in range(Integer(1), Integer(16))]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

nu2()

Return the number of elliptic points of order two.

EXAMPLES:

```
sage: [FareySymbol(Gamma0(n)).nu2() for n in range(1, 16)]
[1, 1, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0]
```

```
>>> from sage.all import *
>>> [FareySymbol(Gamma0(n)).nu2() for n in range(Integer(1), Integer(16))]
[1, 1, 0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0]
```

nu3()

Return the number of elliptic points of order three.

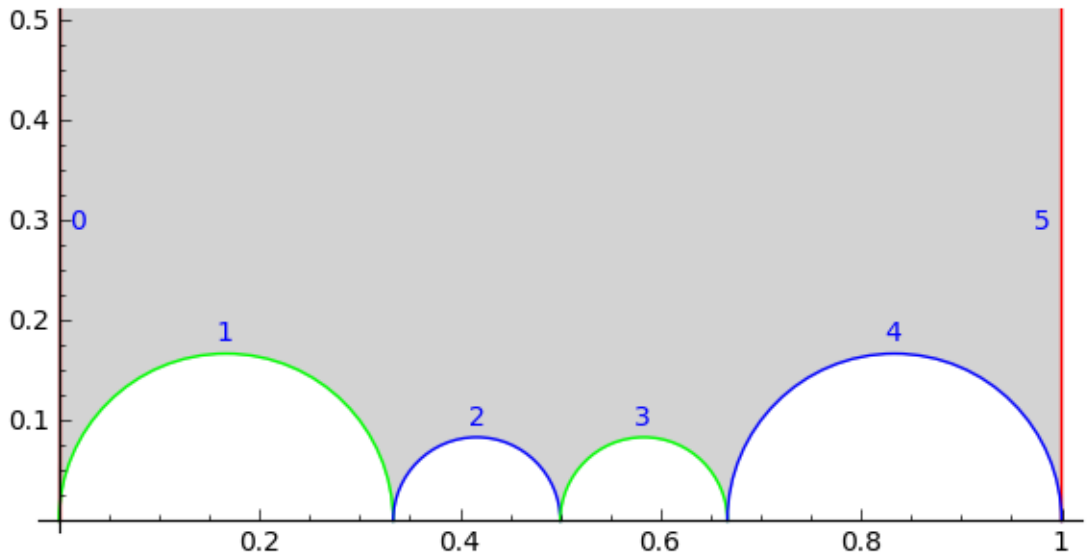
EXAMPLES:

```
sage: [FareySymbol(Gamma0(n)).nu3() for n in range(1, 16)]
[1, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0]
```

```
>>> from sage.all import *
>>> [FareySymbol(Gamma0(n)).nu3() for n in range(Integer(1), Integer(16))]
[1, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0]
```

paired_sides()

Pairs of index of the sides of the fundamental domain of the Farey symbol of the arithmetic group. The sides of the hyperbolic polygon are numbered 0, 1, … from left to right.



EXAMPLES:

```
sage: FareySymbol(Gamma0(11)).paired_sides()
[(0, 5), (1, 3), (2, 4)]
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(11))).paired_sides()
[(0, 5), (1, 3), (2, 4)]
```

indicating that the side 0 is paired with 5, 1 with 3 and 2 with 4.

pairing_matrices()

Pairing matrices of the sides of the fundamental domain. The sides of the hyperbolic polygon are numbered 0, 1, … from left to right.

EXAMPLES:

```
sage: FareySymbol(Gamma0(6)).pairing_matrices()
[
[1 1]  [ 5 -1]  [ 7 -3]  [ 5 -3]  [ 1 -1]  [-1  1]
[0 1], [ 6 -1], [12 -5], [12 -7], [ 6 -5], [ 0 -1]
]
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(6))).pairing_matrices()
[
[1 1]  [ 5 -1]  [ 7 -3]  [ 5 -3]  [ 1 -1]  [-1  1]
[0 1], [ 6 -1], [12 -5], [12 -7], [ 6 -5], [ 0 -1]
]
```

pairing_matrices_to_tietze_index()

Obtain the translation table from pairing matrices to generators.

The result is cached.

OUTPUT:

a list where the i-th entry is a nonzero integer k, such that if $k > 0$ then the i-th pairing matrix is (up to

sign) the $(k-1)$-th generator and, if $k < 0$, then the i-th pairing matrix is (up to sign) the inverse of the $(-k-1)$-th generator.

