Sébastien Labbé Research Code Documentation

Release 0.5

Sébastien Labbé

Apr 10, 2019

CONTENTS

This is the reference manual for the Sébastien Labbé Research Code extension to the Sage mathematical software system. Sage is free open source math software that supports research and teaching in algebra, geometry, number theory, cryptography, and related areas.

Sébastien Labbé Research Code implements digital geometry, combinatorics on words and symbolic dynamical systems simulation code in Sage, via a set of new Python classes. Many of the modules corresponds to research code written for published articles (double square tiles, Christoffel graphs, factor complexity). It is meant to be reused and reusable (full documentation including doctests). Comments are welcome.

To install this module, you do:

```
sage -pip install slabbe
```

To use this module, you need to import it:

```
from slabbe import *
```

This reference manual contains many examples that illustrate the usage of slabbe spkg. The examples are all tested with each release of slabbe spkg, and should produce exactly the same output as in this manual, except for line breaks.

CHAPTER ONE

DIGITAL GEOMETRY

1.1 Discrete Subset

Digital geometry primitives

Subsets of ZZ^d with the edge relation +e_i and -e_i.

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: DiscreteSubset(dimension=2)
Subset of ZZ^2
sage: DiscreteSubset(dimension=4)
Subset of ZZ^4
```

A subset from an iterable:

```
sage: L = [(0,0,0,0), (1,0,0,0), (2,0,0,0), (3,0,0,0)]
sage: s = DiscreteSubset.from_subset(L)
sage: s
Subset of ZZ^4
```

A discrete 2d disk:

```
sage: D = DiscreteSubset(dimension=2, predicate=lambda (x,y) : x^2 + y^2 < 4)
sage: D.list()
[(0, 0), (0, 1), (0, -1), (1, 0), (-1, 0), (-1, 1), (1, -1), (1, 1), (-1, -1)]
sage: D
Subset of ZZ^2
```

A discrete 3d ball:

```
sage: predicate = lambda (x,y,z) : x^2 + y^2 + z^2 <= 4
sage: D = DiscreteSubset(dimension=3, predicate=predicate)
sage: D
Subset of ZZ^3
sage: (0,0,0) in D
True
sage: (10,10,10) in D
False
sage: len(D.list())
33
sage: D.plot()     # optional long
```

A discrete 4d hyperplane:

```
sage: predicate = lambda (x,y,z,w) : 0 <= 2*x + 3*y + 4*z + 5*w < 14
sage: D = DiscreteSubset(dimension=4, predicate=predicate)
sage: D
Subset of ZZ^4
sage: D.an_element()
(0, 0, 0, 0)
```

A 2d discrete box:

```
sage: from slabbe import DiscreteBox
sage: b = DiscreteBox([-5,5], [-5,5])
sage: b
Box: [-5, 5] x [-5, 5]
sage: b.plot()       # optional long
```

A 3d discrete box:

```
sage: b = DiscreteBox([-2,2], [-5,5], [-5,5])
sage: b
Box: [-2, 2] x [-5, 5] x [-5, 5]
sage: b.plot()       # optional long
```

The intersection of two discrete objects of the same dimension:

```
sage: circ = DiscreteSubset(dimension=2, predicate=lambda p: p[0]^2+p[1]^2<=100)
sage: b = DiscreteBox([0,10], [0,10])
sage: I = circ & b
sage: I
Intersection of the following objects:
Subset of ZZ^2
[0, 10] x [0, 10]
sage: I.an_element()
(0, 0)
sage: I.plot()      # optional long
```

A discrete tube (preimage of a discrete box by a matrix):

```
sage: from slabbe import M3to2
sage: M3to2
[-0.866025403784439  0.866025403784439  0.000000000000000]
[-0.500000000000000 -0.500000000000000   1.00000000000000]
sage: from slabbe import DiscreteTube
sage: tube = DiscreteTube([-5,5],[-5,5], projmat=M3to2)
sage: tube
DiscreteTube: Preimage of [-5, 5] x [-5, 5] by a 2 by 3 matrix
sage: it = iter(tube)
sage: [next(it) for _ in range(4)]
[(0, 0, 0), (1, 0, 0), (0, 0, 1), (0, 0, -1)]
```

TODO:

- Code Complement
- The method projection_matrix should be outside of the class?
- DiscreteTube should have a method projection_matrix
- Their should be an input saying whether the object is connected or not and what kind of neighbor connectedness

class slabbe.discrete_subset.**DiscreteBox**(**args*)

Bases: *slabbe.discrete_subset.DiscreteSubset*

Cartesian product of intervals.

INPUT:

- `*args` - intervals, lists of size two : [min, max]

EXAMPLES:

```
sage: from slabbe import DiscreteBox
sage: DiscreteBox([-5,5],[-5,5])
Box: [-5, 5] x [-5, 5]
```

```
sage: D = DiscreteBox([-3,3],[-3,3],[-3,3],[-3,3])
sage: next(iter(D))
(0, 0, 0, 0)
```

TESTS:

```
sage: d = DiscreteBox([-5,5], [-5,5], [-4,4])
sage: d.edges_iterator().next()
((0, 0, 0), (1, 0, 0))
```

clip(*space=1*)

Return a good clip rectangle for this box.

INPUT:

- `space` – number (default: 1), inner space within the box

EXAMPLES:

```
sage: from slabbe import DiscreteBox
sage: box = DiscreteBox([-6,6],[-6,6])
sage: box
Box: [-6, 6] x [-6, 6]
sage: box.clip()
[(-5, -5), (5, -5), (5, 5), (-5, 5), (-5, -5)]
```

```
sage: box = DiscreteBox([-6,6],[-4,3])
sage: box.clip()
[(-5, -3), (5, -3), (5, 2), (-5, 2), (-5, -3)]
```

class `slabbe.discrete_subset.`**DiscreteSubset**(*dimension=3*, *predicate=None*, *edge_predicate=None*, *iterator=None*, *roots=None*)

Bases: `sage.structure.sage_object.SageObject`

A subset of ZZ^d.

INPUT:

- `dimension` – integer, dimension of the space
- `predicate` – function ZZ^d -> {False, True} (default: `None`)
- `edge_predicate` – function ZZ^d,ZZ^d -> {False, True} (default: `None`)
- `iterator` – function (default: `None`) returning an iterator of points, it must be consistent with the predicate
- `roots` – list (default: `None`) of some elements in self. If `iterator` is not provided, it is used to iterate the elements throught connectedness.

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: DiscreteSubset(dimension=3)
Subset of ZZ^3
```

```
sage: p = DiscreteSubset(dimension=3, predicate=lambda x:True)
sage: p
Subset of ZZ^3
```

```
sage: fn = lambda p : p[0]+p[1]<p[2]
sage: p = DiscreteSubset(dimension=3, predicate=fn, roots=[(0,0,1)])
sage: p
Subset of ZZ^3
```

```
sage: F = lambda p: Integers(7)(2*p[0]+5*p[1])
sage: edge_predicate = lambda p,s: F(s) < F(s)
sage: D = DiscreteSubset(dimension=3, edge_predicate=edge_predicate)
sage: D
Subset of ZZ^3
```

From a list:

```
sage: L = [(0,0,0), (1,0,0), (2,0,0), (3,0,0)]
sage: s = DiscreteSubset.from_subset(L)
sage: s
Subset of ZZ^3
```

Providing a root may be necessary if zero (the origin) is not inside:

```
sage: predicate = lambda (x,y) : 4 < x^2 + y^2 < 25
sage: D = DiscreteSubset(dimension=2, predicate=predicate, roots=[(3,0)])
```

TESTS:

No edges go outside of the box:

```
sage: from slabbe import DiscreteBox
sage: B = DiscreteBox([-1,1],[-1,1])
sage: len(list(B.edges_iterator()))
12
sage: sorted(B.edges_iterator())
[((-1, -1), (-1, 0)), ((-1, -1), (0, -1)), ((-1, 0), (-1, 1)),
((-1, 0), (0, 0)), ((-1, 1), (0, 1)), ((0, -1), (0, 0)), ((0, -1),
(1, -1)), ((0, 0), (0, 1)), ((0, 0), (1, 0)), ((0, 1), (1, 1)),
((1, -1), (1, 0)), ((1, 0), (1, 1))]
```

an_element()

Returns an immutable element in self.

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: p = DiscreteSubset(dimension=3)
sage: p.an_element()
(0, 0, 0)
sage: p.an_element().is_immutable()
True
```

```
sage: predicate = lambda (x,y) : 4 < x^2 + y^2 < 25
sage: D = DiscreteSubset(dimension=2, predicate=predicate, roots=[(3,0)])
sage: D.an_element()
(3, 0)
```

base_edges()

Return a list of positive canonical vectors.

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: d = DiscreteSubset(dimension=2)
sage: d.base_edges()
[(1, 0), (0, 1)]
```

```
sage: from slabbe import DiscretePlane
sage: P = DiscretePlane([3,4,5], 12)
sage: P.base_edges()
[(1, 0, 0), (0, 1, 0), (0, 0, 1)]
```

children(*p*)

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: list(p.children(vector((0,0,0))))
[(1, 0, 0), (0, 1, 0), (0, 0, 1)]
```

connected_component_iterator(*roots=None*)

Return an iterator over the connected component of the root.

INPUT:

- `roots` - list of some elements immutable in self

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: p = DiscreteSubset(dimension=3, roots=[(0,0,0)])
sage: it = p.connected_component_iterator()
sage: [next(it) for _ in range(5)]
[(0, 0, 0), (1, 0, 0), (0, 0, 1), (0, 0, -1), (0, -1, 0)]
```

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: root = vector((0,0,0))
sage: root.set_immutable()
sage: it = p.connected_component_iterator(roots=[root])
sage: [next(it) for _ in range(5)]
[(0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1), (-1, 1, 0)]
```

d_neighbors(*p*, *d=2*)

Retourne le voisinage du point p, i.e. les points parmi les 3^d possible qui appartiennent a l'objet discret.

INPUT:

- p - un point discret
- d - integer (optional, default:2),

OUTPUT:

liste de points

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,3,7], 10)
sage: p.d_neighbors((0,0,0))
[(-1, -1, 1), (-1, 0, 1), (-1, 1, 0), (-1, 1, 1), (0, -1, 1),
(0, 0, 0), (0, 0, 1), (0, 1, 0), (1, -1, 1), (1, 0, 0), (1, 0,
1), (1, 1, 0)]
```

dimension()

Returns the dimension of the ambiant space.

OUTPUT:

integer

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: d = DiscreteSubset(dimension=3)
sage: d.dimension()
3
```

```
sage: from slabbe import DiscreteBox
sage: p = DiscreteBox([0,3], [0,3], [0,3], [0,3])
sage: p.dimension()
4
```

edges_iterator()

Returns an iterator over the pair of points in self that are adjacents, i.e. their difference is a canonical vector.

It considers only points that are connected to the given roots.

INPUT:

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: it = p.edges_iterator()
sage: next(it)
((0, 0, 0), (1, 0, 0))
sage: next(it)
((0, 0, 0), (0, 1, 0))
sage: next(it)
((0, 0, 0), (0, 0, 1))
sage: next(it)
((-1, 1, 0), (0, 1, 0))
sage: next(it)
((-2, 1, 0), (-1, 1, 0))
```

classmethod from_subset(*subset*)

Constructor from a finite subset.

INPUT:

- `subset` – iterable of integer coordinate points

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: L = [(0,0,0), (1,0,0), (2,0,0), (3,0,0)]
sage: s = DiscreteSubset.from_subset(L)
sage: s
Subset of ZZ^3
sage: all(p in s for p in s)
True
```

Note that tuple or mutable vectors work fine:

```
sage: (0,0,0) in s
True
```

(continues on next page)

(continued from previous page)

```
sage: vector((0,0,0)) in s
True
```

TESTS:

```
sage: DiscreteSubset.from_subset([])
Subset of ZZ^3
```

has_edge(*p*, *s*)

Returns whether it has the edge (p, s) where s-p is a canonical vector.

INPUT:

- p - point in the space
- s - point in the space

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: F = lambda p: Integers(7)(2*p[0]+5*p[1])
sage: edge_predicate = lambda p,s: F(p) < F(s)
sage: D = DiscreteSubset(dimension=3, edge_predicate=edge_predicate)
sage: D.has_edge(vector((0,0)),vector((1,0)))
True
sage: D.has_edge(vector((0,0)),vector((-1,0)))
True
sage: D.has_edge(vector((-1,1)),vector((1,0)))
False
```

level_iterator()

This returns an iterator of the levels according to the given roots.

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: p = DiscreteSubset(dimension=3, roots=[(0,0,0)])
sage: it = p.level_iterator()
sage: sorted(next(it))
[(0, 0, 0)]
sage: sorted(next(it))
[(-1, 0, 0), (0, -1, 0), (0, 0, -1), (0, 0, 1), (0, 1, 0), (1, 0, 0)]
sage: sorted(next(it))
[(-2, 0, 0),
 (-1, -1, 0),
 (-1, 0, -1),
 (-1, 0, 1),
 (-1, 1, 0),
 (0, -2, 0),
 (0, -1, -1),
 (0, -1, 1),
 (0, 0, -2),
 (0, 0, 2),
 (0, 1, -1),
 (0, 1, 1),
 (0, 2, 0),
 (1, -1, 0),
 (1, 0, -1),
 (1, 0, 1),
 (1, 1, 0),
 (2, 0, 0)]
```

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: it = p.level_iterator()
sage: sorted(next(it))
[(0, 0, 0)]
sage: sorted(next(it))
[(0, 0, 1), (0, 1, 0), (1, 0, 0)]
sage: sorted(next(it))
[(-1, 0, 1),
 (-1, 1, 0),
 (0, -1, 1),
 (0, 1, 1),
 (0, 2, 0),
 (1, 0, 1),
 (1, 1, 0),
 (2, 0, 0)]
```

list()

Return the list of elements in self.

EXAMPLES:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: P = DiscretePlane([3,4,5], 12, mu=20)
sage: tube = DiscreteTube([0,2],[0,2])
sage: I = P & tube
sage: sorted(I.list())
[(-3, -1, -1), (-3, -1, 0), (-2, -2, -1), (-2, -2, 0), (-2, -1,
-1), (-2, -1, 0), (-2, 0, -1), (-1, -1, -1)]
```

plot(*frame=False*, *edgecolor='blue'*, *pointcolor='blue'*)

Return a plot (2d or 3d) of the points and edges of self.

INPUT:

- `frame` - (default: False) if True, draw a bounding frame with labels
- `edgecolor` – string (default: `'blue'`), the color of the edges
- `pointcolor` – string (default: `'blue'`), the color of the points

EXAMPLES:

2d example:

```
sage: from slabbe import DiscreteBox
sage: box = DiscreteBox([-5,5],[-5,5])
sage: box.plot() # optional long
```

3d example:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: P = DiscretePlane([1,3,7], 11)
sage: tube = DiscreteTube([-5,5],[-5,5])
sage: I = P & tube
sage: I.plot() # optional long
```

plot_cubes(***kwds*)

Returns the discrete object as cubes in 3d.

EXAMPLES:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: P = DiscretePlane([3,4,5], 12, mu=20)
```

(continues on next page)

(continued from previous page)

```
sage: tube = DiscreteTube([-5,5],[-5,5])
sage: I = P & tube
sage: I.plot_cubes(color='red', frame_thickness=1 # optional long)
```

TESTS:

```
sage: from slabbe import DiscreteBox
sage: box = DiscreteBox([-5,5],[-5,5])
sage: box.plot_cubes() # optional long
Traceback (most recent call last):
...
ValueError: this method is currently implemented only for objects living in 3 dimensions
```

plot_edges(*color='blue'*, *m=None*)

Returns the mesh of the plane. The mesh is the union of segments joining two adjacents points.

INPUT:

- `color` – string (default: `'blue'`), the color of the edges
- `m` – projection matrix (default: `None`), it can be one of the following:
  - `None` - no projection is done
  - `'isometric'` - the isometric projection
  - matrix - a 2 x 3 matrix
  - `'belle'` - shortcut for `matrix(2, [1/3.0, 1, 0, 2/3.0, 0, 1])`
  - vector - defines the projection on the plane orthogonal to the vector.

EXAMPLES:

A 2d plot of a 2d object:

```
sage: from slabbe import DiscreteSubset, DiscreteBox
sage: D = DiscreteSubset(dimension=2)
sage: box = DiscreteBox([-5,5],[-5,5])
sage: I = D & box
sage: I.plot_edges(color='green') # optional long
```

A 3d plot of a 3d object:

```
sage: D = DiscreteSubset(dimension=3)
sage: box = DiscreteBox([-3,3],[-3,3],[-3,3])
sage: I = D & box
sage: I.plot_edges(color='green') # optional long
```

A 2d plot of a 3d object:

```
sage: D = DiscreteSubset(dimension=3)
sage: box = DiscreteBox([-3,3],[-3,3],[-3,3])
sage: I = D & box
sage: I.plot_edges(color='green', m='isometric') # optional long
```

plot_points(*color='blue'*, *m=None*)

Returns a 2d or 3d graphics object of the points of self.

INPUT:

- `color` – string (default: `'blue'`), the color of the points
- `m` – projection matrix (default: `None`), it can be one of the following:

   – `None` - no projection is done
   – `'isometric'` - the isometric projection
   – matrix - a 2 x n projection matrix
   – `'belle'` - shortcut for `matrix(2, [1/3.0, 1, 0, 2/3.0, 0, 1])`
   – vector - defines the projection on the plane orthogonal to the vector.

EXAMPLES:

A 2d plot of a 2d object:

```
sage: from slabbe import DiscreteSubset, DiscreteBox
sage: D = DiscreteSubset(dimension=2)
sage: box = DiscreteBox([-5,5],[-5,5])
sage: I = D & box
sage: I.plot_points(color='green')      # optional long
```

A 3d plot of a 3d object:

```
sage: D = DiscreteSubset(dimension=3)
sage: box = DiscreteBox([-5,5],[-5,5],[-5,5])
sage: I = D & box
sage: I.plot_points(color='green')      # optional long
```

A 2d plot of a 3d object:

```
sage: D = DiscreteSubset(dimension=3)
sage: box = DiscreteBox([-5,5],[-5,5],[-5,5])
sage: I = D & box
sage: I.plot_points(color='green', m='isometric')      # optional long
```

plot_points_at_distance(*k*, *color='blue'*, *projmat=None*)

Plot points at distance k from the roots.

INPUT:

- `k` - integer

EXAMPLES:

```
sage: alpha = solve(x+x**2+x**3==1, x)[2].right()
sage: vv = vector((alpha, alpha+alpha**2, 1))
sage: omega = (1+alpha)**2 / 2
sage: from slabbe import DiscretePlane
sage: Pr = DiscretePlane(vv, omega, mu=pi, prec=200)
sage: Pr.plot_points_at_distance(200)                # optional long
sage: Pr.plot_points_at_distance(200, projmat='isometric') # optional long
```

projection_matrix(*m='isometric'*, *oblique=None*)

Return a projection matrix.

INPUT:

- `m` – projection matrix (default: `'isometric'`), it can be one of the following:
   – `'isometric'` - the isometric projection is used by default
   – matrix - a 2 x 3 matrix
   – `'belle'` - shortcut for `matrix(2, [1/3.0, 1, 0, 2/3.0, 0, 1])`
   – vector - defines the projection on the plane orthogonal to the vector.
- `oblique` – vector (default: `None`), vector perpendicular to the range space

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: d = DiscreteSubset(dimension=3)
sage: d.projection_matrix(vector((2,3,4))) # tolerance 0.00001
[  1.00000000000000  0.000000000000000 -0.500000000000000]
[ 0.000000000000000   1.00000000000000 -0.750000000000000]
sage: d.projection_matrix((2,3,4)) # tolerance 0.00001
[  1.00000000000000  0.000000000000000 -0.500000000000000]
[ 0.000000000000000   1.00000000000000 -0.750000000000000]
sage: d.projection_matrix()         # tolerance 0.00001
[-0.866025403784  0.866025403784             0.0]
[           -0.5            -0.5             1.0]
sage: d.projection_matrix(_) # tolerance 0.00001
[-0.866025403784439  0.866025403784439  0.000000000000000]
[-0.500000000000000 -0.500000000000000   1.00000000000000]
sage: d.projection_matrix('belle')     # tolerance 0.00001
[0.333333333333           1.0           0.0]
[0.666666666667           0.0           1.0]
```

roots()

Return the roots, i.e., a list of elements in self.

It also makes sure the roots are in self and raises an error otherwise.

EXAMPLES:

```
sage: from slabbe import DiscreteSubset
sage: s = DiscreteSubset.from_subset([(0,0,0)])
sage: s.roots()
[(0, 0, 0)]
```

```
sage: predicate = lambda (x,y) : 4 < x^2 + y^2 < 25
sage: D = DiscreteSubset(dimension=2, predicate=predicate, roots=[(3,0)])
sage: D.roots()
[(3, 0)]
```

TESTS:

```
sage: predicate = lambda (x,y) : 4 < x^2 + y^2 < 25
sage: D = DiscreteSubset(dimension=2, predicate=predicate, roots=[(2,0)])
sage: D.roots()
Traceback (most recent call last):
...
ValueError: root element (=(2, 0)) provided at initialisation is not in self
```

An error is raised if the roots are inconsistent:

```
sage: s = DiscreteSubset.from_subset([])
sage: s.roots()
Traceback (most recent call last):
...
ValueError: default element (=(0, 0, 0)) is not in self, please provide one at initialisation
```

tikz(*projmat=[-0.866025403784439 0.866025403784439 0.000000000000000] [-0.500000000000000 -0.500000000000000 1.00000000000000], scale=1, clip=[], contour=[], edges=True, points=True, axes=False, point_kwds={}, edge_kwds={}, axes_kwds={}, extra_code=''*)

INPUT:

- `projmat` – (default: M3to2) 2 x dim projection matrix where dim is the dimensoin of self, the isometric projection is used by default
- `scale` – real number (default: 1), scaling constant for the whole figure
- `clip` - list (default: `[]`), list of points whose convex hull describes a cliping path

- contour - list (default: []), list of points describing a contour path to be drawn
- edges - bool (default: True), whether to draw edges
- points - bool (default: True), whether to draw points
- axes - bool (default: False), whether to draw axes
- point_kwds - dict (default: {})
- edge_kwds - dict (default: {})
- axes_kwds - dict (default: {})
- extra_code – string (default: ''), extra tikz code to add

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([2,3,5], 10)
sage: print(p.tikz(points=False, edges=False))
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\begin{document}
\begin{tikzpicture}
[scale=1]
\end{tikzpicture}
\end{document}
```

tikz_axes(*xshift=0*, *yshift=0*, *label='e'*, *projmat='isometric'*)

Return the tikz code for drawing axes.

INPUT:

- xshift - integer (default: 0), x shift
- yshift - integer (default: 0), y shift
- label - string (default: "e"), label for base vectors
- projmat - matrix (default: 'isometric'), projection matrix

OUTPUT:

string

EXAMPLES:

2d example:

```
sage: from slabbe import DiscreteSubset
sage: d = DiscreteSubset(dimension=2)
sage: d.tikz_axes()
%the axes
\begin{scope}[xshift=0cm,yshift=0cm]
\draw[->,>=latex, very thick, blue] (0,0) -- (1, 0);
\draw[->,>=latex, very thick, blue] (0,0) -- (0, 1);
\node at (1.40000000000000,0) {$e_1$};
\node at (0,1.40000000000000) {$e_2$};
\end{scope}
```

3d example:

```
sage: d = DiscreteSubset(dimension=3)
sage: d.tikz_axes(projmat='isometric')
%the axes
\begin{scope}
```

(continues on next page)

(continued from previous page)

```
[x={(-0.866025cm,-0.500000cm)}, y={(0.866025cm,-0.500000cm)},
z={(0.000000cm,1.000000cm)}, scale=1,xshift=0,yshift=0]
\draw[fill=white] (2,0,0) rectangle (-1.8,.1,1);
\draw[->,>=latex, very thick, blue] (0,0,0) -- (1, 0, 0);
\draw[->,>=latex, very thick, blue] (0,0,0) -- (0, 1, 0);
\draw[->,>=latex, very thick, blue] (0,0,0) -- (0, 0, 1);
\node at (1.40000000000000,0,0) {$e_1$};
\node at (0,1.40000000000000,0) {$e_2$};
\node at (0,0,1.40000000000000) {$e_3$};
\end{scope}
```

tikz_edges(*style='very thick'*, *color='blue'*, *projmat=None*)

Returns the mesh of the object. The mesh is the union of segments joining two adjacents points.

INPUT:

- `style` - string (default: `'dashed, very thick'`)
- `color` - string or callable (default: `'blue'`), the color of all edges or a function : (u,v) -> color of the edge (u,v)
- `projmat` - matrix (default: `None`), projection matrix, if None, no projection is done.

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([2,3,5], 4)
sage: p.tikz_edges()
\draw[very thick, blue] (0, 0, 0) -- (1, 0, 0);
\draw[very thick, blue] (0, 0, 0) -- (0, 1, 0);
\draw[very thick, blue] (-1, 1, 0) -- (0, 1, 0);
```

```
sage: p.tikz_edges(color='orange')
\draw[very thick, orange] (0, 0, 0) -- (1, 0, 0);
\draw[very thick, orange] (0, 0, 0) -- (0, 1, 0);
\draw[very thick, orange] (-1, 1, 0) -- (0, 1, 0);
```

```
sage: c = lambda u,v: 'red' if u == 0 else 'blue'
sage: p.tikz_edges(color=c)
\draw[very thick, red] (0, 0, 0) -- (1, 0, 0);
\draw[very thick, red] (0, 0, 0) -- (0, 1, 0);
\draw[very thick, blue] (-1, 1, 0) -- (0, 1, 0);
```

```
sage: from slabbe.discrete_subset import M3to2
sage: p.tikz_edges(projmat=M3to2)
\draw[very thick, blue] (0.00000, 0.00000) -- (-0.86603, -0.50000);
\draw[very thick, blue] (0.00000, 0.00000) -- (0.86603, -0.50000);
\draw[very thick, blue] (1.73205, 0.00000) -- (0.86603, -0.50000);
```

tikz_noprojection(*projmat=None*, *scale=1*, *clip=[]*, *edges=True*, *points=True*, *axes=False*, *point_kwds={}*, *edge_kwds={}*, *axes_kwds={}*, *extra_code=''*)

Return the tikz code of self.

In this version, the points are not projected. If the points are in 3d, the tikz 3d picture is used.

INPUT:

- `projmat` – (default: None) 2*3 projection matrix for drawing unit faces, the isometric projection is used by default
- `scale` – real number (default: 1), scaling constant for the whole figure
- `clip` - list (default: `[]`), list of points describing a cliping path once projected. Works only if `self.dimension()` is 2.

- `edges` - bool (default: `True`), whether to draw edges
- `points` - bool (default: `True`), whether to draw points
- `axes` - bool (default: `False`), whether to draw axes
- `point_kwds` - dict (default: `{}`)
- `edge_kwds` - dict (default: `{}`)
- `axes_kwds` - dict (default: `{}`)
- `extra_code` – string (default: `''`), extra tikz code to add

EXAMPLES:

Object in 2d:

```
sage: from slabbe import DiscreteLine, DiscreteBox
sage: L = DiscreteLine([2,5], 2+5, mu=0)
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = L & b
sage: point_kwds = {'label':lambda p:2*p[0]+5*p[1],'label_pos':'above right'}
sage: tikz = I.tikz_noprojection(scale=0.5,point_kwds=point_kwds)
sage: tikz
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\begin{document}
\begin{tikzpicture}
[scale=0.500000000000000]
\draw[very thick, blue] (0, 0) -- (1, 0);
\draw[very thick, blue] (0, 0) -- (0, 1);
\draw[very thick, blue] (2, 0) -- (3, 0);
...
... 40 lines not printed (2659 characters in total) ...
...
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (-5, 2) {};
\node[above right] at (-5, 2) {$0$};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (-5, 3) {};
\node[above right] at (-5, 3) {$5$};
\end{tikzpicture}
\end{document}
```

Object in 3d:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: p = DiscretePlane([1,3,7], 11)
sage: d = DiscreteTube([-5,5],[-5,5])
sage: I = p & d
sage: s = I.tikz_noprojection()
sage: s
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\begin{document}
\begin{tikzpicture}
 [x={(-0.866025cm,-0.500000cm)}, y={(0.866025cm,-0.500000cm)},
z={(0.000000cm,1.000000cm)}, scale=1]
\draw[very thick, blue] (0, 0, 0) -- (1, 0, 0);
\draw[very thick, blue] (0, 0, 0) -- (0, 1, 0);
...
... 311 lines not printed (20339 characters in total) ...
...
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (1, -4, 3) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (4, 4, -1) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (5, 3, -1) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (6, 2, -1) {};
\end{tikzpicture}
\end{document}
```

tikz_points(*size='0.8mm'*, *label=None*, *label_pos='right'*, *fill='black'*, *options="*, *filter=None*, *projmat=None*)

INPUT:

- `size` - string (default: `'0.8mm'`), size of the points
- `label` - function (default: `None`), print some label next to the point
- `label_pos` - function (default: `'right'`), tikz label position
- `fill` - string (default: `'black'`), fill color
- `options` - string (default: `''`), author tikz node circle options
- `filter` - boolean function, if filter(p) is False, the point p is not drawn
- `projmat` - matrix (default: `None`), projection matrix, if None, no projection is done.

EXAMPLES:

```
sage: from slabbe import DiscreteBox
sage: p = DiscreteBox([0,3], [0,3], [0,3])
sage: s = p.tikz_points()
sage: lines = s.splitlines()
sage: lines[0]
'\\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (...) {};'
```

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: p = DiscretePlane([1,3,7], 11)
sage: d = DiscreteTube([-1,1],[-1,1])
sage: I = p & d
sage: I.tikz_points()
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 0, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (1, 0, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 1, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 0, 1) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 1, 1) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (1, 1, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (1, 0, 1) {};
```

Using a filter on the points:

```
sage: I.tikz_points(filter=lambda x:sum(x)==1)
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (1, 0, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 1, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 0, 1) {};
```

Of a finite subset:

```
sage: from slabbe import DiscreteSubset
sage: V = [(0,0,0), (1,1,0), (1,-1,1), (-2,1,0), (2,0,1), (-1,2,0),
....:      (-1,0,1), (0,1,1)]
sage: s = DiscreteSubset.from_subset(V)
sage: s.tikz_points()
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (-2, 1, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (1, 1, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 1, 1) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (-1, 2, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (-1, 0, 1) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (0, 0, 0) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (1, -1, 1) {};
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (2, 0, 1) {};
```

tikz_projection_scale(*projmat='isometric'*, *scale=1*, *extra="*)

INPUT:

- `projmat` – (default: `'isometric'`) It can be one of the following:
  - `'isometric'` - the isometric projection is used by default
  - matrix - a 2 x 3 matrix
  - `'belle'` - shortcut for `matrix(2, [1/3.0, 1, 0, 2/3.0, 0, 1])`
  - vector - defines the projection on the plane orthogonal to the vector.
- `scale` – real number (default: 1), scaling constant for the whole figure
- `extra` – string (default: `''`)

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,3,7], 11)
sage: p.tikz_projection_scale()
[x={(-0.866025cm,-0.500000cm)}, y={(0.866025cm,-0.500000cm)},
z={(0.000000cm,1.000000cm)}, scale=1]
sage: p.tikz_projection_scale(extra="xshift=4cm")
[x={(-0.866025cm,-0.500000cm)}, y={(0.866025cm,-0.500000cm)},
z={(0.000000cm,1.000000cm)}, scale=1,xshift=4cm]
```

class `slabbe.discrete_subset.`**`DiscreteTube`**(*projmat=[-0.866025403784439 0.866025403784439 0.000000000000000] [-0.500000000000000 -0.500000000000000 1.00000000000000], *args, **kwds*)

Bases: *`slabbe.discrete_subset.DiscreteSubset`*

Discrete Tube (preimage of a box by a projection matrix)

Subset of a discrete object such that its projection by a matrix is inside a certain box.

INPUT:

- `*args` - intervals, lists of size two : [min, max]
- `projmat` - matrix (default: `M3to2`), projection matrix

EXAMPLES:

```
sage: from slabbe import DiscreteTube
sage: DiscreteTube([-5,5],[-5,5])
DiscreteTube: Preimage of [-5, 5] x [-5, 5] by a 2 by 3 matrix
```

```
sage: m = matrix(3,4,range(12))
sage: DiscreteTube([2,10],[3,4],[6,7], projmat=m)
DiscreteTube: Preimage of [2, 10] x [3, 4] x [6, 7] by a 3 by 4 matrix
```

EXAMPLES:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: tube = DiscreteTube([-5,5],[-5,5])
sage: I = p & tube
sage: I
Intersection of the following objects:
Set of points x in ZZ^3 satisfying: 0 <= (1, pi, 7) . x + 0 < pi + 8
DiscreteTube: Preimage of [-5, 5] x [-5, 5] by a 2 by 3 matrix
sage: len(list(I))
115
```

`clip`(*space=1*)

Return a good clip rectangle for this box.

INPUT:

- `space` – number (default: 1), inner space within the box

EXAMPLES:

```
sage: from slabbe import DiscreteTube
sage: tube = DiscreteTube([-6,6],[-4,3])
sage: tube.clip()
[(-5, -3), (5, -3), (5, 2), (-5, 2), (-5, -3)]
```

class `slabbe.discrete_subset.`**`Intersection`**(*objets*)

Bases: *`slabbe.discrete_subset.DiscreteSubset`*

Intersection

todo:

- Rendre l'heritage 3d automatique

INPUT:

- `objets` - un tuple d'objets discrets

EXAMPLES:

Intersection de deux plans:

```
sage: from slabbe import DiscretePlane, Intersection
sage: p = DiscretePlane([1,3,7],11)
sage: q = DiscretePlane([1,3,5],9)
sage: Intersection((p,q))
Intersection of the following objects:
Set of points x in ZZ^3 satisfying: 0 <= (1, 3, 7) . x + 0 < 11
Set of points x in ZZ^3 satisfying: 0 <= (1, 3, 5) . x + 0 < 9
```

Shortcut:

```
sage: p & q
Intersection of the following objects:
Set of points x in ZZ^3 satisfying: 0 <= (1, 3, 7) . x + 0 < 11
Set of points x in ZZ^3 satisfying: 0 <= (1, 3, 5) . x + 0 < 9
```

Intersection of a plane and a tube:

```
sage: from slabbe import DiscreteTube
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: d = DiscreteTube([-5,5],[-5,5])
sage: I = p & d
sage: I
Intersection of the following objects:
Set of points x in ZZ^3 satisfying: 0 <= (1, pi, 7) . x + 0 < pi + 8
DiscreteTube: Preimage of [-5, 5] x [-5, 5] by a 2 by 3 matrix
sage: len(list(I))
115
```

Intersection of a line and a box:

```
sage: from slabbe import DiscreteLine, DiscreteBox
sage: L = DiscreteLine([2,5], 2+5, mu=0)
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = L & b
sage: I
Intersection of the following objects:
Set of points x in ZZ^2 satisfying: 0 <= (2, 5) . x + 0 < 7
[-5, 5] x [-5, 5]
```

TESTS:

Intersected objects must be of the same dimension:

```
sage: box = DiscreteBox([-5,5],[-5,5])
sage: p = DiscretePlane([1,pi,7], 1+pi+7)
sage: p & box
Traceback (most recent call last):
...
ValueError: Intersection not defined for objects not of the same dimension
```

an_element()

Returns an element in self.

EXAMPLES:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: P = DiscretePlane([4,6,7], 17, mu=0)
sage: tube = DiscreteTube([-6.4, 6.4], [-5.2, 5.2])
sage: I = tube & P
sage: I.an_element()
(0, 0, 0)
sage: I.an_element() in I
True
```

TESTS:

```
sage: P = DiscretePlane([4,6,7], 17, mu=0)
sage: def contain(p): return 0 < P._v.dot_product(p) + P._mu <= P._omega
sage: P._predicate = contain
sage: tube = DiscreteTube([-6.4, 6.4], [-5.2, 5.2])
sage: I = tube & P
sage: I.an_element()
(0, 0, 0) not in the plane
trying similar points
(0, 0, 1)
```

has_edge(*p*, *s*)

Returns whether it has the edge (p, s) where s-p is a canonical vector.

INPUT:

- p - point in the space
- s - point in the space

EXAMPLES:

```
sage: from slabbe import DiscretePlane, DiscreteSubset
sage: d3 = DiscreteSubset(dimension=3)
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: I = p & d3
sage: I.has_edge(vector((0,0,0)),vector((0,0,1)))
True
sage: I.has_edge(vector((0,0,0)),vector((0,0,-1)))
False
```

TESTS:

```
sage: from slabbe import DiscreteBox
sage: F = lambda p: (2*p[0]+5*p[1]) % 7
sage: edge_predicate = lambda p,s: F(p) < F(s)
sage: D = DiscreteSubset(dimension=2, edge_predicate=edge_predicate)
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = D & b
```

(continues on next page)

(continued from previous page)

```
sage: all(I.has_edge(a,b) for a,b in I.edges_iterator())
True
sage: all(D.has_edge(a,b) for a,b in I.edges_iterator())
True
```

```
sage: from slabbe import ChristoffelGraph
sage: C = ChristoffelGraph((2,5))
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = C & b
sage: all(I.has_edge(a,b) for a,b in I.edges_iterator())
True
sage: all(C.has_edge(a,b) for a,b in I.edges_iterator())
True
```

roots()

EXAMPLES:

```
sage: from slabbe import DiscreteBox, DiscreteSubset
sage: d3 = DiscreteSubset(dimension=3, roots=[(0,0,0), (1,1,1)])
sage: box = DiscreteBox([-5,5],[-5,5],[-5,5])
sage: I = d3 & box
sage: sorted(d3.roots())
[(0, 0, 0), (1, 1, 1)]
sage: box.roots()
[(0, 0, 0)]
sage: sorted(I.roots())
[(0, 0, 0), (1, 1, 1)]
```

slabbe.discrete_subset.**convex_boundary**(*L*)

EXAMPLES:

```
sage: from slabbe.discrete_subset import convex_boundary
sage: convex_boundary([(3,4), (1,2), (3,5)])
[(3, 5), (1, 2), (3, 4)]
```

1.2 Discrete Plane

Discrete Hyperplanes

Intersection of a plane and a tube:

```
sage: from slabbe import DiscretePlane, DiscreteTube
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: d = DiscreteTube([-5,5],[-5,5])
sage: I = p & d
sage: I
Intersection of the following objects:
Set of points x in ZZ^3 satisfying: 0 <= (1, pi, 7) . x + 0 < pi + 8
DiscreteTube: Preimage of [-5, 5] x [-5, 5] by a 2 by 3 matrix
sage: len(list(I))
115
```

Intersection of a line and a box:

```
sage: from slabbe import DiscreteLine, DiscreteBox
sage: L = DiscreteLine([pi,sqrt(2)], pi+sqrt(2), mu=0)
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = L & b
sage: I
```

(continues on next page)

(continued from previous page)

```
Intersection of the following objects:
Set of points x in ZZ^2 satisfying: 0 <= (pi, sqrt(2)) . x + 0 < pi + sqrt(2)
[-5, 5] x [-5, 5]
```

TODO:

- do some dimension checking for DiscreteLine and DiscretePlane

class slabbe.discrete_plane.**DiscreteHyperplane**(*v*, *omega*, *mu=0*, *prec=None*)

Bases: *slabbe.discrete_subset.DiscreteSubset*

This is the set of point p such that

$$0 \leq p \cdot v - mu < \omega$$

INPUT:

- v - normal vector
- omega - width
- mu - intercept (optional, default: 0)

EXAMPLES:

```
sage: from slabbe import DiscreteLine
sage: L = DiscreteLine([pi,sqrt(2)], pi+sqrt(2), mu=10)
sage: L
Set of points x in ZZ^2 satisfying: 0 <= (pi, sqrt(2)) . x + 10 < pi + sqrt(2)
```

```
sage: from slabbe import DiscretePlane
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: p
Set of points x in ZZ^3 satisfying: 0 <= (1, pi, 7) . x + 0 < pi + 8
```

```
sage: from slabbe import DiscreteHyperplane
sage: p = DiscreteHyperplane([1,3,7,9], 20, mu=13)
sage: p
Set of points x in ZZ^4 satisfying: 0 <= (1, 3, 7, 9) . x + 13 < 20
```

TESTS:

```
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=20)
sage: vector((0,0,0)) in p
False
sage: p = DiscretePlane([1,pi,7], 1+pi+7, mu=0)
sage: vector((0,0,0)) in p
True
```

```
sage: p = DiscreteHyperplane((2,3,4,5), 10)
sage: p.dimension()
4
```

```
sage: L = DiscreteLine([1,pi], 1+pi, mu=20)
sage: vector((0,0)) in L
False
sage: L = DiscreteLine([1,pi], 1+pi, mu=0)
sage: vector((0,0)) in L
True
```

an_element(*x=0*, *y=0*)

Returns an element in self.

EXAMPLES:

```
sage: from slabbe import DiscreteHyperplane
sage: p = DiscreteHyperplane([1,pi,7], 1+pi+7, mu=10)
sage: p.an_element()
(0, 0, 0)
```

```
sage: from slabbe import DiscreteLine
sage: L = DiscreteLine([pi,sqrt(2)], pi+sqrt(2), mu=10)
sage: L.an_element()
(-2, -2)
```

```
sage: L = DiscreteLine([pi,sqrt(2)], pi+sqrt(2), mu=0)
sage: L.an_element()
(0, 0)
```

level_value(*p*)

Return the level value of a point p.

INPUT:

- p - point in the space

EXAMPLES:

```
sage: from slabbe import DiscreteHyperplane
sage: H = DiscreteHyperplane([1,3,7,9], 20, mu=13)
sage: p = H._space((1,2,3,4))
sage: H.level_value(p)
64
```

roots()

Return the roots, i.e., a list of elements in self.

It also makes sure the roots are in self and raises an error otherwise.

EXAMPLES:

```
sage: from slabbe import DiscretePlane
sage: P = DiscretePlane([3,4,5], 12, mu=20)
sage: P.roots()
[(-1, -1, -1)]
sage: all(p in P for p in P.roots())
True
```

slabbe.discrete_plane.**DiscreteLine**

alias of *slabbe.discrete_plane.DiscreteHyperplane*

slabbe.discrete_plane.**DiscretePlane**

alias of *slabbe.discrete_plane.DiscreteHyperplane*

1.3 Discrete Lines

Billiard words

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: b
Cubic billiard of direction (1, pi, sqrt(2))
```

TODO:

- Rewrite some parts in cython because it is slow
- Should handle any direction
- Should use Forest structure for enumeration
- Should use +e_i only for children
- Fix documentation of class
- Fix issue with the assertion error in the step iterator
- not robust for non integral start point

class slabbe.billiard.**BilliardCube**(*v*, *start=(0, 0, 0)*)

Bases: *slabbe.discrete_subset.Intersection*

This is the set of point p such that

$$0 \leq p \cdot v - mu < \omega \text{ #fix me } 0 \leq p \cdot v - mu < \omega \text{ #fix me } 0 \leq p \cdot v - mu < \omega \text{ #fix me}$$

INPUT:

- v - directive vector
- `start` - initial point (default = (0,0,0))

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: b
Cubic billiard of direction (1, pi, sqrt(2))
```

```
   sage: b = BilliardCube((1,pi,sqrt(2)))
   sage: it = iter(b)
   sage: [next(it) for _ in range(20)]
   [(0, 0, 0),
    (0, 1, 0),
    (0, 1, 1),
    (0, 2, 1),
    (1, 2, 1),
    (1, 3, 1),
    (1, 3, 2),
    (1, 4, 2),
    (1, 5, 2),
    (2, 5, 2),
    (2, 6, 2),
    (2, 6, 3),
    (2, 7, 3),
    (2, 8, 3),
    (2, 8, 4),
    (3, 8, 4),
    (3, 9, 4),
    (3, 10, 4),
    (3, 10, 5),
    (3, 11, 5)]

::

    sage: b = BilliardCube((1,sqrt(2),pi), start=(11,13,14))
    sage: b.to_word()
    word: 3231323313233213323132331233321332313233...
```

an_element()

Returns an element in self.

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: b.an_element()
(0, 0, 0)
```

children(*p*)

Return the children of a point.

This method overwrites the methods *slabbe.discrete_subset.DiscreteSubset.children()*, because for billiard words, we go only in one direction in each axis.

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: list(b.children(vector((0,0,0))))
[(0, 1, 0)]
```

connected_component_iterator(*roots=None*)

Return an iterator over the connected component of the root.

This method overwrites the methods *slabbe.discrete_subset.DiscreteSubset.connected_component_iterator()*, because for billiard words, we go only in one direction in each axis which allows to use a forest structure for the enumeration.

INPUT:

- `roots` - list of some elements immutable in self

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: p = BilliardCube([1,pi,sqrt(7)])
sage: root = vector((0,0,0))
sage: root.set_immutable()
sage: it = p.connected_component_iterator(roots=[root])
sage: [next(it) for _ in range(5)]
[(0, 0, 0), (0, 1, 0), (0, 1, 1), (0, 2, 1), (1, 2, 1)]
```

```
sage: p = BilliardCube([1,pi,7.45], start=(10.2,20.4,30.8))
sage: it = p.connected_component_iterator()
sage: [next(it) for _ in range(5)]
[(10.2000000000000, 20.4000000000000, 30.8000000000000),
 (10.2000000000000, 20.4000000000000, 31.8000000000000),
 (10.2000000000000, 21.4000000000000, 31.8000000000000),
 (10.2000000000000, 21.4000000000000, 32.8000000000000),
 (10.2000000000000, 21.4000000000000, 33.8000000000000)]
```

step_iterator()

Return an iterator coding the steps of the discrete line.