EXAMPLES:

```
sage: F = Gamma0(40).farey_symbol()
sage: table = F.pairing_matrices_to_tietze_index()
sage: table[12]
(-2, -1)
sage: F.pairing_matrices()[12]
[  3  -1]
[ 40 -13]
sage: F.generators()[1]**-1
[ -3   1]
[-40  13]
```

```
>>> from sage.all import *
>>> F = Gamma0(Integer(40)).farey_symbol()
>>> table = F.pairing_matrices_to_tietze_index()
>>> table[Integer(12)]
(-2, -1)
>>> F.pairing_matrices()[Integer(12)]
[  3  -1]
[ 40 -13]
>>> F.generators()[Integer(1)]**-Integer(1)
[ -3   1]
[-40  13]
```

pairings()

Pairings of the sides of the fundamental domain of the Farey symbol of the arithmetic group.

The sides of the hyperbolic polygon are numbered 0, 1, … from left to right. Conventions: even pairings are denoted by -2, odd pairings by -3 while free pairings are denoted by an integer number greater than zero.

EXAMPLES:

Odd pairings:

```
sage: FareySymbol(Gamma0(7)).pairings()
[1, -3, -3, 1]
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(7))).pairings()
[1, -3, -3, 1]
```

Even and odd pairings:

```
FareySymbol(Gamma0(13)).pairings()
[1, -3, -2, -2, -3, 1]
```

Only free pairings:

```
sage: FareySymbol(Gamma0(23)).pairings()
[1, 2, 3, 5, 3, 4, 2, 4, 5, 1]
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(23))).pairings()
[1, 2, 3, 5, 3, 4, 2, 4, 5, 1]
```

reduce_to_cusp(*r*)

Transformation of a rational number to cusp representative.

INPUT:

- `r` – a rational number

EXAMPLES:

```
sage: FareySymbol(Gamma0(12)).reduce_to_cusp(5/8)
[ 5  -3]
[12  -7]
```

```
>>> from sage.all import *
>>> FareySymbol(Gamma0(Integer(12))).reduce_to_cusp(Integer(5)/Integer(8))
[ 5  -3]
[12  -7]
```

Reduce 11/17 to a cusp of for HsuExample10():

```
sage: # needs sage.groups
sage: from sage.modular.arithgroup.arithgroup_perm import HsuExample10
sage: f = FareySymbol(HsuExample10())
sage: f.reduce_to_cusp(11/17)
[14 -9]
[-3  2]
sage: _.acton(11/17)
1
sage: f.cusps()[f.cusp_class(11/17)]
1
```

```
>>> from sage.all import *
>>> # needs sage.groups
>>> from sage.modular.arithgroup.arithgroup_perm import HsuExample10
>>> f = FareySymbol(HsuExample10())
>>> f.reduce_to_cusp(Integer(11)/Integer(17))
[14 -9]
[-3  2]
>>> _.acton(Integer(11)/Integer(17))
1
>>> f.cusps()[f.cusp_class(Integer(11)/Integer(17))]
1
```

word_problem(*M*, *output='standard'*, *check=True*)

Solve the word problem (up to sign) using this Farey symbol.

INPUT:

- `M` – An element M of $\mathrm{SL}_2(\mathbf{Z})$.
- `output` – (default: `'standard'`) Should be one of `'standard'`, `'syllables'`, `'gens'`.
- `check` – (default: `True`) Whether to check for correct input and output.

OUTPUT:

A solution to the word problem for the matrix M. The format depends on the `output` parameter, as follows.

- `standard` returns the so called the Tietze representation, consists of a tuple of nonzero integers i, where if $i > 0$ then it indicates the $i`thgenerator(thatis, ``self.generators()[0]`$ would correspond to $i = 1$), and if $i < 0$ then it indicates the inverse of the i-th generator.

- `syllables` returns a tuple of tuples of the form (i, n), where (i, n) represents `self.generators()[i] ^ n`, whose product equals M up to sign.
- `gens` returns tuple of tuples of the form (g, n), (g, n) such that the product of the matrices g^n equals M up to sign.

EXAMPLES:

```
sage: F = Gamma0(30).farey_symbol()
sage: gens = F.generators()
sage: g = gens[3] * gens[10] * gens[8]^-1 * gens[5]
sage: g
[-628597   73008]
[-692130   80387]
sage: F.word_problem(g)
(4, 11, -9, 6)
sage: g = gens[3] * gens[10]^2 * gens[8]^-1 * gens[5]
sage: g
[-5048053   586303]
[-5558280   645563]
sage: F.word_problem(g, output = 'gens')
((
[109 -10]
[120 -11], 1
),
 (
[ 19  -7]
[ 30 -11], 2
),
 (
[ 49  -9]
[ 60 -11], -1
),
 (
[17 -2]
[60 -7], 1
))
sage: F.word_problem(g, output = 'syllables')
((3, 1), (10, 2), (8, -1), (5, 1))
```

```
>>> from sage.all import *
>>> F = Gamma0(Integer(30)).farey_symbol()
>>> gens = F.generators()
>>> g = gens[Integer(3)] * gens[Integer(10)] * gens[Integer(8)]**-Integer(1)␣
↪* gens[Integer(5)]
>>> g
[-628597   73008]
[-692130   80387]
>>> F.word_problem(g)
(4, 11, -9, 6)
>>> g = gens[Integer(3)] * gens[Integer(10)]**Integer(2) * gens[Integer(8)]**-
↪Integer(1) * gens[Integer(5)]
>>> g
[-5048053   586303]
[-5558280   645563]
>>> F.word_problem(g, output = 'gens')
((
[109 -10]
```