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: it = b.step_iterator()
sage: [next(it) for _ in range(5)]
[(0, 1, 0), (0, 0, 1), (0, 1, 0), (1, 0, 0), (0, 1, 0)]
```

TESTS:

Fix this:

```
sage: from slabbe import BilliardCube
sage: B = BilliardCube((1.1,2.2,3.3))
sage: B.to_word()
Traceback (most recent call last):
...
AssertionError: step(=(-1, 0, 1)) is not a canonical basis
vector.
```

to_word(*alphabet=[1, 2, 3]*)

Return the billiard word.

INPUT:

- `alphabet` - list

EXAMPLES:

```
sage: from slabbe import BilliardCube
sage: b = BilliardCube((1,pi,sqrt(2)))
sage: b.to_word()
word: 2321232212322312232123221322231223212322...
```

```
sage: B = BilliardCube((sqrt(3),sqrt(5),sqrt(7)))
sage: B.to_word()
word: 3213213231232133213213231232132313231232...
```

1.4 Christoffel Graph

Christoffel graph

This module was developped for the article on a d-dimensional extension of Christoffel Words written with Christophe Reutenauer *[LR2014]*.

EXAMPLES:

Christoffel graph in 2d (tikz code):

```
sage: from slabbe import ChristoffelGraph, DiscreteBox
sage: C = ChristoffelGraph((2,5))
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = C & b
sage: point_kwds = {'label':lambda p:C.level_value(p),'label_pos':'above right'}
sage: tikz = I.tikz_noprojection(scale=0.8,point_kwds=point_kwds)
```

Christoffel graph in 3d (tikz code):

```
sage: C = ChristoffelGraph((2,3,5))
sage: tikz = C.tikz_kernel()
```

TODO:

- Clean kernel_vector method of ChristoffelGraph

class slabbe.christoffel_graph.**ChristoffelGraph**(*v*, *mod=None*)

Bases: *slabbe.discrete_subset.DiscreteSubset*

Subset of a discrete object such that its projection by a matrix is inside a certain box.

INPUT:

- v - vector, normal vector

EXAMPLES:

```
sage: from slabbe import ChristoffelGraph
sage: ChristoffelGraph((2,5))
Christoffel set of edges for normal vector v=(2, 5)
```

```
sage: C = ChristoffelGraph((2,5))
sage: it = C.edges_iterator()
sage: it.next()
((0, 0), (1, 0))
```

```
sage: C = ChristoffelGraph((2,5,8))
sage: it = C.edges_iterator()
sage: it.next()
((0, 0, 0), (1, 0, 0))
```

```
sage: from slabbe import DiscreteBox
sage: C = ChristoffelGraph((2,5))
sage: b = DiscreteBox([-5,5],[-5,5])
sage: I = C & b
sage: point_kwds = {'label':lambda p:C.level_value(p),'label_pos':'above right'}
sage: tikz = I.tikz_noprojection(scale=0.8,point_kwds=point_kwds)
```

TEST:

This was once a bug. We make sure it is fixed:

```
sage: from slabbe import DiscreteSubset
sage: C = ChristoffelGraph((2,3,5))
sage: isinstance(C, DiscreteSubset)
True
```

has_edge(*p*, *s*)

Returns whether it has the edge (p, s) where s-p is a canonical vector.

INPUT:

- p - point in the space
- s - point in the space

EXAMPLES:

```
sage: from slabbe import ChristoffelGraph
sage: C = ChristoffelGraph((2,5,8))
sage: C.has_edge(vector((0,0,0)), vector((0,0,1)))
True
sage: C.has_edge(vector((0,0,0)), vector((0,0,2)))
False
sage: C.has_edge(vector((0,0,0)), vector((0,0,-1)))
False
```

```
sage: C = ChristoffelGraph((2,5))
sage: C.has_edge(vector((0,0)),vector((1,0)))
True
sage: C.has_edge(vector((0,0)),vector((-1,0)))
False
sage: C.has_edge(vector((-1,1)),vector((1,0)))
False
```

kernel_vector(*way='LLL'*, *verbose=False*)

todo: clean this

EXAMPLES:

```
sage: from slabbe import ChristoffelGraph
sage: C = ChristoffelGraph((2,5,7))
sage: C.kernel_vector()
[(-1, -1, 1), (3, -4, 0)]
```

level_value(*p*)

Return the level value of a point p.

INPUT:

- p - point in the space

EXAMPLES:

```
sage: from slabbe import ChristoffelGraph
sage: C = ChristoffelGraph((2,5,8))
sage: C.level_value(vector((2,3,4)))
6
sage: C.level_value(vector((1,1,1)))
0
```

tikz_kernel(*projmat=[-0.866025403784439 0.866025403784439 0.000000000000000] [-0.500000000000000 -0.500000000000000 1.00000000000000], scale=1, edges=True, points=True, label=False, point_kwds={}, edge_kwds={}, extra_code=", way='LLL', kernel_vector=None*)

INPUT:

- `projmat` – (default: M3to2) 2 x dim projection matrix where dim is the dimensoin of self, the isometric projection is used by default
- `scale` – real number (default: 1), scaling constant for the whole figure
- `edges` - bool (optional, default: `True`), whether to draw edges
- `points` - bool (optional, default: `True`), whether to draw points
- `point_kwds` - dict (default: `{}`)
- `edge_kwds` - dict (default: `{}`)
- `extra_code` – string (default: `''`), extra tikz code to add
- `way` – string (default: `'LLL'`), the way the base of the kernel is computed
- `kernel_vector` – list (default: `None`), the vectors, if None it uses `kernel_vector()` output.

EXAMPLES:

```
sage: from slabbe import ChristoffelGraph
sage: C = ChristoffelGraph((2,3,5))
sage: tikz = C.tikz_kernel()
sage: tikz
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\begin{document}
\begin{tikzpicture}
[scale=1]
\clip (-5.239453693, 2.025) -- (-5.239453693, -1.025) -- (-4.330127019, -1.55) -- (0.0, -1.05) --
↪(0.909326674, -0.525) -- (0.909326674, 2.525) -- (0.0, 3.05) -- (-4.330127019, 2.55) -- cycle;
 \draw[very thick, blue] (0.00000, 0.00000) -- (-0.86603, -0.50000);
\draw[very thick, blue] (0.00000, 0.00000) -- (0.86603, -0.50000);
...
... 296 lines not printed (24077 characters in total) ...
...
\node[circle,fill=black,draw=black,minimum size=0.8mm,inner sep=0pt,] at (-6.92820, -3.00000) {};
```

(continues on next page)

(continued from previous page)

```
 \filldraw[fill=white,very thick,dotted,opacity=0.5,even odd rule]
  (-5.239453693, 2.025) -- (-5.239453693, -1.025) -- (-4.330127019, -1.55) -- (0.0, -1.05) -- (0.
↪909326674, -0.525) -- (0.909326674, 2.525) -- (0.0, 3.05) -- (-4.330127019, 2.55) -- cycle
  (-4.330127019, 1.5) -- (-4.330127019, -0.5) -- (0.0, 0.0) -- (0.0, 2.0) -- cycle;
 \end{tikzpicture}
\end{document}
```

1.5 Double Square Tiles

Double Square tiles

If a polyomino P tiles the plane by translation, then there exists a regular tiling of the plane by P *[WVL1984]*, i.e., where the set of translations forms a lattice. Such a polyomino was called *exact* by Wijshoff and van Leeuven. There are two types of regular tiling of the plane : square and hexagonal. These are characterized by the Beauquier-Nivat condition *[BN1991]*. Deciding whether a polyomino is exact can be done efficiently from the boundary and in linear time for square tiling *[BFP2009]*. Brlek, Fédou, Provençal also remarked that there exist polyominoes leading to more than one regular tilings but conjectured that any polyomino produces at most two regular square tilings. This conjecture was proved in *[BBL2012]*. In *[BBGL2011]*, two infinite families of *double square tiles* were provided, that is polyominoes having exactly two distinct regular square tilings of the plane, namely the Christoffel tiles and the Fibonacci tiles. Finally, in *[BGL2012]*, it was shown that any double square tile can be constructed using two simple combinatorial rules: EXTEND and SWAP.

This module is about double square tiles. Notations are chosen according to *[BGL2012]*. It allows to construct, study and show double square tiles. Operations TRIM, SWAP and EXTEND are implemented. Double square tiles can be shown using Sage 2D Graphics objects or using tikz.

REFERENCES:

AUTHORS:

- Sébastien Labbé, 2008: initial version
- Alexandre Blondin Massé, 2008: initial version
- Sébastien Labbé, March 2013: rewrite for inclusion into Sage

EXAMPLES:

Double Square tile from the boundary word of a known double square:

```
sage: from slabbe import DoubleSquare
sage: DoubleSquare(words.fibonacci_tile(2))
Double Square Tile
  w0 = 32303010   w4 = 10121232
  w1 = 30323      w5 = 12101
  w2 = 21232303   w6 = 03010121
  w3 = 23212      w7 = 01030
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
(d0, d1, d2, d3)         = (10, 16, 10, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
```

```
sage: from slabbe import christoffel_tile
sage: DoubleSquare(christoffel_tile(4,7))
Double Square Tile
  w0 = 03                          w4 = 21
  w1 = 0103010103010301010301030   w5 = 2321232321232123232123212
  w2 = 10103010                    w6 = 32321232
  w3 = 1                           w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (2, 25, 8, 1)
```

(continues on next page)

(continued from previous page)

```
(d0, d1, d2, d3)         = (26, 10, 26, 10)
(n0, n1, n2, n3)         = (0, 2, 0, 0)
```

Double Square tile from the lengths of the w_i:

```
sage: DoubleSquare((4,7,4,7))
Double Square Tile
  w0 = 3232      w4 = 1010
  w1 = 1212323   w5 = 3030101
  w2 = 2121      w6 = 0303
  w3 = 0101212   w7 = 2323030
(|w0|, |w1|, |w2|, |w3|) = (4, 7, 4, 7)
(d0, d1, d2, d3)         = (14, 8, 14, 8)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
```

DoubleSquare tile from the words (w_0, w_1, w_2, w_3):

```
sage: DoubleSquare(([3,2], [3], [0,3], [0,1,0,3,0]))
Double Square Tile
  w0 = 32              w4 = 10
  w1 = 3               w5 = 1
  w2 = 03              w6 = 21
  w3 = 01030           w7 = 23212
(|w0|, |w1|, |w2|, |w3|) = (2, 1, 2, 5)
(d0, d1, d2, d3)         = (6, 4, 6, 4)
(n0, n1, n2, n3)         = (0, 0, 0, 1)
```

Reduction of a double square tile:

```
sage: D = DoubleSquare(christoffel_tile(4,7))
sage: D.reduction()
['TRIM_1', 'TRIM_1', 'TRIM_2', 'TRIM_1', 'TRIM_0', 'TRIM_2']
sage: D.apply_reduction()
Double Square Tile
  w0 =     w4 =
  w1 = 0   w5 = 2
  w2 =     w6 =
  w3 = 1   w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (0, 1, 0, 1)
(d0, d1, d2, d3)         = (2, 0, 2, 0)
(n0, n1, n2, n3)         = (0, NaN, 0, NaN)
```

The intermediate steps of the reduction of a double square tile:

```
sage: E,op = D.reduce()
sage: E
Double Square Tile
  w0 = 03                w4 = 21
  w1 = 010301010301030   w5 = 232123232123212
  w2 = 10103010          w6 = 32321232
  w3 = 1                 w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (2, 15, 8, 1)
(d0, d1, d2, d3)         = (16, 10, 16, 10)
(n0, n1, n2, n3)         = (0, 1, 0, 0)
sage: op
'TRIM_1'
```

```
sage: D.reduce_ntimes(3)
Double Square Tile
  w0 = 03      w4 = 21
  w1 = 01030   w5 = 23212
  w2 = 10      w6 = 32
  w3 = 1       w7 = 3
```

(continues on next page)

(continued from previous page)

```
(|w0|, |w1|, |w2|, |w3|) = (2, 5, 2, 1)
(d0, d1, d2, d3)         = (6, 4, 6, 4)
(n0, n1, n2, n3)         = (0, 1, 0, 0)
```

Plot a double square tile and plot its reduction:

```
sage: D = DoubleSquare((34,21,34,21))
sage: _ = D.plot()              # long time (1s)
sage: _ = D.plot_reduction()    # long time (1s)
```

It is not said clear enough in the articles, but double square reduction also works for double square tiles that are 8-connected polyominoes:

```
sage: D = DoubleSquare((55,34,55,34))
sage: _ = D.plot()              # long time (1s)
sage: _ = D.plot_reduction()    # long time (1s)
```

class `slabbe.double_square_tile.`**`DoubleSquare`**(*data*, *rot180=None*, *steps=None*)

Bases: `sage.structure.sage_object.SageObject`

A double square tile.

We represent a double square tile by its boundary, that is a finite sequence on the alphabet $A = \{0, 1, 2, 3\}$ where 0 is a East step, 1 is a North step, 2 is a West step and 3 is a South step.

INPUT:

- `data` - can be one of the following:
  - word - word over over the alphabet A representing the boundary of a double square tile
  - tuple - tuple of 4 elements (w0,w1,w2,w3) or 8 elements (w0,w1,w2,w3,w4,w5,w6,w7) such that each wi is a sequence over the alphabet A. The condition $w_i w_{i+1} = hat(w_{i+4} w_{i+5})$ must be verified for all i modulo 8.
  - tuple - tuple of 4 integers, the lengths of (w0,w1,w2,w3)
- `rot180` - WordMorphism (default: None), involution on the alphabet A and representing a rotation of 180 degrees. If None, the morphism 0->2, 1->3, 2->0, 3->1 is considered.
- `steps` - dict (default: None), mapping letters of A to steps in the plane. If None, the corresondance 0->(1,0), 1->(0,1), 2->(-1,0), 3->(0,-1) is considered.

EXAMPLES:

From a double square:

```
sage: from slabbe import DoubleSquare
sage: DoubleSquare(words.fibonacci_tile(1))
Double Square Tile
  w0 = 32   w4 = 10
  w1 = 3    w5 = 1
  w2 = 03   w6 = 21
  w3 = 0    w7 = 2
(|w0|, |w1|, |w2|, |w3|) = (2, 1, 2, 1)
(d0, d1, d2, d3)         = (2, 4, 2, 4)
(n0, n1, n2, n3)         = (1, 0, 1, 0)
sage: DoubleSquare(words.fibonacci_tile(2))
Double Square Tile
  w0 = 32303010   w4 = 10121232
  w1 = 30323      w5 = 12101
  w2 = 21232303   w6 = 03010121
  w3 = 23212      w7 = 01030
```

(continues on next page)

(continued from previous page)

```
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
(d0, d1, d2, d3)         = (10, 16, 10, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
```

```
sage: from slabbe import christoffel_tile
sage: DoubleSquare(christoffel_tile(9,7))
Double Square Tile
  w0 = 03                          w4 = 21
  w1 = 0101030101030101030         w5 = 2323212323212323212
  w2 = 101010301010301010301010  w6 = 323232123232123232123232
  w3 = 1                           w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (2, 19, 24, 1)
(d0, d1, d2, d3)         = (20, 26, 20, 26)
(n0, n1, n2, n3)         = (0, 0, 1, 0)
```

From the w_i:

```
sage: D = DoubleSquare(([],[],[0,1,0,1],[0,1]))
sage: D.rot180
WordMorphism: 0->2, 1->3, 2->0, 3->1
sage: D._steps
{0: (1, 0), 1: (0, 1), 2: (-1, 0), 3: (0, -1)}
sage: D
Double Square Tile
  w0 =        w4 =
  w1 =        w5 =
  w2 = 0101   w6 = 3232
  w3 = 01     w7 = 32
(|w0|, |w1|, |w2|, |w3|) = (0, 0, 4, 2)
(d0, d1, d2, d3)         = (2, 4, 2, 4)
(n0, n1, n2, n3)         = (0, 0, 2, 0)
```

One may also provide strings as long as other arguments are consistent:

```
sage: steps = {'0':(1,0), '1':(0,1), '2':(-1,0), '3': (0,-1)}
sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: DoubleSquare(('','','0101','01','','','3232','32'), rot180, steps)
Double Square Tile
  w0 =        w4 =
  w1 =        w5 =
  w2 = 0101   w6 = 3232
  w3 = 01     w7 = 32
(|w0|, |w1|, |w2|, |w3|) = (0, 0, 4, 2)
(d0, d1, d2, d3)         = (2, 4, 2, 4)
(n0, n1, n2, n3)         = (0, 0, 2, 0)
```

The first four words wi are sufficient:

```
sage: steps = {'0':(1,0), '1':(0,1), '2':(-1,0), '3': (0,-1)}
sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: DoubleSquare(('','','0101','01'), rot180, steps)
Double Square Tile
  w0 =        w4 =
  w1 =        w5 =
  w2 = 0101   w6 = 3232
  w3 = 01     w7 = 32
(|w0|, |w1|, |w2|, |w3|) = (0, 0, 4, 2)
(d0, d1, d2, d3)         = (2, 4, 2, 4)
(n0, n1, n2, n3)         = (0, 0, 2, 0)
```

alphabet()

Returns the python set of the letters that occurs in the boundary word.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.alphabet()
{0, 1, 2, 3}
```

apply(*L*)

Return the double square obtained after the application of a list of operations.

INPUT:

- L - list, list of strings

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.apply(['SWAP_0', 'EXTEND_3', 'TRIM_3'])
Double Square Tile
  w0 = 01030323      w4 = 23212101
  w1 = 21232303010   w5 = 03010121232
  w2 = 30323212      w6 = 12101030
  w3 = 10121232303   w7 = 32303010121
(|w0|, |w1|, |w2|, |w3|) = (8, 11, 8, 11)
(d0, d1, d2, d3)         = (22, 16, 22, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
```

```
sage: D.apply(D.reduction())
Double Square Tile
  w0 =     w4 =
  w1 = 3   w5 = 1
  w2 =     w6 =
  w3 = 2   w7 = 0
(|w0|, |w1|, |w2|, |w3|) = (0, 1, 0, 1)
(d0, d1, d2, d3)         = (2, 0, 2, 0)
(n0, n1, n2, n3)         = (0, NaN, 0, NaN)
```

apply_morphism(*m*)

INPUT:

- m - a WordMorphism

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: m = WordMorphism({0:[0],1:[1,0,1],2:[2],3:[3,2,3]})
sage: D.apply_morphism(m)
Double Square Tile
  w0 = 3232   w4 = 1010
  w1 = 323    w5 = 101
  w2 = 0323   w6 = 2101
  w3 = 0      w7 = 2
(|w0|, |w1|, |w2|, |w3|) = (4, 3, 4, 1)
(d0, d1, d2, d3)         = (4, 8, 4, 8)
(n0, n1, n2, n3)         = (1, 0, 1, 0)
```

apply_reduction()

Apply the reduction algorithm on self.

This is equivalent to `self.apply(self.reduction())`.

EXAMPLES:

```
sage: from slabbe import DoubleSquare, christoffel_tile
sage: D = DoubleSquare(christoffel_tile(9,7))
sage: D.apply_reduction()
Double Square Tile
  w0 =      w4 =
  w1 = 0    w5 = 2
  w2 =      w6 =
  w3 = 1    w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (0, 1, 0, 1)
(d0, d1, d2, d3)         = (2, 0, 2, 0)
(n0, n1, n2, n3)         = (0, NaN, 0, NaN)
```

```
sage: D = DoubleSquare((5,7,4,13))
sage: D.apply_reduction()
Double Square Tile
  w0 =      w4 =
  w1 =      w5 =
  w2 = 1    w6 = 0
  w3 =      w7 =
(|w0|, |w1|, |w2|, |w3|) = (0, 0, 1, 0)
(d0, d1, d2, d3)         = (0, 1, 0, 1)
(n0, n1, n2, n3)         = (NaN, 0, NaN, 0)
```

```
sage: D = DoubleSquare((5,2,4,13))
sage: D.reduce_ntimes(3)
Double Square Tile
  w0 = 0     w4 = 0
  w1 = 12    w5 = 32
  w2 = 01    w6 = 03
  w3 = 2     w7 = 2
(|w0|, |w1|, |w2|, |w3|) = (1, 2, 2, 1)
(d0, d1, d2, d3)         = (3, 3, 3, 3)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.apply_reduction()
Traceback (most recent call last):
...
ValueError: not reducible, because self is nondegenerate and
d_0 == d_1 == 3. Also, the turning number (=0) must be +1 or -1
for the reduction to apply.
```

boundary_word()

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.boundary_word()
Path: 3230301030323212323032321210121232121010...
```

d(i)

Return the integer d_i.

The value of d_i is defined as $d_i = |w_{i-1}| + |w_{i+1}|$.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: [D.d(i) for i in range(8)]
[10, 16, 10, 16, 10, 16, 10, 16]
```

extend(i)

Apply $EXTEND_i$ on self.

This adds a period of length d_i to w_i and w_{i+4}.

INPUT:

- i - integer

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.extend(3)
Double Square Tile
  w0 = 32303010                w4 = 10121232
  w1 = 30323                   w5 = 12101
  w2 = 21232303                w6 = 03010121
  w3 = 23212101212323032 3212   w7 = 01030323030101210 1030
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 21)
(d0, d1, d2, d3)         = (26, 16, 26, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 1)
```

factorization_points()

Returns the eight factorization points of this configuration

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.factorization_points()
[0, 2, 3, 5, 6, 8, 9, 11]
```

hat()

Return the hat function returning the reversal of a word path.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.w(0)
Path: 32303010
sage: D.hat(D.w(0))
Path: 23212101
```

height()

Returns the width of this polyomino, i.e. the difference between its uppermost and lowermost coordinates

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.height()
9
```

```
sage: D = DoubleSquare((34,21,34,21))
sage: D.height()
23
```

is_degenerate()

Return whether self is degenerate.

A double square is *degenerate* if one of the w_i is empty.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.is_degenerate()
False
```

```
sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: D = DoubleSquare(('','0','10','1'), rot180)
sage: D.is_degenerate()
True
```

is_flat()

Return whether self is flat.

A double square is *flat* if one of the $w_i w_{i+1}$ is empty.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.is_flat()
False
```

```
sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: D = DoubleSquare(('','','0101','01'), rot180)
sage: D.is_flat()
True
```

is_morphic_pentamino()

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.is_morphic_pentamino()
True
```

is_singular()

Return whether self is singular.

A double square is *singular* if there exists i such that w_{i-1} and w_{i+1} are empty, equivalently if $d_i = 0$.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.is_singular()
False
```

```
sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: D = DoubleSquare(('','03010','','1011'), rot180)
sage: D.is_singular()
True
```

latex_8_tuple()

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.latex_8_tuple()
('{\\bf 32303010}', '{\\bf 30323}', '{\\bf 21232303}', '{\\bf 23212}',
 '{\\bf 10121232}', '{\\bf 12101}', '{\\bf 03010121}', '{\\bf 01030}')
```

latex_array()

Return a LaTeX array of self.

This code was used to create Table 1 in *[BGL2012]*.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: print(D.latex_array())
\begin{array}{lllllll}
i & w_i & u_i & v_i & |w_i| & d_i & n_i
\\
\hline
0 & {\bf 32} & {\bf } & {\bf 32} & 2 & 2 & 1\\
1 & {\bf 3} & {\bf 3} & {\bf 032} & 1 & 4 & 0\\
2 & {\bf 03} & {\bf } & {\bf 03} & 2 & 2 & 1\\
3 & {\bf 0} & {\bf 0} & {\bf 103} & 1 & 4 & 0\\
4 & {\bf 10} & {\bf } & {\bf 10} & 2 & 2 & 1\\
5 & {\bf 1} & {\bf 1} & {\bf 210} & 1 & 4 & 0\\
6 & {\bf 21} & {\bf } & {\bf 21} & 2 & 2 & 1\\
7 & {\bf 2} & {\bf 2} & {\bf 321} & 1 & 4 & 0\\
\hline
\end{array}
```

latex_table()

Returns a Latex expression of a table containing the parameters of this double square.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.latex_table()
\begin{tabular}{|c|}
\hline
\\
\begin{tikzpicture}
[first/.style={circle,draw=black,fill=gray, inner sep=0pt, minimum size=3pt},
second/.style={rectangle,draw=black,fill=white, inner sep=0pt, minimum size=3pt}]
...
\end{tikzpicture} \\[1ex]
\hline\\
$(w_0,w_1,w_2,w_3) = (8,5,8,5)$ \\
$u_0 = 32303010$\quad $u_1 = 30323$\\$u_2 = 21232303$\quad $u_3 = 23212$\\
$v_0 = 30$\quad $v_1 = 21232303010$\\$v_2 = 23$\quad $v_3 = 10121232303$\\
$(n_0,n_1,n_2,n_3) = (0,0,0,0)$ \\
Turning number = -1\\
Self-avoiding = True\\
\hline
\end{tabular}
```

n(*i*)

Return the integer n_i.

The value of n_i is defined as the quotient of $|w_i|$ by d_i.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: [D.n(i) for i in range(8)]
[0, 0, 0, 0, 0, 0, 0, 0]
```

```
sage: A = D.extend(1).extend(1).extend(1).extend(1)
sage: [A.n(i) for i in range(8)]
[0, 4, 0, 0, 0, 4, 0, 0]
```

If $d_i = 0$ then n_i is not defined:

```
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: B = D.reduce_ntimes(2)
```

(continues on next page)

(continued from previous page)

```
sage: [B.n(i) for i in range(8)]
[0, NaN, 0, NaN, 0, NaN, 0, NaN]
```

plot(*pathoptions={'rgbcolor': 'black', 'thickness': 3}, fill=True, filloptions={'alpha': 0.2, 'rgbcolor': 'black'}, startpoint=True, startoptions={'pointsize': 100, 'rgbcolor': 'black'}, endarrow=True, arrowoptions={'arrowsize': 5, 'rgbcolor': 'black', 'width': 3}, gridlines=False, gridoptions={}, axes=False*)

Returns a 2d Graphics illustrating the double square tile associated to this configuration including the factorizations points.