(continues on next page)

(continued from previous page)

```
[120 -11], 1
),
 (
[ 19  -7]
[ 30 -11], 2
),
 (
[ 49  -9]
[ 60 -11], -1
),
 (
[17 -2]
[60 -7], 1
))
>>> F.word_problem(g, output = 'syllables')
((3, 1), (10, 2), (8, -1), (5, 1))
```

CHAPTER

ELEVEN

HELPER FUNCTIONS FOR CONGRUENCE SUBGROUPS

This file contains optimized Cython implementations of a few functions related to the standard congruence subgroups $\Gamma_0, \Gamma_1, \Gamma_H$. These functions are for internal use by routines elsewhere in the Sage library.

`sage.modular.arithgroup.congroup.`**`degeneracy_coset_representatives_gamma0`**(*N*, *M*, *t*)

Let N be a positive integer and M a divisor of N. Let t be a divisor of N/M, and let T be the 2×2 matrix $(1, 0; 0, t)$. This function returns representatives for the orbit set $\Gamma_0(N)\backslash T\Gamma_0(M)$, where $\Gamma_0(N)$ acts on the left on $T\Gamma_0(M)$.

INPUT:

- `N` – int
- `M` – int (divisor of N)
- `t` – int (divisor of N/M)

OUTPUT:

list – list of lists `[a,b,c,d]`, where `[a,b,c,d]` should be viewed as a 2x2 matrix.

This function is used for computation of degeneracy maps between spaces of modular symbols, hence its name.

We use that $T^{-1} \cdot (a, b; c, d) \cdot T = (a, bt; c/t, d)$, that the group $T^{-1}\Gamma_0(N)T$ is contained in $\Gamma_0(M)$, and that $\Gamma_0(N)T$ is contained in $T\Gamma_0(M)$.

ALGORITHM:

1. Compute representatives for $\Gamma_0(N/t, t)$ inside of $\Gamma_0(M)$:

- COSET EQUIVALENCE: Two right cosets represented by $[a, b; c, d]$ and $[a', b'; c', d']$ of $\Gamma_0(N/t, t)$ in $\mathrm{SL}_2(\mathbf{Z})$ are equivalent if and only if $(a, b) = (a', b')$ as points of $\mathbf{P}^1(\mathbf{Z}/t\mathbf{Z})$, i.e., $ab' \cong ba' \pmod{t}$, and $(c, d) = (c', d')$ as points of $\mathbf{P}^1(\mathbf{Z}/(N/t)\mathbf{Z})$.
- ALGORITHM to list all cosets:
  a) Compute the number of cosets.
  b) Compute a random element x of $\Gamma_0(M)$.
  c) Check if x is equivalent to anything generated so far; if not, add x to the list.
  d) Continue until the list is as long as the bound computed in step (a).

2. There is a bijection between $\Gamma_0(N)\backslash T\Gamma_0(M)$ and $\Gamma_0(N/t, t)\backslash\Gamma_0(M)$ given by $Tr \leftrightarrow r$. Consequently we obtain coset representatives for $\Gamma_0(N)\backslash T\Gamma_0(M)$ by left multiplying by T each coset representative of $\Gamma_0(N/t, t)\backslash\Gamma_0(M)$ found in step 1.

EXAMPLES:

```
sage: from sage.modular.arithgroup.all import degeneracy_coset_representatives_
↪gamma0
sage: len(degeneracy_coset_representatives_gamma0(13, 1, 1))
14
sage: len(degeneracy_coset_representatives_gamma0(13, 13, 1))
1
sage: len(degeneracy_coset_representatives_gamma0(13, 1, 13))
14
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.all import degeneracy_coset_representatives_
↪gamma0
>>> len(degeneracy_coset_representatives_gamma0(Integer(13), Integer(1),␣
↪Integer(1)))
14
>>> len(degeneracy_coset_representatives_gamma0(Integer(13), Integer(13),␣
↪Integer(1)))
1
>>> len(degeneracy_coset_representatives_gamma0(Integer(13), Integer(1),␣
↪Integer(13)))
14
```

`sage.modular.arithgroup.congroup.`**`degeneracy_coset_representatives_gamma1`**(N, M, t)

Let N be a positive integer and M a divisor of N. Let t be a divisor of N/M, and let T be the 2×2 matrix $(1, 0; 0, t)$. This function returns representatives for the orbit set $\Gamma_1(N)\backslash T\Gamma_1(M)$, where $\Gamma_1(N)$ acts on the left on $T\Gamma_1(M)$.

INPUT:

- `N` – int
- `M` – int (divisor of N)
- `t` – int (divisor of N/M)

OUTPUT:

list – list of lists `[a,b,c,d]`, where `[a,b,c,d]` should be viewed as a 2x2 matrix.

This function is used for computation of degeneracy maps between spaces of modular symbols, hence its name.

ALGORITHM:

Everything is the same as for *`degeneracy_coset_representatives_gamma0()`*, except for coset equivalence. Here $\Gamma_1(N/t, t)$ consists of matrices that are of the form $(1, *; 0, 1) \bmod N/t$ and $(1, 0; *, 1) \bmod t$.

COSET EQUIVALENCE: Two right cosets represented by $[a, b; c, d]$ and $[a', b'; c', d']$ of $\Gamma_1(N/t, t)$ in $\mathrm{SL}_2(\mathbf{Z})$ are equivalent if and only if

$$a \cong a' \pmod{t}, b \cong b' \pmod{t}, c \cong c' \pmod{N/t}, d \cong d' \pmod{N/t}.$$

EXAMPLES:

```
sage: from sage.modular.arithgroup.all import degeneracy_coset_representatives_
↪gamma1
sage: len(degeneracy_coset_representatives_gamma1(13, 1, 1))
168
sage: len(degeneracy_coset_representatives_gamma1(13, 13, 1))
1
```

(continues on next page)

(continued from previous page)

```
sage: len(degeneracy_coset_representatives_gamma1(13, 1, 13))
168
```

```
>>> from sage.all import *
>>> from sage.modular.arithgroup.all import degeneracy_coset_representatives_
↪gamma1
>>> len(degeneracy_coset_representatives_gamma1(Integer(13), Integer(1),␣
↪Integer(1)))
168
>>> len(degeneracy_coset_representatives_gamma1(Integer(13), Integer(13),␣
↪Integer(1)))
1
>>> len(degeneracy_coset_representatives_gamma1(Integer(13), Integer(1),␣
↪Integer(13)))
168
```

`sage.modular.arithgroup.congroup.`**`generators_helper`**(*coset_reps*, *level*)

Helper function for generators of Gamma0, Gamma1 and GammaH.

These are computed using coset representatives, via an "inverse Todd-Coxeter" algorithm, and generators for $\mathrm{SL}_2(\mathbf{Z})$.

ALGORITHM: Given coset representatives for a finite index subgroup G of $\mathrm{SL}_2(\mathbf{Z})$ we compute generators for G as follows. Let R be a set of coset representatives for G. Let $S, T \in \mathrm{SL}_2(\mathbf{Z})$ be defined by $(0, -1; 1, 0)$ and $(1, 1, 0, 1)$, respectively. Define maps $s, t : R \to G$ as follows. If $r \in R$, then there exists a unique $r' \in R$ such that $GrS = Gr'$. Let $s(r) = rSr'^{-1}$. Likewise, there is a unique r' such that $GrT = Gr'$ and we let $t(r) = rTr'^{-1}$. Note that $s(r)$ and $t(r)$ are in G for all r. Then G is generated by $s(R) \cup t(R)$.

There are more sophisticated algorithms using group actions on trees (and Farey symbols) that give smaller generating sets – this code is now deprecated in favour of the newer implementation based on Farey symbols.

EXAMPLES:

```
sage: Gamma0(7).generators(algorithm="todd-coxeter") # indirect doctest
[
[1 1]  [-1  0]  [ 1 -1]  [1 0]  [1 1]  [-3 -1]  [-2 -1]  [-5 -1]
[0 1], [ 0 -1], [ 0  1], [7 1], [0 1], [ 7  2], [ 7  3], [21  4],

[-4 -1]  [-1  0]  [ 1  0]
[21  5], [ 7 -1], [-7  1]
]
```

```
>>> from sage.all import *
>>> Gamma0(Integer(7)).generators(algorithm="todd-coxeter") # indirect doctest
[
[1 1]  [-1  0]  [ 1 -1]  [1 0]  [1 1]  [-3 -1]  [-2 -1]  [-5 -1]
[0 1], [ 0 -1], [ 0  1], [7 1], [0 1], [ 7  2], [ 7  3], [21  4],
<BLANKLINE>
[-4 -1]  [-1  0]  [ 1  0]
[21  5], [ 7 -1], [-7  1]
]
```

CHAPTER

TWELVE

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

m

INDEX

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

W