INPUT:

- `pathoptions` - (dict, default:dict(rgbcolor='red',thickness=3)), options for the path drawing
- `fill` - (boolean, default: True), if fill is True and if the path is closed, the inside is colored
- `filloptions` - (dict, default:dict(rgbcolor='red',alpha=0.2)), ptions for the inside filling
- `startpoint` - (boolean, default: True), draw the start point?
- `startoptions` - (dict, default:dict(rgbcolor='red',pointsize=100)) options for the start point drawing
- `endarrow` - (boolean, default: True), draw an arrow end at the end?
- `arrowoptions` - (dict, default:dict(rgbcolor='red',arrowsize=20, width=3)) options for the end point arrow
- `gridlines`- (boolean, default: False), show gridlines?
- `gridoptions` - (dict, default: {}), options for the gridlines
- `axes` - (boolean, default: False), options for the axes

EXAMPLES:

The cross of area 5 together with its double square factorization points:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: _ = D.plot()               # long time (1s)
```

plot_reduction(*ncols=3, options={}*)

Return a graphics array of the reduction.

INPUT:

- `ncols` - integer (default: 3), number of columns
- `options` - dict (default: {}), options given to the plot method of each double square

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: _ = D.plot_reduction()          # long time (1s)
```

Using the color options:

```
sage: p = dict(rgbcolor='red', thickness=1)
sage: q = dict(rgbcolor='blue', alpha=1)
sage: options = dict(endarrow=False,startpoint=False,pathoptions=p,filloptions=q)
sage: _ = D.plot_reduction(options=options)      # long time (1s)
```

reduce()

Reduces self by the application of TRIM or otherwise SWAP.

INPUT:

- `self` - non singular double square tile on the alphabet $0, 1, 2, 3$ such that its turning number is +1 or -1.

OUTPUT:

- DoubleSquare - the reduced double square
- string - the operation which was performed

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare((34,21,34,21))
sage: E,op = D.reduce()
sage: E
Double Square Tile
  w0 = 32303010                 w4 = 10121232
  w1 = 303232123230301030323    w5 = 121010301012123212101
  w2 = 21232303                 w6 = 03010121
  w3 = 232121012123230323212    w7 = 010303230301012101030
(|w0|, |w1|, |w2|, |w3|) = (8, 21, 8, 21)
(d0, d1, d2, d3)         = (42, 16, 42, 16)
(n0, n1, n2, n3)         = (0, 1, 0, 1)
sage: op
'SWAP_1'
```

```
sage: D = DoubleSquare((1,2,2,1))
sage: D
Double Square Tile
  w0 = 1    w4 = 1
  w1 = 23   w5 = 03
  w2 = 12   w6 = 10
  w3 = 3    w7 = 3
(|w0|, |w1|, |w2|, |w3|) = (1, 2, 2, 1)
(d0, d1, d2, d3)         = (3, 3, 3, 3)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.reduce()
Traceback (most recent call last):
...
ValueError: not reducible, because self is nondegenerate and
d_0 == d_1 == 3. Also, the turning number (=0) must be +1 or -1
for the reduction to apply.
```

TESTS:

```
sage: D = DoubleSquare((5,4,3,4))
sage: D
Double Square Tile
  w0 = 90128   w4 = 40123
  w1 = 7659    w5 = 7654
  w2 = 012     w6 = 012
  w3 = 3765    w7 = 8765
(|w0|, |w1|, |w2|, |w3|) = (5, 4, 3, 4)
(d0, d1, d2, d3)         = (8, 8, 8, 8)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.reduce()
Traceback (most recent call last):
...
ValueError: not reducible, because self is nondegenerate and
d_0 == d_1 == 8. Also, the turning number (=-1) must be +1 or
-1 for the reduction to apply.
```

reduce_ntimes(*iteration=1*)

Reduces the double square self until it is singular.

INPUT:

- `iteration` - integer (default: 1), number of iterations to perform

OUTPUT:

DoubleSquare

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare((34,21,34,21))
sage: D.reduce_ntimes(10)
Double Square Tile
  w0 =      w4 =
  w1 = 3    w5 = 1
  w2 =      w6 =
  w3 = 2    w7 = 0
(|w0|, |w1|, |w2|, |w3|) = (0, 1, 0, 1)
(d0, d1, d2, d3)         = (2, 0, 2, 0)
(n0, n1, n2, n3)         = (0, NaN, 0, NaN)
```

reduction()

Return the list of operations to reduce self to a singular double square.

OUTPUT:

- list of strings

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare((34,21,34,21))
sage: D.reduction()
['SWAP_1', 'TRIM_1', 'TRIM_3', 'SWAP_1', 'TRIM_1', 'TRIM_3', 'TRIM_0', 'TRIM_2']
```

```
sage: from slabbe import christoffel_tile
sage: D = DoubleSquare(christoffel_tile(9,7))
sage: D.reduction()
['TRIM_2', 'TRIM_1', 'TRIM_1', 'TRIM_1', 'TRIM_0', 'TRIM_2', 'TRIM_2']
```

reverse()

Apply $REVERSE$ on self.

This reverses the words w_i.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D
Double Square Tile
  w0 = 32303010   w4 = 10121232
  w1 = 30323      w5 = 12101
  w2 = 21232303   w6 = 03010121
  w3 = 23212      w7 = 01030
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
(d0, d1, d2, d3)         = (10, 16, 10, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.reverse()
Double Square Tile
  w0 = 21232      w4 = 03010
```

(continues on next page)

(continued from previous page)

```
  w1 = 30323212   w5 = 12101030
  w2 = 32303      w6 = 10121
  w3 = 01030323   w7 = 23212101
(|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
(d0, d1, d2, d3)         = (16, 10, 16, 10)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.reverse().reverse() == D
True
```

shift()

Apply $SHIFT$ on self.

This replaces w_i by w_{i+1}.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D
Double Square Tile
  w0 = 32303010   w4 = 10121232
  w1 = 30323      w5 = 12101
  w2 = 21232303   w6 = 03010121
  w3 = 23212      w7 = 01030
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
(d0, d1, d2, d3)         = (10, 16, 10, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.shift()
Double Square Tile
  w0 = 30323      w4 = 12101
  w1 = 21232303   w5 = 03010121
  w2 = 23212      w6 = 01030
  w3 = 10121232   w7 = 32303010
(|w0|, |w1|, |w2|, |w3|) = (5, 8, 5, 8)
(d0, d1, d2, d3)         = (16, 10, 16, 10)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
sage: D.shift().shift().shift().shift().shift().shift().shift().shift() == D
True
sage: D.shift().shift().shift().shift() == D
False
```

swap(i)

Apply $SWAP_i$ on self.

This replaces w_j by $\hat{w_{j+4}}$ for each $j = i, i+2, i+4, i+6$ and $w_j = (u_j * v_j)^{n_j} u_j$ by $(v_j * u_j)^{n_j} v_j$ for each $j = i+1, i+3, i+5, i+7$. This is an involution if the u_j are non empty.

INPUT:

- i - integer

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D
Double Square Tile
  w0 = 32303010   w4 = 10121232
  w1 = 30323      w5 = 12101
  w2 = 21232303   w6 = 03010121
  w3 = 23212      w7 = 01030
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
(d0, d1, d2, d3)         = (10, 16, 10, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
```

(continues on next page)

(continued from previous page)

```
sage: D.swap(1)
Double Square Tile
  w0 = 30      w4 = 12
  w1 = 32303   w5 = 10121
  w2 = 23      w6 = 01
  w3 = 21232   w7 = 03010
(|w0|, |w1|, |w2|, |w3|) = (2, 5, 2, 5)
(d0, d1, d2, d3)         = (10, 4, 10, 4)
(n0, n1, n2, n3)         = (0, 1, 0, 1)
```

tikz_boxed(*scale=1*, *boxsize=10*)

Return a tikzpicture of self included in a box.

INPUT:

- `scale` - number (default: 1), tikz scale
- `boxsize` - integer (default: `10`), size of the box. If the width and height of the double square is less than the boxsize, then unit step are of size 1 and the (w_i) 8-tuple is added below the figure. Otherwise, if the width or height is larger than the boxsize, then the unit step are made smaller to fit the box and the (w_i) 8-tuple is not shown.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.tikz_boxed()
\begin{tabular}{c}
\begin{tikzpicture}
[scale=1]
\filldraw[-to, very thick, draw=black, fill=black!20] (0.000,
0.000) -- (0.000, -1.00) -- (-1.00, -1.00) -- (-1.00, -2.00)
-- (0.000, -2.00) -- (0.000, -3.00) -- (1.00, -3.00) -- (1.00,
-2.00) -- (2.00, -2.00) -- (2.00, -1.00) -- (1.00, -1.00) --
(1.00, 0.000) -- (0.000, 0.000);
\node[first] at (0.0000, 0.0000) {};
\node[first] at (-1.000, -2.000) {};
\node[first] at (1.000, -3.000) {};
\node[first] at (2.000, -1.000) {};
\node[second] at (-1.000, -1.000) {};
\node[second] at (0.0000, -3.000) {};
\node[second] at (2.000, -2.000) {};
\node[second] at (1.000, 0.0000) {};
\end{tikzpicture}
\\
$({\bf 32},{\bf 3},{\bf 03},{\bf 0},$ \\
$\phantom{(}{\bf 10},{\bf 1},{\bf 21},{\bf 2})$ \\
\end{tabular}
```

Smaller boxsize:

```
sage: D.tikz_boxed(boxsize=1.5)
\begin{tikzpicture}
[scale=1]
\filldraw[-to, very thick, draw=black, fill=black!20] (0.000, 0.000) --
(0.000, -0.500) -- (-0.500, -0.500) -- (-0.500, -1.00) -- (0.000, -1.00) --
(0.000, -1.50) -- (0.500, -1.50) -- (0.500, -1.00) -- (1.00, -1.00) --
(1.00, -0.500) -- (0.500, -0.500) -- (0.500, 0.000) -- (0.000, 0.000);
\node[first] at (0.0000, 0.0000) {};
\node[first] at (-0.5000, -1.000) {};
\node[first] at (0.5000, -1.500) {};
\node[first] at (1.000, -0.5000) {};
\node[second] at (-0.5000, -0.5000) {};
\node[second] at (0.0000, -1.500) {};
```

(continues on next page)

(continued from previous page)

```
\node[second] at (1.000, -1.000) {};
\node[second] at (0.5000, 0.0000) {};
\end{tikzpicture}
```

tikz_commutative_diagram(*tile*, *N=1*, *scale=(1, 1)*, *labels=True*, *newcommand=True*)

Return a tikz commutative diagram for the composition.

INPUT:

- `tile` - WordMorphism, a square tile
- `N` - integer (default:1), length of the diagram
- `scale` - tuple of number (default:`(1,1)`), one for each line
- `labels` - arrow labels (default:`True`). It may take the following values:
  - `True` - prints TRIM, SWAP, etc.
  - `'T'` - prints T_i, etc.
  - `False` - print nothing
- `newcommand` - bool (default: `True`), whether newcommand which defines `\SWAP`, `\TRIM`, etc.

EXAMPLES:

The following command creates the tikz code for Figure 16 in *[BGL2012]*:

```
sage: from slabbe import DoubleSquare
sage: fibo2 = words.fibonacci_tile(2)
sage: S = WordMorphism({0:[0,0],1:[1,0,1],2:[2,2],3:[3,2,3]}, codomain=fibo2.parent())
sage: cfibo2 = DoubleSquare(fibo2)
sage: options = dict(tile=S,N=3,scale=(0.25,0.15),labels=True,newcommand=True)
sage: s = cfibo2.tikz_commutative_diagram(**options)     # long time (2s)
sage: s                                                   # long time
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\begin{document}
\newcommand{\TRIM}{\textsc{trim}}
\newcommand{\EXTEND}{\textsc{extend}}
\newcommand{\SWAP}{\textsc{swap}}
\newcommand{\SHIFT}{\textsc{shift}}
\newcommand{\REVERSE}{\textsc{reverse}}
...
... 105 lines not printed (12001 characters in total) ...
...
\path[thick, ->] (q0) edge node[midway, left] {$\varphi$} (r0);
\path[thick, ->] (q1) edge node[midway, left] {$\varphi$} (r1);
\path[thick, ->] (q2) edge node[midway, left] {$\varphi$} (r2);
\path[thick, ->] (q3) edge node[midway, left] {$\varphi$} (r3);
\end{tikzpicture}
\end{document}
```

tikz_reduction(*scale=1*, *ncols=3*, *gridstep=5*, *labels=True*, *newcommand=True*)

INPUT:

- `scale` - number
- `ncols` - integer, number of columns displaying the reduction
- `gridstep` - number (default: `5`), the gridstep for the snake node positions
- `labels` - arrow labels (default:`True`). It may take the following values:
  - `True` - prints TRIM, SWAP, etc.

  - `'T'` - prints T_i, etc.
  - `False` - print nothing
- `newcommand` - bool (default: `True`), whether newcommand which defines `\SWAP`, `\TRIM`, etc.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: fibo2 = words.fibonacci_tile(2)
sage: cfibo2 = DoubleSquare(fibo2)
sage: s = cfibo2.tikz_reduction(scale=0.5,ncols=4,labels=True)
sage: s
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\usetikzlibrary{pgfplots.groupplots}
\begin{document}
\newcommand{\TRIM}{\textsc{trim}}
\newcommand{\EXTEND}{\textsc{extend}}
\newcommand{\SWAP}{\textsc{swap}}
\newcommand{\SHIFT}{\textsc{shift}}
\newcommand{\REVERSE}{\textsc{reverse}}
...
... 103 lines not printed (6363 characters in total) ...
...
\path[->] (q1) edge node[midway, rectangle, fill=white, rotate=90] {$\TRIM_1$} (q2);
\path[->] (q2) edge node[midway, rectangle, fill=white, rotate=90] {$\TRIM_3$} (q3);
\path[->] (q3) edge node[midway, rectangle, fill=white] {$\TRIM_0$} (q4);
\path[->] (q4) edge node[midway, rectangle, fill=white, rotate=90] {$\TRIM_2$} (q5);
\end{tikzpicture}
\end{document}
```

```
sage: S = WordMorphism({0:[0,0],1:[1,0,1],2:[2,2],3:[3,2,3]}, codomain=fibo2.parent())
sage: cSfibo2 = cfibo2.apply_morphism(S)
sage: s = cSfibo2.tikz_reduction(scale=0.15,ncols=4,labels='T')
sage: s
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\usetikzlibrary{pgfplots.groupplots}
\begin{document}
\newcommand{\TRIM}{\textsc{trim}}
\newcommand{\EXTEND}{\textsc{extend}}
\newcommand{\SWAP}{\textsc{swap}}
\newcommand{\SHIFT}{\textsc{shift}}
\newcommand{\REVERSE}{\textsc{reverse}}
...
... 93 lines not printed (9437 characters in total) ...
...
\path[thick, ->] (q1) edge node[midway, above] {$T_2$} (q2);
\path[thick, ->] (q2) edge node[midway, above] {$T_3$} (q3);
\path[thick, ->] (q3) edge node[midway, above] {$T_4$} (q4);
\path[thick, ->] (q4) edge node[midway, above] {$T_5$} (q5);
\end{tikzpicture}
\end{document}
```

tikz_tiling(*nx=2*, *ny=2*, *kind=1*, *rectangle=None*, *clip=None*)

Return a tikz of the tiling.

INPUT:

- `nx` – integer
- `ny` – integer
- `kind` – integer, 1 or 2, first or second tiling
- `rectangle` – list of two points (to practice for the clip)

- `clip` – list of two points

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: d = DoubleSquare((5,8,5,8))
sage: t = d.tikz_tiling(nx=4, ny=4)
sage: t = d.tikz_tiling(nx=4, ny=4, kind=2)
```

tikz_trajectory(*step=1*, *arrow='->'*)

Returns a tikz string describing the double square induced by this configuration together with its factorization points

The factorization points respectively get the tikz attribute 'first' and 'second' so that when including it in a tikzpicture environment, it is possible to modify the way those points appear.

INPUT:

- `step` - integer (default: 1)
- `arrow` - string (default: ->), tikz arrow shape

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.tikz_trajectory()
\filldraw[->, very thick, draw=black, fill=black!20] (0.000, 0.000)
-- (0.000, -1.00) -- (-1.00, -1.00) -- (-1.00, -2.00) -- (0.000, -2.00) --
(0.000, -3.00) -- (1.00, -3.00) -- (1.00, -2.00) -- (2.00, -2.00) -- (2.00,
-1.00) -- (1.00, -1.00) -- (1.00, 0.000) -- (0.000, 0.000); \node[first] at
(0.0000, 0.0000) {};
\node[first] at (-1.000, -2.000) {};
\node[first] at (1.000, -3.000) {};
\node[first] at (2.000, -1.000) {};
\node[second] at (-1.000, -1.000) {};
\node[second] at (0.0000, -3.000) {};
\node[second] at (2.000, -2.000) {};
\node[second] at (1.000, 0.0000) {};
```

translation_vectors()

Returns two pairs of translation vectors of the two associated tiling.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.translation_vectors()
(((-1, -2), (2, -1)), ((1, -2), (2, 1)))
```

trim(*i*)

Apply $TRIM_i$ on self.

This removes a period of length d_i to w_i and w_{i+4}.

INPUT:

- `i` - integer

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare((3,6,3,2))
sage: D.trim(1)
Double Square Tile
```

(continues on next page)

(continued from previous page)

```
  w0 = 212   w4 = 030
  w1 =       w5 =
  w2 = 303   w6 = 121
  w3 = 03    w7 = 21
(|w0|, |w1|, |w2|, |w3|) = (3, 0, 3, 2)
(d0, d1, d2, d3)         = (2, 6, 2, 6)
(n0, n1, n2, n3)         = (1, 0, 1, 0)
```

```
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.extend(3).trim(3)
Double Square Tile
  w0 = 32303010   w4 = 10121232
  w1 = 30323      w5 = 12101
  w2 = 21232303   w6 = 03010121
  w3 = 23212      w7 = 01030
(|w0|, |w1|, |w2|, |w3|) = (8, 5, 8, 5)
(d0, d1, d2, d3)         = (10, 16, 10, 16)
(n0, n1, n2, n3)         = (0, 0, 0, 0)
```

turning_number()

Return the turning number of self.

INPUT:

- `self` - double square defined on the alphabet of integers $\{0, 1, 2, 3\}$

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: D.turning_number()
1
sage: D.reverse().turning_number()
-1
```

Turning number of a degenerate double square:

```
sage: D = DoubleSquare(([],[0],[1,0],[1]))
sage: D.turning_number()
1
```

Turning number of a singular double square:

```
sage: D = DoubleSquare(([],[0,3,0,1,0],[],[1,0,1,1]))
sage: D.turning_number()
1
```

Turning number of a flat double square:

```
sage: D = DoubleSquare(([],[],[0,1,0,1],[0,1]))
sage: D.turning_number()
0
```

u(*i*)

Return the word u_i.

The word u_i is the unique word such that $w_i = (u_i * v_i)^{n_i} u_i$ where $0 \leq |u_i| < d_i$.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
```

(continues on next page)

(continued from previous page)

```
sage: D.u(1)
Path: 30323
```

```
sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: steps = {'0':(1,0), '1':(0,1), '2':(-1,0), '3': (0,-1)}
sage: D = DoubleSquare(('','03010','','1011'), rot180, steps)
sage: D.u(0)
word:
sage: D.u(1)
Traceback (most recent call last):
...
ValueError: u_1 is not defined when d_1 == 0
```

v(*i*)

Return the word v_i.

The word v_i is the unique word such that $w_i = (u_i * v_i)^{n_i} u_i$ where $0 \leq |u_i| < d_i$, $\hat{w_{i-3}} w_{i-1} = u_i v_i$ and $0 < |u_i| \leq d_i$.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.v(1)
Path: 21232303010
```

```
sage: rot180 = WordMorphism('0->2,2->0,3->1,1->3')
sage: steps = {'0':(1,0), '1':(0,1), '2':(-1,0), '3': (0,-1)}
sage: D = DoubleSquare(('','03010','','1011'), rot180, steps)
sage: D.v(0)
word: 030103323
sage: D.v(1)
Traceback (most recent call last):
...
ValueError: v_1 is not defined when d_1 == 0
```

verify_definition()

Checks that the input verifies the definition.

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.verify_definition()
```

```
sage: DoubleSquare(([],[0],[0,1,0,1],[0,1]))
Traceback (most recent call last):
...
AssertionError: wiwi+1 = hat(wi+4,wi+5) is not verified for i=1
```

w(*i*)

Return the factor w_i

This corresponds to the new definition of configuration (solution).

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(1))
sage: [D.w(i) for i in range(8)]
[Path: 32, Path: 3, Path: 03, Path: 0, Path: 10, Path: 1, Path: 21, Path: 2]
```

```
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: [D.w(i) for i in range(8)]
[Path: 32303010, Path: 30323, Path: 21232303, Path: 23212, Path: 10121232, Path: 12101, Path:␣
↪03010121, Path: 01030]
```

width()

Returns the width of this polyomino, i.e. the difference between its rightmost and leftmost coordinates

EXAMPLES:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(words.fibonacci_tile(2))
sage: D.width()
9
```

```
sage: D = DoubleSquare((34,21,34,21))
sage: D.width()
23
```

slabbe.double_square_tile.**christoffel_tile**(*p*, *q*)

Returns the (p, q) Christoffel Tile *[BBGL2011]*.

EXAMPLES:

sage: from slabbe import christoffel_tile sage: christoffel_tile(7,9) Path: 0301030101030101030101030103010103010103… sage: christoffel_tile(9,7) Path: 0301010301010301010301010103010103010103… sage: christoffel_tile(2,3) Path: 0301030101030101212232123232123 sage: christoffel_tile(0,1) Path: 03012123 sage: print(christoffel_tile(4,5)) 030103010103010103010103010121232123232123232123232123

slabbe.double_square_tile.**double_hexagon_from_boundary_word**(*ds*)

Creates a double square object from the boundary word of a double square tile.

INPUT:

- ds - word, the boundary of a double square. The parent alphabet is assumed to be in the order : East, North, West, South.

OUTPUT:

- tuple - tuple of 8 words over the alphabet A
- WordMorphism, involution on the alphabet A and representing a rotation of 180 degrees.
- dict - mapping letters of A to steps in the plane.

EXAMPLES:

```
sage: from slabbe.double_square_tile import double_square_from_boundary_word
sage: fibo = words.fibonacci_tile
sage: W, rot180, steps = double_square_from_boundary_word(fibo(1))
sage: map(len, W)
[2, 1, 2, 1, 2, 1, 2, 1]
sage: W, rot180, steps = double_square_from_boundary_word(fibo(2))
sage: map(len, W)
[8, 5, 8, 5, 8, 5, 8, 5]
sage: W, rot180, steps = double_square_from_boundary_word(fibo(3))  # long time (6s)
sage: map(len, W)                                                   # long time
[34, 21, 34, 21, 34, 21, 34, 21]
sage: rot180                                                        # long time
WordMorphism: 0->2, 1->3, 2->0, 3->1
```

slabbe.double_square_tile.**double_hexagon_from_integers**(*l0*, *l1*, *l2*, *l3*, *l4*, *l5*)

Creates a double hexagon from the lengths of the w_i.

INPUT:

- l0 - integer, length of w_0
- l1 - integer, length of w_1
- l2 - integer, length of w_2
- l3 - integer, length of w_3
- l4 - integer, length of w_4
- l5 - integer, length of w_5

OUTPUT:

- tuple - tuple of 12 words over alphabet A
- WordMorphism, involution on the alphabet A and representing a rotation of 180 degrees.
- dict - mapping letters of A to steps in the plane.

EXAMPLES:

It seems difficult to find examples that do not overlap. Here are some on the square grid:

```
sage: from slabbe.double_square_tile import double_hexagon_from_integers
sage: w,rot180,steps = double_hexagon_from_integers(1,3,1,6,1,6)
sage: w
(Path: 2,
 Path: 222,
 Path: 2,
 Path: 323232,
 Path: 3,
 Path: 030303,
 Path: 0,
 Path: 000,
 Path: 0,
 Path: 101010,
 Path: 1,
 Path: 212121)
sage: w,rot180,steps = double_hexagon_from_integers(1,10,1,5,1,10)
```

On the hexagonal grid:

```
sage: w,rot180,steps = double_hexagon_from_integers(1,2,1,2,1,2)
sage: w,rot180,steps = double_hexagon_from_integers(2,5,2,5,2,5)
sage: w,rot180,steps = double_hexagon_from_integers(5,14,5,14,5,14) # une fleur!
sage: w,rot180,steps = double_hexagon_from_integers(5,22,5,22,5,22)
sage: w,rot180,steps = double_hexagon_from_integers(5,38,5,38,5,38)
```

To plot them:

```
sage: prod(w).plot()        # long time
Graphics object consisting of 4 graphics primitives
```

slabbe.double_square_tile.**double_square_from_boundary_word**(*ds*)

Creates a double square object from the boundary word of a double square tile.

INPUT:

- ds - word, the boundary of a double square. The parent alphabet is assumed to be in the order : East, North, West, South.

OUTPUT:

- tuple - tuple of 8 words over the alphabet A

- WordMorphism, involution on the alphabet A and representing a rotation of 180 degrees.
- dict - mapping letters of A to steps in the plane.

EXAMPLES:

```
sage: from slabbe.double_square_tile import double_square_from_boundary_word
sage: fibo = words.fibonacci_tile
sage: W, rot180, steps = double_square_from_boundary_word(fibo(1))
sage: map(len, W)
[2, 1, 2, 1, 2, 1, 2, 1]
sage: W, rot180, steps = double_square_from_boundary_word(fibo(2))
sage: map(len, W)
[8, 5, 8, 5, 8, 5, 8, 5]
sage: W, rot180, steps = double_square_from_boundary_word(fibo(3))  # long time (6s)
sage: map(len, W)                                                   # long time
[34, 21, 34, 21, 34, 21, 34, 21]
sage: rot180                                                        # long time
WordMorphism: 0->2, 1->3, 2->0, 3->1
```

slabbe.double_square_tile.**double_square_from_four_integers**(*l0*, *l1*, *l2*, *l3*)

Creates a double square from the lengths of the w_i.

INPUT:

- `l0` - integer, length of w_0
- `l1` - integer, length of w_1
- `l2` - integer, length of w_2
- `l3` - integer, length of w_3

OUTPUT:

- tuple - tuple of 8 words over alphabet A
- WordMorphism, involution on the alphabet A and representing a rotation of 180 degrees.
- dict - mapping letters of A to steps in the plane.

EXAMPLES:

```
sage: from slabbe.double_square_tile import double_square_from_four_integers
sage: w,rot180,steps = double_square_from_four_integers(2,1,2,1)
sage: w
(Path: 21, Path: 2, Path: 32, Path: 3, Path: 03, Path: 0, Path: 10, Path: 1)
sage: rot180
WordMorphism: 0->2, 1->3, 2->0, 3->1
sage: sorted(steps.items())
[(0, (1, 0)), (1, (0, 1)), (2, (-1, 0)), (3, (0, -1))]
```

If the input integers do not define a double square uniquely, the alphabet might be larger than 8:

```
sage: w,rot180,steps = double_square_from_four_integers(4,2,4,2)
sage: w
(Path: 7601,
 Path: 76,
 Path: 5476,
 Path: 54,
 Path: 2354,
 Path: 23,
 Path: 0123,
 Path: 01)
sage: rot180
WordMorphism: 0->4, 1->5, 2->6, 3->7, 4->0, 5->1, 6->2, 7->3
sage: sorted(steps.items())
```

(continues on next page)

(continued from previous page)

```
[(0, (1, 0)),
 (1, (1/2*sqrt(2), 1/2*sqrt(2))),
 (2, (0, 1)),
 (3, (-1/2*sqrt(2), 1/2*sqrt(2))),
 (4, (-1, 0)),
 (5, (-1/2*sqrt(2), -1/2*sqrt(2))),
 (6, (0, -1)),
 (7, (1/2*sqrt(2), -1/2*sqrt(2)))]
```

`slabbe.double_square_tile.`**`figure_11_BGL2012`**(*scale=0.5*, *boxsize=10*, *newcommand=True*)

Return the tikz code of the Figure 11 for the article *[BGL2012]*.

INPUT:

- `scale` - number (default: `0.5`), tikz scale
- `boxsize` - integer (default: `10`), size of box the double squares must fit in
- `newcommand` - bool (default: `True`), whether to include latex newcommand for TRIM, EXTEND and SWAP

EXAMPLES:

```
sage: from slabbe.double_square_tile import figure_11_BGL2012
sage: s = figure_11_BGL2012()
sage: s
\newcommand{\TRIM}{\textsc{trim}}
\newcommand{\EXTEND}{\textsc{extend}}
\newcommand{\SWAP}{\textsc{swap}}
\begin{tikzpicture}
[first/.style={circle,draw=black,fill=black, inner sep=0pt, minimum size=3pt},
second/.style={circle,draw=black,fill=white, inner sep=0pt, minimum size=3pt},
>=latex,
node distance=3cm]
\node (A) at (15,0)
{
\begin{tabular}{c}
\begin{tikzpicture}
[scale=0.5]
...
\end{tikzpicture}
};
\path[<-] (A) edge node[midway, rectangle, fill=white] {$\TRIM_2$} (B);
\path[<-] (B) edge node[midway, rectangle, fill=white] {$\TRIM_0$} (C);
\path[<-] (C) edge node[midway, rectangle, fill=white] {$\TRIM_1$} (D);
\path[<-] (D) edge node[midway, rectangle, fill=white] {$\TRIM_3$} (E);
\path[<-] (E) edge node[midway, rectangle, fill=white] {$\SWAP_1$} (F);
\path[<-] (F) edge node[midway, rectangle, fill=white] {$\TRIM_1$} (G);
\path[<-] (G) edge node[midway, rectangle, fill=white,rotate=90] {$\TRIM_3$} (H);
\path[<-] (H) edge node[midway, rectangle, fill=white,rotate=90] {$\SWAP_1$} (I);
\path[<-] (D) edge node[midway, rectangle, fill=white] {$\TRIM_2$} (E2);
\end{tikzpicture}
```

`slabbe.double_square_tile.`**`find_square_factorisation`**(*ds*, *factorisation=None*, *alternate=True*)

Return a square factorisation of the double square ds, distinct from the given factorisation.

INPUT:

- `ds` - word, the boundary word of a square tile
- `factorisation` - tuple (default: `None`), a known factorisation
- `alternate` - bool (default: `True`), if True the search for the second factorisation is restricted to those who alternates with the first factorisation

OUTPUT:

tuple of four positions of a square factorisation

EXAMPLES:

```
sage: from slabbe.double_square_tile import find_square_factorisation
sage: find_square_factorisation(words.fibonacci_tile(0))
(0, 1, 2, 3)
sage: find_square_factorisation(words.fibonacci_tile(1))
(0, 3, 6, 9)
sage: find_square_factorisation(words.fibonacci_tile(2))
(0, 13, 26, 39)
sage: find_square_factorisation(words.fibonacci_tile(3))
(0, 55, 110, 165)
```

```
sage: f = find_square_factorisation(words.fibonacci_tile(3))
sage: f
(0, 55, 110, 165)
sage: find_square_factorisation(words.fibonacci_tile(3),f)         # long time (6s)
(34, 89, 144, 199)
sage: find_square_factorisation(words.fibonacci_tile(3),f,False)   # long time (11s)
(34, 89, 144, 199)
```

```
sage: from slabbe import christoffel_tile
sage: find_square_factorisation(christoffel_tile(4,5))
(0, 7, 28, 35)
sage: find_square_factorisation(christoffel_tile(4,5),_)
(2, 27, 30, 55)
```

TESTS:

```
sage: find_square_factorisation(Words('abcd')('aaaaaa'))
Traceback (most recent call last):
...
ValueError: no square factorization found
sage: find_square_factorisation(Words('abcd')('aaaaaa'),(1,2,3,4))
Traceback (most recent call last):
...
ValueError: no second square factorization found
```

`slabbe.double_square_tile.`**`snake`**(*i*, *ncols=2*)

Return the coordinate of the ith node of a snake.

This is used for the tikz drawing of a double square reduction.

INPUT:

- `i` - integer, the ith node
- `ncols` - integer (default 2), number of columns

EXAMPLES:

```
sage: from slabbe.double_square_tile import snake
sage: for i in range(8): snake(i, 3)
(0, 0)
(1, 0)
(2, 0)
(2, -1)
(1, -1)
(0, -1)
(0, -2)
(1, -2)
```

slabbe.double_square_tile.**triple_square_example**(*i*)

Return a triple square factorisation example.

These words having three square factorisations were provided by Xavier Provençal.

INPUT:

- i - integer, accepted values are 1, 2 or 3.

EXAMPLES:

```
sage: from slabbe.double_square_tile import triple_square_example
sage: triple_square_example(1)
Path: abaBAbabaBAbabaBAbabABABabABABabABAB
sage: triple_square_example(2)
Path: abaBABaabaBABaabaBABaabABAAbabABAAbabABA...
sage: triple_square_example(3)
Path: aabAAbaabAAbaabAAbaaBAABaaBAABaaBAAB
```

Triple square tile do not exist. Hence the example provided by Xavier Provençal can not be the boundary word of a tile. One can see it by ploting it or by the fact that the turning number is zero:

```
sage: from slabbe import DoubleSquare
sage: D = DoubleSquare(triple_square_example(1))
sage: D
Double Square Tile
  w0 = a          w4 = a
  w1 = baBA       w5 = bABA
  w2 = babaB      w6 = BabAB
  w3 = AbabaBAb   w7 = ABabABAB
(|w0|, |w1|, |w2|, |w3|) = (1, 4, 5, 8)
(d0, d1, d2, d3)         = (12, 6, 12, 6)
(n0, n1, n2, n3)         = (0, 0, 0, 1)
sage: D.turning_number()
0
```

CHAPTER

TWO

COMBINATORICS ON WORDS

2.1 Kolakoski Word

Kolakoski Word

The classical Kolakoski sequence *[K65]* was first studied by Oldenburger *[O39]*, where it appears as the unique solution to the problem of a trajectory on the alphabet $\{1, 2\}$ which is identical to its exponent trajectory.

See http://en.wikipedia.org/wiki/Kolakoski_sequence

It uses cython implementation inspired from the 10 lines of C code written by Dominique Bernardi and shared during Sage Days 28 in Orsay, France, in January 2011. The alphabet must be `(1,2)`.

EXAMPLES:

```
sage: from slabbe import KolakoskiWord
sage: K = KolakoskiWord()
sage: K
word: 1221121221221121122121121221121121221221...
```

The cython implementation is much faster than the python one:

```
sage: K = KolakoskiWord()
sage: K[10^6]       # takes 0.02 seconds
2
sage: K = words.KolakoskiWord()
sage: K[10^6]       # not tested : takes too long
2
```

TESTS:

We make sure both implementation correspond for the prefix of length 100:

```
sage: Kb = KolakoskiWord()
sage: Kp = words.KolakoskiWord()
sage: Kp[:100] == Kb[:100]
True
```

REFERENCES:

AUTHORS:

- Sébastien Labbé (February 2011) - Added a Cython implementation which was easily translated from the 10 lines of C code written by Dominique Bernardi and shared during Sage Days 28 in Orsay. Needs review at #13346 since too long time...

`class slabbe.kolakoski_word.`**`KolakoskiWord`**`(`*parent=None*`)`

Bases: *`slabbe.kolakoski_word_pyx.WordDatatype_Kolakoski`*, `sage.combinat.words.infinite_word.InfiniteWord_class`

Constructor. See documentation of WordDatatype_Kolakoski for more details.

EXAMPLES:

```
sage: from slabbe import KolakoskiWord
sage: K = KolakoskiWord()
sage: K
word: 1221121221221121122121121221121121221221...
```

TESTS:

Pickle is supported:

```
sage: K = KolakoskiWord()
sage: loads(dumps(K))
word: 1221121221221121122121121221121121221221...
```

Kolakoski word (datatype)

class slabbe.kolakoski_word_pyx.**WordDatatype_Kolakoski**

Bases: `object`

Word datatype class for the Kolakoski word.

This is a cython implementation inspired from the 10 lines of C code written by Dominique Bernardi and shared during Sage Days 28 in Orsay, France, in January 2011.

INPUT:

- `parent` - the parent

EXAMPLES:

```
sage: from slabbe.kolakoski_word_pyx import WordDatatype_Kolakoski
sage: parent = Words([1,2])
sage: K = WordDatatype_Kolakoski(parent)
sage: K
<slabbe.kolakoski_word_pyx.WordDatatype_Kolakoski object at ...>
sage: K[0]
1
sage: K[1]
2
```

2.2 Factor complexity and Bispecial Extension Type

Factor complexity, Bispecial factors and Extension types

This module was developped for the article on the factor complexity of infinite sequences genereated by substitutions written with Valérie Berthé *[BL2014]*.

EXAMPLES:

The extension type of an ordinary bispecial factor:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E
  E(w)   1   2   3
   1             X
   2             X
   3     X   X   X
```

(continues on next page)

(continued from previous page)

```
 m(w)=0, ord.
sage: E.is_ordinaire()
True
```

Creation of a strong-weak pair of bispecial words from a neutral not ordinairy word:

```
sage: m = WordMorphism({1:[1,2,3],2:[2,3],3:[3]})
sage: E = ExtensionType1to1([(1,2),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: E
  E(w)  1   2   3
   1        X
   2            X
   3    X   X   X
 m(w)=0, neutral
sage: E1, E2 = E.apply(m)
sage: E1
  E(w)  1   2   3
   1
   2        X   X
   3    X   X   X
 m(w)=1, strong
sage: E2
  E(w)   1   2   3
    1        X
    2
    3            X
 m(w)=-1, weak
```

TODO:

- use __classcall_private__ stuff for ExtensionType ?
- fix bug of apply for ExtensionTypeLong when the word appears in the image of a letter (first initial fix: 18 May 2016, to be confirmed)
- use this to compute the factor complexity function
- When should two bispecial extension type be equal? In graphs, we sometimes prefer when they are all different…

class slabbe.bispecial_extension_type.**ExtensionType**

Bases: `object`

bispecial_factors_table_under_sadic(*substitutions*, *substitutions_dict*, *keep_empty=True*)

Return the summary table of bispecial factors obtain from this extension type after the application of substitutions.

INPUT:

- `substitutions` – the sequence of substitutions
- `substitutions_dict` - dict of substitutions
- `keep_empty` – (default: True) whether to keep images that are empty, thus it will include all bispecial factors of age <= k on the highest graded component.

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
```

(continues on next page)

(continued from previous page)

```
sage: E1.bispecial_factors_table_under_sadic([132]*2+[123]*6, S)
  |w|   w                     m(w)   d^-(w)   d2^-(w)   info
+-----+---------------------+------+--------+---------+---------+
  0                           0      3        5         ord.
  1     2                     0      3        4         neutral
  2     22                    0      2        2         ord.
  4     2322                  0      2        3         ord.
  5     22322                 0      2        2         ord.
  7     2322322               0      2        3         ord.
  8     22322322              0      2        2         ord.
  10    2322322322            0      2        3         ord.
  11    22322322322           0      2        2         ord.
  13    2322322322322         0      2        3         ord.
  14    22322322322322        0      2        2         ord.
  16    2322322322322322      0      2        3         ord.
  17    22322322322322322     0      2        2         ord.
  19    2322322322322322322   1      2        3         strong
  20    22322322322322322322  -1     2        2         weak
```

cardinality()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.cardinality()
5
```

distinct_bispecial_factors_under_sadic(*substitutions*, *substitutions_dict*, *keep_empty=True*)

Return the list of distinct bispecial factors obtain from this extension type after the application of substitutions.

This method essentially removes duplicates with distinct extension types but subset of others.

INPUT:

- `substitutions` – the sequence of substitutions
- `substitutions_dict` - dict of substitutions
- `keep_empty` – (default: True) whether to keep images that are empty, thus it will include all bispecial factors of age <= k on the highest graded component.

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
sage: L = E1.distinct_bispecial_factors_under_sadic([132]*2+[123]*6, S)
sage: [Z.factor() for Z in L]
[word: 2322322322322,
 word: ,
 word: 2,
 word: 22,
 word: 2322322322322322,
 word: 22322322322322322,
 word: 2322,
 word: 22322,
 word: 2322322322322322322,
 word: 22322322322322322322,
 word: 2322322,
```

(continues on next page)

(continued from previous page)

```
 word: 22322322,
 word: 2322322322,
 word: 22322322322,
 word: 22322322322322]
sage: [Z.multiplicity() for Z in L]
[0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 0, 0, 0, 0]
```

equivalence_class()

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: len(E.equivalence_class())
6
```

factor()

Return the bispecial factor.

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: p = WordMorphism({1:[1,2,3],2:[2,3],3:[3]})
sage: E = ExtensionType1to1([(1,2),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: A,B = E.apply(p)
sage: A.factor()
word: 3
sage: B.factor()
word: 23
```

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:      2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: b12 = WordMorphism({1:[1,2],2:[2],3:[3]})
sage: A,B = E.apply(b12)
sage: A.factor()
word:
sage: B.factor()
word: 2
```

```
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: data = [((3, 1), (2,)), ((1, 2), (3,)), ((3, 2), (3,)), ((2,
....:      3), (1,)), ((2, 3), (2,)), ((2, 3), (3,)), ((3, 3), (1,))]
sage: e = ExtensionTypeLong(data, [1,2,3])
sage: e = e.apply(S[132])[1]
sage: e.factor()
word: 2
sage: e = e.apply(S[321])[0]
sage: e.factor()
word: 21
sage: e = e.apply(S[312])[0]
sage: e.factor()
word: 212
```

static from_factor(*bispecial*, *word*, *nleft=1*, *nright=1*, *repr_options=None*)

INPUT:

- `bispecial` – the bispecial factor
- `word` – the word describing the language

- `nleft` – length of left extensions (default: 1)
- `nright` – length of right extensions (default: 1)
- `repr_options` - dict (default:`None`) representation options whether to include the factor and or the valence in the string or latex representation. The value `None` is replaced by `dict(factor=False, multiplicity=True,valence=False))`.

EXAMPLES:

```
sage: from slabbe import ExtensionType
sage: W = Words([0,1,2])
sage: ExtensionType.from_factor(W(), W([0,1,1,2,0]))
  E(w)   0   1   2
   0         X
   1         X   X
   2     X
 m(w)=-1, weak
```

```
sage: ExtensionType.from_factor(W(), W([0,1,1,2,0]), nleft=2)
  E(w)   0   1   2
   01        X
   11            X
   12    X
 m(w)=-1, weak
```

```
sage: ExtensionType.from_factor(W(), W([0,1,1,2,0]), nright=2)
  E(w)   11   12   20
   0     X
   1          X    X
 m(w)=0, ord.
```

```
sage: prefix = words.FibonacciWord()[:1000]
sage: ExtensionType.from_factor(W(), prefix, nright=2, nleft=2)
  E(w)   00   01   10
   00              X
   10         X    X
   01    X    X
 m(w)=0, ord.
```

static from_morphism(*m*, *repr_options=None*)

Return the extension type of the empty word in the language defined by the image of the free monoid under the morphism m.

INPUT:

- `m` - endomorphim
- `repr_options` - dict (default:`None`) representation options whether to include the factor and or the valence in the string or latex representation. The value `None` is replaced by `dict(factor=False, multiplicity=True,valence=False))`.

EXAMPLES:

```
sage: from slabbe import ExtensionType
sage: ar = WordMorphism({1:[1,3],2:[2,3],3:[3]})
sage: ExtensionType.from_morphism(ar)
  E(w)   1   2   3
   1             X
   2             X
   3     X   X   X
 m(w)=0, ord.
```

```
sage: p = WordMorphism({1:[1,2,3],2:[2,3],3:[3]})
sage: ExtensionType.from_morphism(p)
  E(w)   1   2   3
   1         X
   2             X
   3     X   X   X
 m(w)=0, neutral
```

```
sage: b12 = WordMorphism({1:[1,2],2:[2],3:[3]})
sage: ExtensionType.from_morphism(b12)
  E(w)   1   2   3
   1         X
   2     X   X   X
   3     X   X   X
 m(w)=2, strong
```

graph_under_language(*language*, *initial*, *substitutions_dict*, *keep_empty=False*, *max_depth=inf*, *growth_limit=inf*)

Return the recursively enumerated set of extension type generated by a language of substitutions.

INPUT:

- `language` – the language of substitutions
- `initial` – initial substitution
- `substitutions_dict` - dict of substitutions
- `keep_empty` – bool (default: False) whether to keep images that are empty
- `max_depth` – integer (default: `float('inf')`), max depth
- `growth_limit` – integer (default: `float('inf')`), the maximal growth in length of the bispecial extended images

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
sage: from slabbe.language import languages
sage: L = languages.Brun()
sage: E = [E for E in E1.apply(S[123]) if E.factor().length() == 1][0]
sage: E.graph_under_language(L, 123, S, max_depth=2)  # long time (3s)
Looped multi-digraph on 41 vertices
```

graph_under_language_joined(*language*, *initial*, *substitutions_dict*, *keep_empty=False*, *max_depth=inf*, *growth_limit=inf*)

Return the recursively enumerated set of extension type generated by a language of substitutions where the extension type of the same age and joined.

INPUT:

- `language` – the language of substitutions
- `initial` – initial substitution
- `substitutions_dict` - dict of substitutions
- `keep_empty` – bool (default: False) whether to keep images that are empty
- `max_depth` – integer (default: `float('inf')`), max depth

- `growth_limit` – integer (default: `float('inf')`), the maximal growth in length of the bispecial extended images

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
sage: from slabbe.language import languages
sage: L = languages.Brun()
sage: E = [E for E in E1.apply(S[123]) if E.factor().length() == 1][0]
sage: E.graph_under_language_joined(L, 123, S, max_depth=2)
Looped multi-digraph on 26 vertices
sage: E.graph_under_language_joined(L, 123, S, max_depth=2, growth_limit=1)
Looped multi-digraph on 21 vertices
sage: E.graph_under_language_joined(L, 123, S)   # not tested long time
Looped multi-digraph on 715 vertices
```

`graph_under_sadic`(*substitutions*, *substitutions_dict*, *keep_equal_length=False*, *raw=False*, *growth_limit=inf*)

Return the graph of extension types under the application of an s-adic word.

INPUT:

- `substitutions` – the sequence of substitutions
- `substitutions_dict` - dict of substitutions
- `keep_equal_length` – (default: False) whether to keep images that have equal length, thus it will include all bispecial factors of age <= k on the highest graded component.
- `raw` – bool (default: False), whether to keep the vertices raw, i.e. including history and factors information
- `growth_limit` – integer (default: `float('inf')`), the maximal growth in length of the bispecial extended images

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
sage: E1.graph_under_sadic([132]*2+[123]*6, S)
Looped multi-digraph on 9 vertices
sage: E1.graph_under_sadic([132]*2+[123]*6, S, keep_equal_length=True)
Looped multi-digraph on 19 vertices
sage: E1.graph_under_sadic([132]*2+[123]*6, S, raw=True)
Looped multi-digraph on 18 vertices
sage: E1.graph_under_sadic([132]*2+[123]*6, S, raw=True, keep_equal_length=True)
Looped multi-digraph on 59 vertices
```

```
sage: G = E1.graph_under_sadic([132]*2+[123]*6, S)
sage: from slabbe.tikz_picture import TikzPicture
sage: _ = TikzPicture.from_graph(G).pdf(view=False) # long time (9s)
```

`graph_under_sadic_joined`(*substitutions*, *substitutions_dict*, *keep_equal_length=False*, *keep_unique=False*, *growth_limit=inf*, *filter_fn=None*, *raw=False*)

Return the graph of extension types under the application of an s-adic word where the extension type of the same age are joined.

INPUT:

- `substitutions` – the sequence of substitutions
- `substitutions_dict` - dict of substitutions
- `keep_equal_length` – (default: False) whether to keep images that have equal length
- `keep_unique` – (default: False) whether to keep a unique copy of equal extension types
- `growth_limit` – integer (default: `float('inf')`), the maximal growth in length of the bispecial extended images
- `filter_fn` – function (default: `None`)
- `raw` – bool (default: False), whether to keep the vertices raw, i.e. including history and factors information

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
sage: E1._repr_options=dict(factor=False)
sage: seq = [231,213,213,213,321]+[213,231,231,231,123]+[132,123]
sage: E1.graph_under_sadic_joined(seq, S, growth_limit=1)
Looped multi-digraph on 10 vertices
```

image(*m*)

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: b23 = WordMorphism({1:[1],2:[2,3],3:[3]})
sage: E.image(b23)
  E(w)   1   2   3
   31        X
   12            X
   32            X
   23    X   X   X
   33    X
 m(w)=0, neutral
```

information()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.information()
'ord.'
```

is_bispecial()

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
```

(continues on next page)

(continued from previous page)

```
sage: E.is_bispecial()
True
```

is_empty()

Return whether the word associated to this extension type is empty.

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.is_empty()
False
```

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.is_empty()
True
```

```
sage: E = ExtensionTypeLong(L, (1,2,3), empty=False)
sage: E.is_empty()
False
```

is_equivalent(*other*)

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.is_equivalent(E)
True
```

is_neutral()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.is_neutral()
True
```

is_ordinaire()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.is_ordinaire()
True
```

is_subset(*other*)

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
```

(continues on next page)

(continued from previous page)

```
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: F = ExtensionTypeLong(L, (1,2,3))
sage: E.is_subset(F)
True
sage: F.is_subset(E)
False
```

left_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.left_extensions()
{1, 2, 3}
```

left_right_projection()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,2), (2,2), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.left_right_projection()
(Counter({3: 3, 1: 1, 2: 1}), Counter({2: 3, 1: 1, 3: 1}))
```

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: left, right = E.left_right_projection()
sage: sorted(left.iteritems())
[(word: 12, 3), (word: 21, 1), (word: 22, 1), (word: 23, 1), (word: 31, 1)]
sage: sorted(right.iteritems())
[(word: 1, 3), (word: 2, 3), (word: 3, 1)]
```

multiplicity()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.multiplicity()
0
```

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.multiplicity()
0
```

palindromic_valence()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.palindromic_valence()
1
```

`rec_enum_set_under_language`(*language*, *initial*, *substitutions_dict*, *keep_empty=False*, *label='history'*, *growth_limit=inf*)

Return the recursively enumerated set of extension type generated by a language of substitutions.

INPUT:

- `language` – the language of substitutions
- `initial` – initial substitution
- `substitutions_dict` - dict of substitutions
- `keep_empty` – (default: False) whether to keep images that are empty
- `label` – 'history' or 'previous' (default: `'history'`), whether the vertices contain the whole history of the bispecial word or only the previous applied substitution
- `growth_limit` – integer (default: `float('inf')`), the maximal growth in length of the bispecial extended images

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: from slabbe.bispecial_extension_type import ExtensionType
sage: from slabbe.language import languages
sage: algo = Brun()
sage: S = algo.substitutions()
sage: L = languages.Brun()
sage: v = algo.image((1,e,pi), 5)
sage: prefix = algo.s_adic_word(v)[:100000]
sage: E = ExtensionType.from_factor(prefix.parent()(), prefix, nleft=2)
sage: E.rec_enum_set_under_language(L, 123, S)
An enumerated set with a forest structure
```

```
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
sage: from slabbe.language import languages
sage: L = languages.Brun()
sage: E = [E for E in E1.apply(S[123]) if E.factor().length() == 1][0]
sage: R = E.rec_enum_set_under_language(L, 123, S, label='previous')
sage: R
A recursively enumerated set (breadth first search)
```

`rec_enum_set_under_language_joined`(*language*, *initial*, *substitutions_dict*, *keep_equal_length=False*, *keep_unique=False*, *label='history'*, *growth_limit=inf*)

Return the recursively enumerated set of extension type generated by a language of substitutions where the extension type of the same age and joined.

INPUT:

- `language` – the language of substitutions
- `initial` – initial substitution
- `substitutions_dict` - dict of substitutions
- `keep_equal_length` – (default: False) whether to keep images that have equal length
- `keep_unique` – (default: False) whether to keep a unique copy of equal extension types
- `label` – 'history' or 'previous' (default: `'history'`), whether the vertices contain the whole history of the bispecial word or only the previous applied substitution

- `growth_limit` – integer (default: `float('inf')`), the maximal growth in length of the bispecial extended images

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import ExtensionType
sage: from slabbe.mult_cont_frac import Brun
sage: from slabbe.language import languages
sage: algo = Brun()
sage: S = algo.substitutions()
sage: L = languages.Brun()
sage: v = algo.image((1,e,pi), 5)
sage: prefix = algo.s_adic_word(v)[:100000]
sage: E = ExtensionType.from_factor(prefix.parent()(), prefix, nleft=2)
sage: E.rec_enum_set_under_language_joined(L, 123, S)
A recursively enumerated set (breadth first search)
sage: E.rec_enum_set_under_language_joined(L, 123, S, label='previous')
A recursively enumerated set (breadth first search)
```

```
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
sage: from slabbe.language import languages
sage: L = languages.Brun()
sage: E = [E for E in E1.apply(S[123]) if E.factor().length() == 1][0]
sage: R = E.rec_enum_set_under_language_joined(L, 123, S, label='previous')
sage: R
A recursively enumerated set (breadth first search)
```

rec_enum_set_under_sadic(*substitutions*, *substitutions_dict*, *keep_equal_length=False*, *growth_limit=inf*)

Return the graded recursively enumerated set of all the extension type leading to those of age k generated by a finite sequence of substitutions of length k.

INPUT:

- `substitutions` – the sequence of substitutions
- `substitutions_dict` - dict of substitutions
- `keep_equal_length` – (default: False) whether to keep images that have equal length, thus it will include all bispecial factors of age <= k on the highest graded component.
- `growth_limit` – integer (default: `float('inf')`), the maximal growth in length of the bispecial extended images

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
sage: R = E1.rec_enum_set_under_sadic([132]*2+[123]*6, S)
sage: R
A recursively enumerated set with a graded structure (breadth first search)
sage: R.graded_component(0)
{(  E(w)   1   2   3
     21        X
     31        X
     12    X   X   X
```

(continues on next page)

(continued from previous page)

```
     22             X
     23    X
  m(w)=0, neutral, word: , ())}
sage: [len(R.graded_component(i)) for i in range(9)]
[1, 2, 2, 2, 2, 2, 2, 2, 3]
```

This used to be a bug in sage, it is now fixed:

```
sage: R.graded_component(9)
set()
```

Including all younger bispecial factors:

```
sage: R = E1.rec_enum_set_under_sadic([132]*2+[123]*6, S, keep_equal_length=True)
sage: [len(R.graded_component(i)) for i in range(9)]
[1, 3, 4, 5, 6, 7, 8, 9, 16]
sage: B = R.graded_component(8)
sage: [(Z.factor(),Z.multiplicity()) for Z,_,_ in B]
[(word: 2322322322322322322, 1),
 (word: 22, 0),
 (word: 22322, 0),
 (word: 2322, 0),
 (word: 2, 0),
 (word: 22322322322322322322, -1),
 (word: 2322322322322, 0),
 (word: 2322322, 0),
 (word: , 0),
 (word: 22322322322, 0),
 (word: 22322322322322322, 0),
 (word: 22322322, 0),
 (word: 2322322322322322322, 0),
 (word: 22322322322322, 0),
 (word: 2322322322322322, 0),
 (word: 2322322322, 0)]
```

rec_enum_set_under_sadic_joined(*substitutions*, *substitutions_dict*, *keep_equal_length=False*, *keep_unique=False*, *growth_limit=inf*, *filter_fn=None*)

Return the recursively enumerated set of extension type generated by a language of substitutions where the extension type of the same age are joined.

INPUT:

- `substitutions` – the sequence of substitutions
- `substitutions_dict` - dict of substitutions
- `keep_equal_length` – (default: False) whether to keep images that have equal length
- `keep_unique` – (default: False) whether to keep a unique copy of equal extension types
- `growth_limit` – integer (default: `float('inf')`), the maximal growth in length of the bispecial extended images
- `filter_fn` – function (default: `None`)

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
```

```
sage: seq = [231,213,213,321]+[213,231,231,123]+[132,123]
sage: R = E1.rec_enum_set_under_sadic_joined(seq, S, growth_limit=1)
sage: R
A recursively enumerated set with a graded structure (breadth first search)
```

```
sage: from slabbe.bispecial_extension_type import recursively_enumerated_set_to_digraph
sage: G = recursively_enumerated_set_to_digraph(R)
sage: G
Looped multi-digraph on 11 vertices
```

right_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.right_extensions()
{1, 2, 3}
```

weakstrong_poset(*language*, *initial*, *substitutions_dict*, *depth*)

Return the Poset of words of the language ending with initial letter that gives weak or strong bispecial factors with the "is suffix" relation.

INPUT:

- `language` – the language of substitutions
- `initial` – initial substitution
- `substitutions_dict` - dict of substitutions
- `depth` – depth

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: from slabbe.bispecial_extension_type import ExtensionType
sage: from slabbe.language import languages
sage: algo = Brun()
sage: S = algo.substitutions()
sage: L = languages.Brun()
sage: v = algo.image((1,e,pi), 5)
sage: prefix = algo.s_adic_word(v)[:1000]
sage: E = ExtensionType.from_factor(prefix.parent()(), prefix, nleft=2)
sage: P = E.weakstrong_poset(L, 123, S, 4)
sage: P
Finite poset containing 2 elements
```

```
sage: from slabbe.tikz_picture import TikzPicture
sage: tikz = TikzPicture.from_poset(P)
sage: _ = tikz.pdf(view=False)
```

weakstrong_sublanguage(*language*, *initial*, *substitutions_dict*, *depth*, *keep_empty=False*)

Return the word of length depth+1 ending with initial letter of the language that gives weak or strong bispecial factors.

INPUT:

- `language` – the language of substitutions
- `initial` – initial substitution
- `substitutions_dict` - dict of substitutions
- `depth` – depth

- `keep_empty` – (default: False) whether to keep images that are empty

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: from slabbe.bispecial_extension_type import ExtensionType
sage: from slabbe.language import languages
sage: algo = Brun()
sage: S = algo.substitutions()
sage: L = languages.Brun()
sage: v = algo.image((1,e,pi), 5)
sage: prefix = algo.s_adic_word(v)[:100000]
sage: E = ExtensionType.from_factor(prefix.parent()(), prefix, nleft=2)
sage: E.weakstrong_sublanguage(L, 123, S, 2)
set()
sage: E.weakstrong_sublanguage(L, 123, S, 3)
{(213, 213, 231, 123), (231, 213, 231, 123)}
sage: E.weakstrong_sublanguage(L, 123, S, 4)   # long time (8s)
{(132, 213, 213, 231, 123),
 (213, 213, 213, 231, 123),
 (213, 213, 231, 231, 123),
 (213, 231, 213, 231, 123),
 (231, 213, 213, 231, 123),
 (231, 213, 231, 231, 123),
 (231, 231, 213, 231, 123),
 (312, 231, 213, 231, 123)}
```

class `slabbe.bispecial_extension_type.`**`ExtensionType1to1`**(*L*, *alphabet*, *chignons=('', '')*, *factor=word:*, *repr_options=None*)

Bases: *slabbe.bispecial_extension_type.ExtensionType*

INPUT:

- L - list of pairs of letters
- `alphabet` - the alphabet
- `chignons` - optional (default: None), pair of words added to the left and to the right of the image of the previous bispecial
- `factor` - optional (default: empty word), the factor
- `repr_options` - dict (default:`None`) representation options whether to include the factor and or the valence in the string or latex representation. The value `None` is replaced by `dict(factor=False, multiplicity=True,valence=False)`.

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E
  E(w)   1   2   3
   1             X
   2             X
   3     X   X   X
 m(w)=0, ord.
```

With chignons:

```
sage: E = ExtensionType1to1(L, [1,2,3], ('a','b'))
sage: E
  E(w)   1   2   3
   1             X
   2             X
```

(continues on next page)

(continued from previous page)

```
  3     X   X   X
m(w)=0, ord.
```

apply(*m*, *growth_limit=inf*)
EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E
  E(w)   1   2   3
   1             X
   2             X
   3     X   X   X
 m(w)=0, ord.
sage: ar = WordMorphism({1:[1,3],2:[2,3],3:[3]})
sage: E.apply(ar)
(  E(w)  1   2   3
    1            X
    2            X
    3    X   X   X
 m(w)=0, ord.,)
```

```
sage: ar = WordMorphism({1:[3,1],2:[3,2],3:[3]})
sage: E.apply(ar)
(  E(w)   1   2   3
    1             X
    2             X
    3     X   X   X
 m(w)=0, ord.,)
```

Creation of a pair of ordinaire bispecial words from an **ordinaire** word:

```
sage: e = ExtensionType1to1([(1,3),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: p0 = WordMorphism({1:[1,2,3],2:[2,3],3:[3]})
sage: e.apply(p0)
( E(w)   1   2   3
    1
    2            X
    3    X   X   X
 m(w)=0, ord.,)
sage: p3 = WordMorphism({1:[1,3,2],2:[2],3:[3,2]})
sage: e.apply(p3)
(  E(w)   1   2   3
    1
    2             X
    3     X   X   X
 m(w)=0, ord.,
   E(w)   1   2   3
    1             X
    2     X   X   X
    3
 m(w)=0, ord.)
```

Creation of a strong-weak pair of bispecial words from a neutral **not ordinaire** word:

```
sage: p0 = WordMorphism({1:[1,2,3],2:[2,3],3:[3]})
sage: e = ExtensionType1to1([(1,2),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: e.apply(p0)
(  E(w)   1   2   3
    1
    2         X   X
    3     X   X   X
```

(continues on next page)

(continued from previous page)

```
m(w)=1, strong,
  E(w)   1   2   3
   1         X
   2
   3             X
m(w)=-1, weak)
```

Creation of a pair of ordinaire bispecial words from an **not ordinaire** word:

```
sage: p1 = WordMorphism({1:[1,2],2:[2],3:[3,1,2]})
sage: e = ExtensionType1to1([(1,2),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: e.apply(p1)
(  E(w)   1   2   3
    1     X   X   X
    2             X
    3
 m(w)=0, ord.,
   E(w)   1   2   3
    1
    2         X
    3     X   X   X
 m(w)=0, ord.)
```

This result is now fixed:

```
sage: e = ExtensionType1to1([(1,2), (3,3)], [1,2,3])
sage: p3 = WordMorphism({1:[1,3,2],2:[2],3:[3,2]})
sage: e.apply(p3)
(  E(w)   1   2   3
    1         X
    2             X
    3
 m(w)=-1, weak,)
```

```
sage: e = ExtensionType1to1([(2,2),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: e.apply(p3)
(  E(w)   1   2   3
    1
    2         X   X
    3     X   X   X
 m(w)=1, strong,)
```

This result is now fixed:

```
sage: e = ExtensionType1to1([(2,2),(2,3),(3,1),(3,2),(3,3)], [1,2,3])
sage: p2 = WordMorphism({1:[1],2:[2,3,1],3:[3,1]})
sage: e.apply(p2)
(  E(w)   1   2   3
    1     X   X   X
    2         X   X
    3
 m(w)=1, strong,)
```

```
sage: e = ExtensionType1to1([(1,2),(3,3)], [1,2,3])
sage: e.apply(p2)
(  E(w)   1   2   3
    1         X
    2
    3             X
 m(w)=-1, weak,)
```

TESTS:

```
sage: L = [(1,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E
  E(w)   1   2   3
   1             X
   2
   3
 m(w)=0, neutral
sage: ar = WordMorphism({1:[1,3],2:[2,3],3:[3]})
sage: E.apply(ar)
()
```

POSSIBLE BUG:

```
sage: from slabbe import ExtensionType
sage: b23 = WordMorphism({1:[1],2:[2,3],3:[3]})
sage: b13 = WordMorphism({1:[1,3],2:[2],3:[3]})
sage: b31 = WordMorphism({1:[1],2:[2],3:[3,1]})
sage: e = ExtensionType.from_morphism(b23)
sage: r = e.apply(b23)[0]
sage: r.apply(b13)
()
sage: r.apply(b31)
()
```

cardinality()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.cardinality()
5
```

chignons_multiplicity_tuple()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3], ('a', 'b'))
sage: E.chignons_multiplicity_tuple()
('a', 'b', 0)
```

extension_digraph()

Return the extension directed graph made of edges

(-1,a) -> (+1,b)

for each pair (a,b) in the extension set.

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.extension_digraph()
Bipartite graph on 6 vertices
```

extension_graph(*loops=False*)

Return the extension graph made of edges (a,b) for each pair (a,b) in the extension set.

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.extension_graph()
Graph on 3 vertices
sage: E.extension_graph(loops=True)
Looped graph on 3 vertices
```

```
sage: L = [(1,1), (1,2), (2,1), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: G = E.extension_graph()
sage: G.is_connected()
False
```

is_ordinaire()

EXAMPLES:

ordinary:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E
  E(w)   1   2   3
   1             X
   2             X
   3     X   X   X
 m(w)=0, ord.
sage: E.is_ordinaire()
True
```

strong:

```
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3), (1,1)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.is_ordinaire()
False
```

neutral but not ordinary:

```
sage: L = [(1,1), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E
  E(w)   1   2   3
   1     X
   2             X
   3     X   X   X
 m(w)=0, neutral
sage: E.is_neutral()
True
sage: E.is_ordinaire()
False
```

not neutral, not ordinaire:

```
sage: L = [(1,1), (2,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E
  E(w)   1   2   3
   1     X
   2     X
   3         X   X
 m(w)=-1, weak
sage: E.is_neutral()
```

(continues on next page)

(continued from previous page)

```
False
sage: E.is_ordinaire()
False
```

left_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.left_extensions()
{1, 2, 3}
```

left_valence(*length=1*)

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.left_valence()
3
```

palindromic_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.palindromic_extensions()
{3}
```

right_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.right_extensions()
{1, 2, 3}
```

right_valence(*length=1*)

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3])
sage: E.right_valence()
3
```

table()

return a table representation of self.

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, alphabet=(1,2,3))
sage: E.table()
  E(w)   1   2   3
   1             X
   2             X
   3     X   X   X
```

```
sage: E = ExtensionType1to1(L, alphabet=(1,2,3), chignons=('a', 'b'))
sage: E.table()
  E(w)   1   2   3
    1            X
    2            X
    3    X   X   X
```

class slabbe.bispecial_extension_type.**ExtensionTypeLong**(*L*, *alphabet*, *chignons=('', '')*, *factor=word:*, *factors_length_k=None*, *empty=None*, *repr_options=None*)

Bases: *slabbe.bispecial_extension_type.ExtensionType*

Generalized to words.

INPUT:

- `L` - list of pairs of *words*
- `alphabet` - the alphabet
- `chignons` - optional (default: None), pair of words added to the left and to the right of the image of the previous bispecial
- `factor` - optional (default: empty word), the factor
- `factors_length_k` - list of factors of length 2. If None, they are computed from the provided extension assuming the bispecial factor is *empty*.
- `empty` - bool, (optional, default: None), if None, then it is computed from the chignons and takes value True iff the chignons are empty.
- `repr_options` - dict (default:`None`) representation options whether to include the factor and or the valence in the string or latex representation. The value `None` is replaced by `dict(factor=False, multiplicity=True,valence=False)`.

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E
  E(w)   1   2   3
   21        X
   31        X
   12    X   X   X
   22    X
   23    X
m(w)=0, neutral
```

apply(*m*, *l=2*, *r=1*, *growth_limit=inf*)

The code works for Brun here because we take length 2 on the left and length 1 on the right.

On utilise les facteurs de longueur 2 pour completer l'info qui peut manquer.

TODO: bien corriger les facteurs de longueurs 2 de l'image!!!

INPUT:

- `m` - substitution
- `l` - integer, length of left extension
- `r` - integer, length of right extension

- `growth_limit` – integer (default: `float('inf')`), the maximal growth in length of the bispecial extended images

OUTPUT:

list of Extension type of the bispecial images

POSSIBLE BUG:

```
sage: from slabbe import ExtensionType
sage: b23 = WordMorphism({1:[1],2:[2,3],3:[3]})
sage: b13 = WordMorphism({1:[1,3],2:[2],3:[3]})
sage: b31 = WordMorphism({1:[1],2:[2],3:[3,1]})
sage: e = ExtensionType.from_morphism(b23)
sage: r = e.apply(b23)[0]
sage: r.apply(b13)
()
sage: r.apply(b31)
()
```

Ce bug se corrige avec de plus grandes extensions a gauche et en considérant les facteurs de longueur 2:

```
sage: from slabbe import ExtensionTypeLong
sage: E = ExtensionTypeLong((([a],[b]) for a,b in e), (1,2,3))
sage: R = E.apply(b23, l=1)[0]
sage: R.apply(b13, l=1)
(  E(w)   1   2   3
    1             X
    2             X
    3     X   X   X
 m(w)=0, ord.,)
sage: R.apply(b31, l=1)
(  E(w)   1   2   3
    1     X   X   X
    2             X
    3     X
 m(w)=0, neutral,
   E(w)   1   2   3
    1     X   X   X
    3     X   X   X
 m(w)=2, strong)
```

EXAMPLES:

On imagine qu'on vient de faire b12:

```
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:       2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E
  E(w)   1   2   3
   21        X
   31        X
   12    X   X   X
   22    X
   23    X
 m(w)=0, neutral
sage: b12 = WordMorphism({1:[1,2],2:[2],3:[3]})
sage: E.apply(b12)
(  E(w)   1   2   3
    21        X
    31        X
    12        X
    22    X   X   X
    23    X
  m(w)=0, neutral,
```

(continues on next page)

(continued from previous page)

```
   E(w)   1   2   3
    21        X
    31        X
    12    X   X   X
    22    X
  m(w)=0, ord.)
```

```
sage: b21 = WordMorphism({1:[1],2:[2,1],3:[3]})
sage: E.apply(b21)
(  E(w)   1   2   3
    11        X
    21    X   X   X
    31        X
    12    X
    13    X
  m(w)=0, ord.,
   E(w)   1   2   3
    21        X
    12    X   X   X
    13        X
  m(w)=0, ord.)
sage: b23 = WordMorphism({1:[1],2:[2,3],3:[3]})
sage: E.apply(b23)
(  E(w)   1   2   3
    31    X   X   X
    23    X
  m(w)=0, ord.,
   E(w)   1   2   3
    31        X
    12            X
    32            X
    23    X   X   X
    33    X
  m(w)=0, neutral,
   E(w)   1   2   3
    12    X   X   X
    32    X
    23    X
  m(w)=0, ord.)
```

Not letter is missing:

```
sage: data = [[(1, 1, 1), (3,)], [(1, 1, 1), (1,)], [(1, 1, 1), (2,)], [(1, 2),
....:   (1,)], [(2, 1), (1,)], [(1, 3), (1,)], [(3, 1), (2,)]]
sage: E5 = ExtensionTypeLong(data, (1,2,3))
sage: b21 = WordMorphism({1:[1],2:[2,1],3:[3]})
sage: E51, = [E for E in E5.apply(b21) if E.factor().length()==1]
sage: b21 = WordMorphism({1:[1],2:[2,1],3:[3]})
sage: E51.apply(b21)
(  E(w)   1   2   3
    11    X   X   X
    21    X
    12    X
    13        X
 m(w)=0, neutral,
   E(w)   1   2   3
    11    X   X   X
    21    X
    12    X
 m(w)=0, ord.)
```

We check that 12 is a left extension of X because this can not be guessed only from the direct image of left extensions:

```
sage: b21 = WordMorphism({1:[1],2:[2,1],3:[3]})
sage: data = [((1, 1), (2,)), ((2, 1), (3,)), ((2, 1), (2,)),
....:         ((1, 2), (1,)), ((2, 1), (1,)), ((1, 3), (2,))]
sage: F = [(1,1,1), (1,2,1), (1,1,3), (3,1,2), (2,1,1), (1,1,2), (1,3,1)]
sage: F = map(Word, F)
sage: E4_1 = ExtensionTypeLong(data, (1,2,3), factor=Word([1]),
....:                          factors_length_k=F, empty=False)
sage: X,Y = E4_1.apply(b21)
sage: X
  E(w)   1   2   3
   11    X   X   X
   21    X
   12    X
   13        X
m(w)=0, neutral
```

cardinality()

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.cardinality()
5
```

chignons_multiplicity_tuple()

EXAMPLES:

```
sage: from slabbe import ExtensionType1to1
sage: L = [(1,3), (2,3), (3,1), (3,2), (3,3)]
sage: E = ExtensionType1to1(L, [1,2,3], ('a', 'b'))
sage: E.chignons_multiplicity_tuple()
('a', 'b', 0)
```

extension_type_1to1()

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.extension_type_1to1()
  E(w)   1   2   3
   1         X
   2     X   X   X
   3     X
 m(w)=0, neutral
```

factors_length_k(*k=None*)

Returns the set of factors of length k of the language.

This is computed from the extension type if it was not provided at the construction.

INPUT:

- k – integer or None, if None, return factors of length k already computed

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
```

(continues on next page)

(continued from previous page)

```
sage: sorted(E.factors_length_k(2))
[word: 12, word: 21, word: 22, word: 23, word: 31]
sage: sorted(E.factors_length_k())
[word: 12, word: 21, word: 22, word: 23, word: 31]
```

It stores the value and can not compute for other lengths:

```
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: sorted(E.factors_length_k(3))
[word: 121, word: 122, word: 123, word: 212, word: 221, word: 231, word: 312]
sage: sorted(E.factors_length_k())
[word: 121, word: 122, word: 123, word: 212, word: 221, word: 231, word: 312]
sage: sorted(E.factors_length_k(4))
Traceback (most recent call last):
...
NotImplementedError: can't compute factors of length k for this word
```

```
sage: L = [(1, 2), (3, 2), (1, 3), (3, 3), (3, 1), (2, 3), (1, 1)]
sage: E = ExtensionTypeLong((([a], [b]) for a,b in L), (1,2,3))
sage: E.factors_length_k(2)
{word: 11, word: 12, word: 13, word: 23, word: 31, word: 32, word: 33}
```

TESTS:

```
sage: L = [(1, 2), (3, 2), (1, 3), (3, 3), (3, 1), (2, 3), (1, 1)]
sage: E = ExtensionTypeLong((([a], [b]) for a,b in L), (1,2,3), factors_length_k=set())
sage: E.factors_length_k(2)
Traceback (most recent call last):
...
NotImplementedError: can't compute factors of length k for this word
```

is_chignons_empty()

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.is_chignons_empty()
True
```

is_ordinaire()

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.extension_type_1to1()
  E(w)   1   2   3
   1         X
   2     X   X   X
   3     X
 m(w)=0, neutral
sage: E.is_ordinaire()
False
```

is_subset(*other*)

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 3), (1,)), ((2, 1), (2,)), ((1,
```

(continues on next page)

(continued from previous page)

```
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: F = ExtensionTypeLong(L, (1,2,3))
sage: E.is_subset(F)
True
sage: F.is_subset(E)
False
```

With incomplete word extensions:

```
sage: L = [((3,), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: G = ExtensionTypeLong(L, (1,2,3))
sage: G
  E(w)   1   2   3
   21        X
   31        X
   12    X   X   X
   3     X
m(w)=0, neutral
sage: G.is_subset(E)
True
```

is_valid()

Return whether self is valid, i.e, each left and right extension is non empty.

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.is_valid()
True
```

left_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.left_extensions()
{1, 2, 3}
```

left_valence(*length=1*)

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.left_valence()
3
sage: E.left_valence(2)
5
```

left_word_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:         2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: sorted(E.left_word_extensions())
[word: 12, word: 21, word: 22, word: 23, word: 31]
```

letters_before_and_after(*factors*)

Returns a pair of dict giving the words letters that goes before the left extensions and after the right extensions.

INPUT:

- `factors` – factors of length k

OUTPUT:

pair of dict

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: F = sorted(E.factors_length_k(2))
sage: E.letters_before_and_after(F)
({word: 12: {word: 2, word: 3},
  word: 21: {word: 1, word: 2},
  word: 22: {word: 1, word: 2},
  word: 23: {word: 1, word: 2},
  word: 31: {word: 2}},
 {word: 1: {word: 2},
  word: 2: {word: 1, word: 2, word: 3},
  word: 3: {word: 1}})
```

```
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: F = sorted(E.factors_length_k(3))
sage: E.letters_before_and_after(F)
({word: 12: {word: 2, word: 3},
  word: 21: {word: 1, word: 2},
  word: 22: {word: 1},
  word: 23: {word: 1},
  word: 31: {word: 2}},
 {word: 1: {word: 21, word: 22, word: 23},
  word: 2: {word: 12, word: 21, word: 31},
  word: 3: {word: 12}})
```

```
sage: data = [[(1, 1, 1), (3,)], [(1, 1, 1), (1,)], [(1, 1, 1), (2,)], [(1, 2),
....:   (1,)], [(2, 1), (1,)], [(1, 3), (1,)], [(3, 1), (2,)]]
sage: E5 = ExtensionTypeLong(data, (1,2,3))
sage: b21 = WordMorphism({1:[1],2:[2,1],3:[3]})
sage: E51, = [E for E in E5.apply(b21) if E.factor().length()==1]
sage: F = sorted(E51.factors_length_k(3))
sage: F
[word: 111, word: 112, word: 113, word: 121, word: 131, word: 211, word: 312]
sage: E.letters_before_and_after(F)
({word: 11: {word: 1, word: 2},
  word: 12: {word: 1, word: 3},
  word: 13: {word: 1},
  word: 21: {word: 1}},
 {word: 1: {word: 11, word: 12, word: 13, word: 21, word: 31},
  word: 2: {word: 11},
  word: 3: {word: 12}})
```

palindromic_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.palindromic_extensions()
{2}
```

right_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.right_extensions()
{1, 2, 3}
```

right_valence(*length=1*)

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E.right_valence()
3
```

right_word_extensions()

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: sorted(E.right_word_extensions())
[word: 1, word: 2, word: 3]
```

table()

return a table representation of self.

EXAMPLES:

```
sage: from slabbe import ExtensionTypeLong
sage: L = [((2, 2), (1,)), ((2, 3), (1,)), ((2, 1), (2,)), ((1,
....:          2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((3, 1), (2,))]
sage: E = ExtensionTypeLong(L, (1,2,3))
sage: E
  E(w)   1   2   3
   21        X
   31        X
   12    X   X   X
   22    X
   23    X
 m(w)=0, neutral
```

slabbe.bispecial_extension_type.**longest_common_prefix**(*L*)

Return the longest common prefix of a list of words.

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import longest_common_prefix
sage: longest_common_prefix((Word('ab'), Word('abc'), Word('abd')))
word: ab
```

slabbe.bispecial_extension_type.**longest_common_suffix**(*L*)

Return the longest common suffix of a list of words.

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import longest_common_suffix
sage: longest_common_suffix((Word('abc'), Word('bc'), Word('xabc')))
word: bc
```

slabbe.bispecial_extension_type.**rec_enum_set_under_language_joined_from_pairs**(*pairs*, *language*, *substitutions_dict*, *keep_equal_length=False*, *keep_unique=False*, *label='history'*, *growth_limit=inf*, *filter_fn=None*)

Return the recursively enumerated set of extension type generated by a language of substitutions where the extension type of the same age and joined.

INPUT:

- `pairs` – list of pairs of (extension type, previous substitution key)
- `language` – the language of substitutions
- `substitutions_dict` - dict of substitutions
- `keep_equal_length` – (default: False) whether to keep images that have equal length
- `keep_unique` – (default: False) whether to keep a unique copy of equal extension types
- `label` – 'history' or 'previous' (default: `'history'`), whether the vertices contain the whole history of the bispecial word or only the previous applied substitution
- `growth_limit` – integer (default: `float('inf')`), the maximal growth in length of the bispecial extended images
- `filter_fn` – function (default: `None`)

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import rec_enum_set_under_language_joined_from_pairs
sage: from slabbe.bispecial_extension_type import ExtensionType
sage: from slabbe.mult_cont_frac import Brun
sage: from slabbe.language import languages
sage: algo = Brun()
sage: S = algo.substitutions()
sage: L = languages.Brun()
sage: v = algo.image((1,e,pi), 5)
sage: prefix = algo.s_adic_word(v)[:1000]
sage: E = ExtensionType.from_factor(prefix.parent()(), prefix, nleft=2)
sage: pairs = [(E,123)]
sage: rec_enum_set_under_language_joined_from_pairs(pairs, L, S)
```

(continues on next page)

(continued from previous page)

```
A recursively enumerated set (breadth first search)
sage: rec_enum_set_under_language_joined_from_pairs(pairs, L, S, label='previous')
A recursively enumerated set (breadth first search)
```

```
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
sage: from slabbe.language import languages
sage: L = languages.Brun()
sage: E = [E for E in E1.apply(S[123]) if E.factor().length() == 1][0]
sage: pairs = [(E,123)]
sage: rec_enum_set_under_language_joined_from_pairs(pairs, L, S, label='previous')
A recursively enumerated set (breadth first search)
```

```
sage: from slabbe.mult_cont_frac import Brun
sage: algo = Brun()
sage: S = algo.substitutions()
sage: from slabbe.language import languages
sage: LBrun = languages.Brun()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:    2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
sage: pairs = [(E1, 312)] #, (E2, 312), (E3, 312), (E4, 321), (E5, 321)]
sage: f = lambda S:any(len(ext.left_word_extensions())>2 for ext in S)
sage: from slabbe.bispecial_extension_type import rec_enum_set_under_language_joined_from_pairs
sage: R = rec_enum_set_under_language_joined_from_pairs(pairs,
....:    LBrun, S, keep_equal_length=False, label='previous', growth_limit=1, filter_fn=f)
sage: R
A recursively enumerated set (breadth first search)
sage: from slabbe.bispecial_extension_type import recursively_enumerated_set_to_digraph
sage: recursively_enumerated_set_to_digraph(R)    # long time (12s)
Looped multi-digraph on 127 vertices
```

Testing the keep_unique option (not a good example apparently?):

```
sage: from slabbe.mult_cont_frac import Brun
sage: algo = Brun()
sage: S = algo.substitutions()
sage: from slabbe.language import languages
sage: LBrun = languages.Brun()
sage: data = [((1,1),(2,)),((2,1),(3,)),((2,1),(2,)),
....:         ((1,2),(1,)),((2,1),(1,)),((1,3),(2,))]
sage: factors = map(Word,[(1,1,1),(1,2,1),(1,1,3),(3,1,2),(2,1,1),(1,1,2),(1,3,1)])
sage: E4_1 = ExtensionTypeLong(data, (1,2,3), factor=Word([1]), factors_length_k=factors)
sage: pairs = [(E4_1, 321)]
sage: f = lambda S:any(len(ext.left_word_extensions())>2 for ext in S)
sage: R = rec_enum_set_under_language_joined_from_pairs(pairs,
....:    LBrun, S, keep_equal_length=False,
....:    keep_unique=False, label='previous', growth_limit=1, filter_fn=f)
sage: R.to_digraph()                # long time (15s)
Looped multi-digraph on 129 vertices
```

```
sage: R = rec_enum_set_under_language_joined_from_pairs(pairs,
....:    LBrun, S, keep_equal_length=False,
....:    keep_unique=True, label='previous', growth_limit=1, filter_fn=f)
sage: R.to_digraph()                # long time (15s)
Looped multi-digraph on 129 vertices
```

`slabbe.bispecial_extension_type.`**`recursively_enumerated_set_to_digraph`**(*R*, *max_depth=inf*)

Return the graph of the recursively enumerated set.

TODO:

Move this to sage.

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import recursively_enumerated_set_to_digraph
sage: child = lambda i: [(i+3) % 10, (i+8)%10]
sage: R = RecursivelyEnumeratedSet([0], child)
sage: G = recursively_enumerated_set_to_digraph(R)
sage: G
Looped multi-digraph on 10 vertices
```

`slabbe.bispecial_extension_type.`**`remove_extension_types_subsets`**(*extensions*)

Remove the extension types that are subset of another one associated to the same factor.

INPUT:

- `extensions` – iterable for extension types

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import ExtensionTypeLong
sage: from slabbe.mult_cont_frac import Brun
sage: S = Brun().substitutions()
sage: data = [((2, 1), (2,)), ((3, 1), (2,)), ((2, 2), (3,)), ((1,
....:     2), (1,)), ((1, 2), (2,)), ((1, 2), (3,)), ((2, 3), (1,))]
sage: E1 = ExtensionTypeLong(data, (1,2,3))
sage: R = E1.rec_enum_set_under_sadic([132]*2+[123]*6, S)
sage: A = [E for E,w,h in R.graded_component(8)]
sage: [a.factor() for a in A]
[word: 2322322322322322322,
 word: 2322322322322322322,
 word: 22322322322322322322]
sage: A[1].is_subset(A[0])
True
sage: from slabbe.bispecial_extension_type import remove_extension_types_subsets
sage: B = remove_extension_types_subsets(A)
sage: B
[  E(w)   1   2   3
    21    X   X   X
    22            X
    32    X
 m(w)=1, strong,
   E(w)   1   3
    32        X
    23    X
 m(w)=-1, weak]
sage: [b.factor() for b in B]
[word: 2322322322322322322,
 word: 22322322322322322322]
```

`slabbe.bispecial_extension_type.`**`table_bispecial`**(*word*, *k*, *nleft=1*, *nright=1*)

Return the table of the first k bispecial factors of a word.

INPUT:

- `word` – finite word
- `k` – integer

OUTPUT:

table

EXAMPLES:

```
sage: from slabbe.bispecial_extension_type import table_bispecial
sage: w = words.FibonacciWord()
sage: table_bispecial(w[:10000], 6)
  |w|   word                   m(w)   info   d^-(w)   d^+(w)
+-----+----------------------+------+------+--------+--------+
  0                            0      ord.   2        2
  1     0                      0      ord.   2        2
  3     010                    0      ord.   2        2
  6     010010                 0      ord.   2        2
  11    01001010010            0      ord.   2        2
  19    0100101001001010010    0      ord.   2        2
```

```
sage: w = words.FibonacciWord()
sage: table_bispecial(w[:10000], 6, nleft=2)
  |w|   word                   m(w)   info   d^-(w)   d_2^-(w)   d^+(w)
+-----+----------------------+------+------+--------+----------+--------+
  0                            0      ord.   2        3          2
  1     0                      0      ord.   2        2          2
  3     010                    0      ord.   2        2          2
  6     010010                 0      ord.   2        2          2
  11    01001010010            0      ord.   2        2          2
  19    0100101001001010010    0      ord.   2        2          2
```

```
sage: w = words.ThueMorseWord()
sage: table_bispecial(w[:10000], 11)
  |w|   word     m(w)   info     d^-(w)   d^+(w)
+-----+--------+------+--------+--------+--------+
  0              1      strong   2        2
  1     0        0      ord.     2        2
  1     1        0      ord.     2        2
  2     01       1      strong   2        2
  2     10       1      strong   2        2
  3     010      -1     weak     2        2
  3     101      -1     weak     2        2
  4     0110     1      strong   2        2
  4     1001     1      strong   2        2
  6     011001   -1     weak     2        2
  6     100110   -1     weak     2        2
```

2.3 Finite words

Finite words

Methods that are not in Sage (for now!)

AUTHORS:

- Sébastien Labbé, 2015
- Sébastien Labbé, 2017, added lexicographic Lyndon stuff

EXAMPLES:

```
sage: from slabbe.finite_word import discrepancy
sage: w = words.ChristoffelWord(5,8)
sage: discrepancy(w)
12/13
```

`slabbe.finite_word.`**`are_overlapping_factors`**`(`*u*, *v*, *d*`)`

Returns whether there exists a word w such that u occurs in w at position 0 and v occurs in w at position d.

INPUT:

- u – word
- v – word
- d – integer (positive or negative)

EXAMPLES:

```
sage: from slabbe.finite_word import are_overlapping_factors
sage: are_overlapping_factors('abc', 'abc', 0)
True
sage: are_overlapping_factors('abc', 'abc', 1)
False
sage: are_overlapping_factors('abcdef', 'bc', 1)
True
sage: are_overlapping_factors('abcdef', 'bcdefgh', 1)
True
sage: are_overlapping_factors('abcdef', 'bcddefgh', 1)
False
sage: are_overlapping_factors('abcdef', 'aabcdefgh', -1)
True
sage: are_overlapping_factors('abcdef', 'aabcd', -1)
True
sage: are_overlapping_factors('abcdef', 'aabcc', -1)
False
```

`slabbe.finite_word.`**`discrepancy`**(*self*, *freq=None*)

Return the discrepancy of the word.

This is a distance to the euclidean line defined in *[T1980]*.

INPUT:

- `freq` – frequency vector (default: `None`)

EXAMPLES:

```
sage: from slabbe.finite_word import discrepancy
sage: w = words.ChristoffelWord(5,8)
sage: w
word: 0010010100101
sage: discrepancy(w)
12/13
```

```
sage: for c in w.conjugates(): print (c, discrepancy(c))
0010010100101 12/13
0100101001010 7/13
1001010010100 10/13
0010100101001 10/13
0101001010010 7/13
1010010100100 12/13
0100101001001 8/13
1001010010010 9/13
0010100100101 11/13
0101001001010 6/13
1010010010100 11/13
0100100101001 9/13
1001001010010 8/13
```

REFERENCES:

`slabbe.finite_word.`**`is_lyndon_mod_reverse`**(*self*)

EXAMPLES:

```
sage: from slabbe.finite_word import is_lyndon_mod_reverse
sage: is_lyndon_mod_reverse(Word('111222'))
True
sage: is_lyndon_mod_reverse(Word('1112221'))
False
sage: is_lyndon_mod_reverse(Word('143'))
False
```

```
sage: w = words.ChristoffelWord(72452,462443)
sage: is_lyndon_mod_reverse(w)
True
```

slabbe.finite_word.**minimum_lexicographic_conjugate**(*self*)

Return the conjugate word which is minimal for the lexicographic order.

The output is a Lyndon word (or some power of).

EXAMPLES:

```
sage: from slabbe.finite_word import minimum_lexicographic_conjugate
sage: minimum_lexicographic_conjugate(Word([1,3,2,2,2]))
word: 13222
sage: minimum_lexicographic_conjugate(Word([1,4,3]))
word: 143
sage: minimum_lexicographic_conjugate(Word([3,4,1]))
word: 134
```

The code is fast:

```
sage: w = words.ChristoffelWord(72452, 462443)
sage: minimum_lexicographic_conjugate(w)
word: 0000000100000010000000100000010000001000...
```

slabbe.finite_word.**minimum_lexicographic_conjugate_reversal**(*self*)

TODO: Use Lyndon factorisation to improve the time/space…

EXAMPLES:

```
sage: from slabbe.finite_word import minimum_lexicographic_conjugate_reversal
sage: minimum_lexicographic_conjugate_reversal(Word([1,3,2,2,2]))
word: 12223
```

```
sage: w = words.ChristoffelWord(72452,462443)
sage: minimum_lexicographic_conjugate_reversal(w)
word: 0000000100000010000000100000010000001000...
sage: _ == w
True
sage: minimum_lexicographic_conjugate_reversal(Word([1,3,2,2,1,1,2]))
word: 1121322
```

slabbe.finite_word.**run_length_encoding**(*self*)

EXAMPLES:

```
sage: from slabbe.finite_word import run_length_encoding
sage: L = [0, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3]
sage: run_length_encoding(L)
[(0, 1), (1, 6), (2, 1), (1, 1), (2, 5), (3, 6)]
```

slabbe.finite_word.**sort_word_by_length_lex_key**(*w*)

A key function to sort word (by length, and then lexicographically).

EXAMPLES:

```
sage: from slabbe.finite_word import sort_word_by_length_lex_key
sage: L = ['aa', 'aaa', 'bb', 'ccc']
sage: sorted(L, key=sort_word_by_length_lex_key)
['aa', 'bb', 'aaa', 'ccc']
```

`slabbe.finite_word.`**`to_image`**(*self*, *width=1000*)

Creates an image from a word

INPUT:

- `width` – integer, width of image

EXAMPLES:

```
sage: from slabbe.finite_word import to_image
sage: t = words.ThueMorseWord()
sage: img = to_image(t[:10000], width=100)
sage: img
<PIL.Image.Image image mode=RGB size=100x100 at 0x...>
sage: img.show()    # not tested
```

```
sage: W = FiniteWords(range(10))
sage: d = {a:W.random_element(7) for a in range(10)}
sage: m = WordMorphism(d, codomain=W)
sage: w = m.periodic_points()[0][0]
```

```
sage: s = map(int, str(pi.n(digits=40001))[2:])
sage: len(s)
40000
sage: img = to_image(W(s), 200)
sage: img.show()    # not tested
```

`slabbe.finite_word.`**`word_to_polyomino`**(*self*)

Returns the inside points of a polyomino.

INPUT:

- `self` – list of integers in 0,1,2,3 describing a closed path

OUTPUT:

- list of 2d vectors

EXAMPLES:

```
sage: from slabbe.finite_word import word_to_polyomino
sage: w = [0,0,0,1,1,1,2,2,2,3,3,3]
sage: word_to_polyomino(w)
[(0, 0), (0, 1), (1, 0), (2, 0), (1, 1), (0, 2), (1, 2), (2, 1), (2, 2)]
sage: w = [1,1,1,0,0,0,3,3,3,2,2,2]
sage: word_to_polyomino(w)
[(0, 0), (0, 1), (1, 0), (2, 0), (1, 1), (0, 2), (1, 2), (2, 1), (2, 2)]
```

2.4 Infinite words

Infinite words

Methods that are not in Sage (for now!)

AUTHORS:

- Sébastien Labbé, 2016

EXAMPLES:

```
sage: from slabbe.infinite_word import derived_sequence
sage: w = words.ThueMorseWord()
sage: derived_sequence(w, w[:1])
word: 0120210121020120210201210120210121020121...
```

`slabbe.infinite_word.`**`derived_sequence`**(*self*, *u*, *coding=False*)

Return the derived sequence of according to the return words to a factor of self.

INPUT:

- `u` – finite word, the length of the nonempty prefix
- `coding` – boolean (default: `False`), whether to include the return word coding dictionnary

EXAMPLES:

```
sage: from slabbe.infinite_word import derived_sequence
sage: w = words.ThueMorseWord()
sage: derived_sequence(w, w[:1])
word: 0120210121020120210201210120210121020121...
sage: derived_sequence(w, w[:2])
word: 0123013201232013012301320130123201230132...
sage: derived_sequence(w, w[:3])
word: 0123013201232013012301320130123201230132...
```

With the return word coding:

```
sage: w = words.ThueMorseWord()
sage: derived, D = derived_sequence(w, w[:1], True)
sage: derived
word: 0120210121020120210201210120210121020121...
sage: D
{word: 0: 2, word: 01: 1, word: 011: 0}
```

It gets into a cycle of length 1:

```
sage: words.ThueMorseWord()
word: 0110100110010110100101100110100110010110...
sage: derived_sequence(_, _[:1])
word: 0120210121020120210201210120210121020121...
sage: derived_sequence(_, _[:1])
word: 0123013201232013012301320130123201230132...
sage: derived_sequence(_, _[:1])
word: 0123013201232013012301320130123201230132...
```

Note: Note that method `return_words_derivate` of finite words in Sage does the same for finite words but without returning the translation dictionnary:

```
sage: w = words.ThueMorseWord()
sage: prefix = w[:1000]
sage: prefix.return_words_derivate(prefix[:1])
word: 1231321232131231321312321231321232131232...
```

2.5 Languages

Regular languages

EXAMPLES:

Language over all finite words on an alphabet:

```
sage: from slabbe.language import Language
sage: Language(alphabet=['a', 'b'])
Language of finite words over alphabet ['a', 'b']
```

Finite language:

```
sage: from slabbe.language import FiniteLanguage
sage: S = ['a', 'ab', 'aab', 'aaab']
sage: FiniteLanguage(alphabet=['a', 'b'], words=S)
Finite language of cardinality 4 over alphabet ['a', 'b']
```

Regular language:

```
sage: from slabbe.language import RegularLanguage
sage: alphabet = ['a', 'b']
sage: trans = [(0, 1, 'a'), (1, 2, 'b'), (2, 3, 'b'), (3, 4, 'a')]
sage: automaton = Automaton(trans, initial_states=[0], final_states=[4])
sage: RegularLanguage(alphabet, automaton)
Regular language over ['a', 'b']
defined by: Automaton with 5 states
```

Predefined languages:

```
sage: from slabbe.language import languages
sage: languages.ARP()
Regular language over [1, 2, 3, 123, 132, 213, 231, 312, 321]
defined by: Automaton with 7 states
```

AUTHORS:

- Sébastien Labbé, initial clean and full doctested version, October 2015

`class slabbe.language.`**`FiniteLanguage`**`(`*alphabet*, *words*`)`

Bases: *`slabbe.language.Language`*

Finite language

INPUT:

- `alphabet` – iterable of letters
- `words` – finite iterable of words

EXAMPLES:

```
sage: from slabbe.language import FiniteLanguage
sage: L = ['a', 'aa', 'aaa']
sage: FiniteLanguage(alphabet=['a'], words=L)
Finite language of cardinality 3 over alphabet ['a']
```

`automaton``()`

Return the automaton recognizing this finite language.

EXAMPLES:

```
sage: from slabbe.language import FiniteLanguage
sage: L = ['a', 'aa', 'aaa']
sage: F = FiniteLanguage(alphabet=['a'], words=L)
sage: F.automaton()
Automaton with 7 states
```

`minimal_automaton``()`

Return the minimal automaton recognizing this finite language.

Note: One of the state is not final. You may want to remove it…

EXAMPLES:

```
sage: from slabbe.language import FiniteLanguage
sage: L = ['a', 'aa', 'aaa']
sage: F = FiniteLanguage(alphabet=['a'], words=L)
sage: F.minimal_automaton()
Automaton with 5 states
```

number_of_states()

EXAMPLES:

```
sage: from slabbe.language import FiniteLanguage
sage: L = ['a', 'aa', 'aaa']
sage: F = FiniteLanguage(alphabet=['a'], words=L)
sage: F.number_of_states()
5
```

class slabbe.language.**Language**(*alphabet*)

Bases: `object`

Language of finite words

INPUT:

- `alphabet` – iterable of letters

EXAMPLES:

```
sage: from slabbe.language import Language
sage: Language(alphabet=['a', 'b'])
Language of finite words over alphabet ['a', 'b']
```

complexity(*length*)

Returns the number of words of given length.

Note: This method is defined from *words_of_length_iterator()*.

INPUT:

- `length` – integer

EXAMPLES:

```
sage: from slabbe.language import Language
sage: F = Language(alphabet=['a', 'b'])
sage: map(F.complexity, range(5))
[1, 2, 4, 8, 16]
```

words_of_length_iterator(*length*)

Return an iterator over words of given length.

INPUT:

- `length` – integer

EXAMPLES:

```
sage: from slabbe.language import Language
sage: F = Language(alphabet=['a', 'b'])
sage: it = F.words_of_length_iterator(2)
sage: list(it)
[word: aa, word: ab, word: ba, word: bb]
```

class slabbe.language.**LanguageGenerator**

Bases: `object`

ARP()

Return the Arnoux-Rauzy-Poincaré regular language.

sage: from slabbe.language import languages sage: L = languages.ARP() sage: L Regular language over [1, 2, 3, 123, 132, 213, 231, 312, 321] defined by: Automaton with 7 states sage: map(L.complexity, range(4)) [1, 9, 57, 345]

Brun()

Return the Brun regular language.

EXAMPLES:

```
sage: from slabbe.language import languages
sage: L = languages.Brun()
sage: L
Regular language over [123, 132, 213, 231, 312, 321]
defined by: Automaton with 6 states
sage: map(L.complexity, range(4))
[1, 6, 18, 54]
sage: list(L.words_of_length_iterator(2))
[word: 123,123,
 word: 123,132,
 word: 123,312,
 word: 132,123,
 word: 132,132,
 word: 132,213,
 word: 213,213,
 word: 213,231,
 word: 213,321,
 word: 231,123,
 word: 231,213,
 word: 231,231,
 word: 312,231,
 word: 312,312,
 word: 312,321,
 word: 321,132,
 word: 321,312,
 word: 321,321]
```

Cassaigne()

Return the Cassaigne regular language over the alphabet [11, 22, 122, 211, 121, 212].

EXAMPLES:

```
sage: from slabbe.language import languages
sage: L = languages.Cassaigne()
sage: L
Regular language over [11, 22, 122, 211, 121, 212]
defined by: Automaton with 1 state
sage: map(L.complexity, range(4))
[1, 6, 36, 216]
```

Selmer()

Return the Selmer regular language.

EXAMPLES:

```
sage: from slabbe.language import languages
sage: L = languages.Selmer()
sage: L
Regular language over [123, 132, 213, 231, 312, 321]
defined by: Automaton with 6 states
sage: map(L.complexity, range(4))
[1, 6, 12, 24]
sage: list(L.words_of_length_iterator(2))
[word: 123,132,
 word: 123,312,
 word: 132,123,
 word: 132,213,
 word: 213,231,
 word: 213,321,
 word: 231,123,
 word: 231,213,
 word: 312,231,
 word: 312,321,
 word: 321,132,
 word: 321,312]
```

class slabbe.language.**RegularLanguage**(*alphabet*, *automaton*)

Bases: `slabbe.language.Language`

Regular language

INPUT:

- `alphabet` – iterable of letters
- `automaton` – finite state automaton

EXAMPLES:

```
sage: from slabbe.language import RegularLanguage
sage: alphabet = ['a', 'b']
sage: trans = [(0, 1, 'a'), (1, 2, 'b'), (2, 3, 'b'), (3, 4, 'a')]
sage: automaton = Automaton(trans, initial_states=[0], final_states=[4])
sage: RegularLanguage(alphabet, automaton)
Regular language over ['a', 'b']
defined by: Automaton with 5 states
```

words_of_length_iterator(*length*)

Return an iterator over words of given length.

INPUT:

- `length` – integer

EXAMPLES:

```
sage: from slabbe.language import RegularLanguage
sage: alphabet = ['a', 'b']
sage: trans = [(0, 1, 'a'), (1, 2, 'b'), (2, 3, 'b'), (3, 4, 'a')]
sage: automaton = Automaton(trans, initial_states=[0], final_states=[4])
sage: R = RegularLanguage(alphabet, automaton)
sage: [list(R.words_of_length_iterator(i)) for i in range(6)]
[[], [], [], [], [word: abba], []]
```

2.6 Word Morphisms

Word morphisms methods and iterators

EXAMPLES:

```
sage: from slabbe.word_morphisms import iter_primitive_marked_classP_morphisms
sage: F = FiniteWords('ab')
sage: it = iter_primitive_marked_classP_morphisms(F, 4)
sage: list(it)
[WordMorphism: a->aba, b->a,
 WordMorphism: a->bab, b->a,
 WordMorphism: a->b, b->aba,
 WordMorphism: a->b, b->bab,
 WordMorphism: a->abb, b->a,
 WordMorphism: a->ab, b->aa,
 WordMorphism: a->bb, b->ba,
 WordMorphism: a->b, b->baa]
```

slabbe.word_morphisms.**compute_xsi**(*self*, *u*)

EXAMPLES:

```
sage: from slabbe.word_morphisms import compute_xsi
sage: s = WordMorphism({0:[0,1],1:[1,0]})
sage: compute_xsi(s, Word([0]))
sigma_u= 0->012, 1->02, 2->1
theta_u= 0->011, 1->01, 2->0
psi= 0->(0, 0),(0, 1),(0, 2), 1->(1, 0),(1, 1), 2->(2, 0)
psi*sigma_u= 0->(0, 0),(0, 1),(0, 2),(1, 0),(1, 1),(2, 0), 1->(0, 0),(0, 1),(0, 2),(2, 0), 2->(1, 0),(1,␣
↪1)
Finite words over {(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (2, 0)}
[1 0 0]
[1 0 0]
[1 0 0]
[0 1 0]
[0 1 0]
[0 0 1]
We want zeta such that:
zeta((0, 0),(0, 1),(0, 2)) = (0, 0),(0, 1),(0, 2),(1, 0),(1, 1),(2, 0)
zeta((1, 0),(1, 1)) = (0, 0),(0, 1),(0, 2),(2, 0)
zeta((2, 0)) = (1, 0),(1, 1)
```

slabbe.word_morphisms.**desubstitute**(*self*, *u*)

EXAMPLES:

Unique preimage:

```
sage: from slabbe.word_morphisms import desubstitute
sage: s = WordMorphism({0:[0,1],1:[1,0]})
sage: desubstitute(s, Word([0,1,0,1,1,0]))
[word: 001]
```

Non-unique preimage:

```
sage: s = WordMorphism({0:[0,1],1:[1,0],2:[1,0]})
sage: desubstitute(s, Word([0,1,0,1,1,0]))
[word: 001, word: 002]
```

No preimage:

```
sage: s = WordMorphism({0:[0,1],1:[1,0]})
sage: desubstitute(s, Word([0,1,0,1,1,1]))
[]
```

Lot of preimages (computation is done in parallel with Florent's Hivert parallel map reduce code):

```
sage: s = WordMorphism({0:[0,1],1:[0,1]})
sage: w = Word([0,1]) ^ 10
```

(continues on next page)

(continued from previous page)

```
sage: L = desubstitute(s, w)
sage: len(L)
1024
```

slabbe.word_morphisms.**desubstitute_prefix_code**(*self*, *u*)

Return the preimage of u under self.

INPUT:

- `self` – word morphism, a prefix code
- `u` – word

EXAMPLES:

```
sage: from slabbe.word_morphisms import desubstitute_prefix_code
sage: s = WordMorphism({0:[0,1],1:[1,0]})
sage: w = desubstitute_prefix_code(s, Word([0,1,0,1,1,0]))
sage: w
word: 001
```

The result lives in the domain of the given substitution:

```
sage: w.parent()
Finite words over {0, 1}
```

TESTS:

```
sage: s = WordMorphism({0:[0,1],1:[1,0],2:[1,0]})
sage: desubstitute_prefix_code(s, Word([0,1,0,1,1,0]))
Traceback (most recent call last):
...
ValueError: non unique desubstitution, m(1)=10, m(2)=10 are prefixes of u[4:]
```

```
sage: s = WordMorphism({0:[0,1],1:[1,0]})
sage: desubstitute_prefix_code(s, Word([0,1,0,1,1,1]))
Traceback (most recent call last):
...
ValueError: desubstitution is impossible for u[4:]
```

slabbe.word_morphisms.**is_left_marked**(*m*)

EXAMPLES:

```
sage: from slabbe.word_morphisms import is_left_marked
sage: m = WordMorphism('0->00001,1->00010')
sage: is_left_marked(m)
True
sage: m = WordMorphism('0->00001,1->00001')
sage: is_left_marked(m)
False
sage: m = WordMorphism('0->00001,1->00001,2->201')
sage: is_left_marked(m)
False
sage: m = WordMorphism('0->00001,1->10001,2->201')
sage: is_left_marked(m)
True
sage: m = WordMorphism('0->000001,1->010001,2->0201')
sage: is_left_marked(m)
True
sage: m = WordMorphism('0->000001,1->010001,2->0101')
sage: is_left_marked(m)
False
```

slabbe.word_morphisms.**is_marked**(*m*)

EXAMPLES:

```
sage: from slabbe.word_morphisms import is_marked
sage: m = WordMorphism('0->00001,1->00010')
sage: is_marked(m)
True
```

slabbe.word_morphisms.**iter_conjugate_classP**(*words*, *n*)

EXAMPLES:

```
sage: from slabbe.word_morphisms import iter_conjugate_classP
sage: F = FiniteWords('ab')
sage: list(iter_conjugate_classP(F, 2))
[WordMorphism: a->a, b->a,
 WordMorphism: a->a, b->b,
 WordMorphism: a->b, b->a,
 WordMorphism: a->b, b->b]
sage: list(iter_conjugate_classP(F, 3))
[WordMorphism: a->aa, b->a,
 WordMorphism: a->aa, b->b,
 WordMorphism: a->bb, b->a,
 WordMorphism: a->bb, b->b,
 WordMorphism: a->a, b->aa,
 WordMorphism: a->a, b->bb,
 WordMorphism: a->b, b->aa,
 WordMorphism: a->b, b->bb,
 WordMorphism: a->ba, b->b,
 WordMorphism: a->ab, b->a,
 WordMorphism: a->b, b->ba,
 WordMorphism: a->a, b->ab]
```

slabbe.word_morphisms.**iter_palindromes**(*words*, *length*)

EXAMPLES:

```
sage: from slabbe.word_morphisms import iter_palindromes
sage: list(iter_palindromes(Words('ab'), 2))
[word: aa, word: bb]
sage: list(iter_palindromes(Words('ab'), 3))
[word: aaa, word: aba, word: bab, word: bbb]
sage: list(iter_palindromes(Words('ab'), 4))
[word: aaaa, word: abba, word: baab, word: bbbb]
sage: list(iter_palindromes(Words('ab'), 5))
[word: aaaaa,
 word: aabaa,
 word: ababa,
 word: abbba,
 word: baaab,
 word: babab,
 word: bbabb,
 word: bbbbb]
```

slabbe.word_morphisms.**iter_pisot_irreductible**(*d=3*, *arg=None*)

Return an iterator over Pisot irreductible substitutions

INPUT:

- d – size of alphabet, [0,1,. . . ,d-1]
- “arg” – (optional, default: None) It can be one of the following :
  - “None” – then the method iterates through all morphisms.
  - tuple (a, b) of two integers - It specifies the range “range(a, b)” of values to consider for the sum of the length

EXAMPLES:

```
sage: from slabbe.word_morphisms import iter_pisot_irreductible
sage: it = iter_pisot_irreductible(3)
sage: for _ in range(4): next(it)
WordMorphism: 0->01, 1->2, 2->0
WordMorphism: 0->02, 1->0, 2->1
WordMorphism: 0->10, 1->2, 2->0
WordMorphism: 0->12, 1->0, 2->1
```

Pour linstant, avec le tuple, il y a un bogue:

```
sage: it = iter_pisot_irreductible(3, (5,10))
sage: for _ in range(4): next(it)
WordMorphism: 0->0000001, 1->2, 2->0
WordMorphism: 0->0000002, 1->0, 2->1
WordMorphism: 0->0000010, 1->2, 2->0
WordMorphism: 0->0000012, 1->0, 2->1
```

`slabbe.word_morphisms.`**`iter_primitive_marked_classP_morphisms`**(*words*, *n*)

EXAMPLES:

```
sage: from slabbe.word_morphisms import iter_primitive_marked_classP_morphisms
sage: F = FiniteWords('ab')
sage: it = iter_primitive_marked_classP_morphisms(F, 2)
sage: list(it)
[]
sage: it = iter_primitive_marked_classP_morphisms(F, 3)
sage: list(it)
[WordMorphism: a->ab, b->a, WordMorphism: a->b, b->ba]
sage: it = iter_primitive_marked_classP_morphisms(F, 4)
sage: list(it)
[WordMorphism: a->aba, b->a,
 WordMorphism: a->bab, b->a,
 WordMorphism: a->b, b->aba,
 WordMorphism: a->b, b->bab,
 WordMorphism: a->abb, b->a,
 WordMorphism: a->ab, b->aa,
 WordMorphism: a->bb, b->ba,
 WordMorphism: a->b, b->baa]
```

`slabbe.word_morphisms.`**`return_substitution`**(*self*, *u*, *coding=False*, *length=1000*)

Return the return substitution of self according to factor u.

INPUT:

- `self` – word morphism
- `u` – word such that u is a prefix of self(u)
- `coding` – boolean (default: `False`), whether to include the return word coding morphism
- `length` – integer (default: `1000`), compute the first 1000 letters of the derived sequence to make sure every return word are seen

EXAMPLES:

```
sage: from slabbe.word_morphisms import return_substitution
sage: s = WordMorphism({0:[0,1],1:[1,0]})
sage: return_substitution(s, Word([0]))
WordMorphism: 0->012, 1->02, 2->1
sage: return_substitution(s, Word([0,1]))
WordMorphism: 0->01, 1->23, 2->013, 3->2
sage: return_substitution(s, Word([0,1,1]))
WordMorphism: 0->01, 1->23, 2->013, 3->2
```

```
sage: return_substitution(s, Word([0]), True)
(WordMorphism: 0->012, 1->02, 2->1,
 WordMorphism: 0->011, 1->01, 2->0)
sage: return_substitution(s, Word([0,1]), True)
(WordMorphism: 0->01, 1->23, 2->013, 3->2,
 WordMorphism: 0->011, 1->010, 2->0110, 3->01)
```

```
sage: s = WordMorphism({0:[0,0,1],1:[0,1]})
sage: return_substitution(s, Word([0]))
WordMorphism: 0->01, 1->011
```

TESTS:

```
sage: s = WordMorphism({0:[0,1],1:[1,0]})
sage: sigma_u, theta_u = return_substitution(s, Word([0]), coding=True)
sage: sigma_u
WordMorphism: 0->012, 1->02, 2->1
sage: theta_u
WordMorphism: 0->011, 1->01, 2->0
sage: theta_u*sigma_u == s*theta_u
True
sage: theta_u*sigma_u
WordMorphism: 0->011010, 1->0110, 2->01
```

2.7 Ostrowski numeration

Ostrowski numeration

See *[Ber2001]*.

REFERENCES:

AUTHOR:

- Sébastien Labbé, May 24, 2017

`slabbe.ostrowski.`**`cf_positive_representation`**(*beta*, *alpha*)

`slabbe.ostrowski.`**`ostrowski_integer`**(*n*, *alpha*)

INPUT:

- `n` – integer >= 0
- `alpha` – irrational real number > 0

EXAMPLES:

```
sage: from slabbe.ostrowski import ostrowski_integer
sage: ostrowski_integer(5, golden_ratio)
([0, 0, 0, 0, 1], [1, 1, 2, 3, 5])
sage: ostrowski_integer(10, golden_ratio)
([0, 0, 1, 0, 0, 1], [1, 1, 2, 3, 5, 8])
sage: ostrowski_integer(123456, e)
([0, 0, 1, 0, 2, 0, 0, 5, 0, 0, 5, 0, 1, 6],
 [1, 1, 3, 4, 7, 32, 39, 71, 465, 536, 1001, 8544, 9545, 18089])
sage: ostrowski_integer(123456, pi)
([4, 11, 0, 211, 0, 0, 0, 1],
 [1, 7, 106, 113, 33102, 33215, 66317, 99532])
```

TESTS:

```
sage: ostrowski_integer(10, 4/5)
Traceback (most recent call last):
...
ValueError: alpha (=4/5) must be irrational
```

```
sage: for i in range(10): print(i, ostrowski_integer(i, golden_ratio))
0 ([], [])
1 ([0, 1], [1, 1])
2 ([0, 0, 1], [1, 1, 2])
3 ([0, 0, 0, 1], [1, 1, 2, 3])
4 ([0, 1, 0, 1], [1, 1, 2, 3])
5 ([0, 0, 0, 0, 1], [1, 1, 2, 3, 5])
6 ([0, 1, 0, 0, 1], [1, 1, 2, 3, 5])
7 ([0, 0, 1, 0, 1], [1, 1, 2, 3, 5])
8 ([0, 0, 0, 0, 0, 1], [1, 1, 2, 3, 5, 8])
9 ([0, 1, 0, 0, 0, 1], [1, 1, 2, 3, 5, 8])
```

Digits of numbers from 1 to 24 in base sqrt(2)-1 where (q_k)_0^3=(1,2,5,12) appearing in *[Bou2015]*:

```
sage: rows = [[i]+ostrowski_integer(i, sqrt(2)-1)[0]+[0,0,0,0] for i in range(25)]
sage: table(rows=rows,header_row='N c1 c2 c3 c4'.split())
  N    c1   c2   c3   c4
+----+----+----+----+----+
  0    0    0    0    0
  1    1    0    0    0
  2    0    1    0    0
  3    1    1    0    0
  4    0    2    0    0
  5    0    0    1    0
  6    1    0    1    0
  7    0    1    1    0
  8    1    1    1    0
  9    0    2    1    0
  10   0    0    2    0
  11   1    0    2    0
  12   0    0    0    1
  13   1    0    0    1
  14   0    1    0    1
  15   1    1    0    1
  16   0    2    0    1
  17   0    0    1    1
  18   1    0    1    1
  19   0    1    1    1
  20   1    1    1    1
  21   0    2    1    1
  22   0    0    2    1
  23   1    0    2    1
  24   0    0    0    2
```

slabbe.ostrowski.**ostrowski_real**(*beta*, *alpha*, *stop=10*, *verbose=False*)

this is broken code

EXAMPLES:

```
sage: from slabbe.ostrowski import ostrowski_real
sage: ostrowski_real(golden_ratio^-2, golden_ratio-1, stop=5)    # not tested
Traceback (most recent call last):
...
AssertionError: 0 <= b_2(=3) <= a_2(=1) is false
```

```
sage: ostrowski_real(golden_ratio^-3, golden_ratio-1, stop=5)
([0, 0, 1, 0, 0],
[golden_ratio - 1,
golden_ratio - 2,
```

(continues on next page)

(continued from previous page)

```
2*golden_ratio - 3,
3*golden_ratio - 5,
5*golden_ratio - 8])
```

CHAPTER

THREE

COMBINATORICS

3.1 Joyal Bijection

André Joyal's Bijection

Problem suggested by Doron Zeilberger during a talk done at CRM, Montreal, May 11th, 2012 to compare code in different languages. This is a implementation of the Joyal's Bijection using Sage. It will not win for the most brief code, but it is object oriented, documented, reusable, testable and allows introspection.

AUTHOR:

- Sébastien Labbé, May 12th 2012

TODO:

- Base Endofunction class on sage's FiniteSetMap classes (for both element and parent)

EXAMPLES:

3.1.1 Creation of an endofunction

```
sage: from slabbe import Endofunction
sage: L = [7, 0, 6, 1, 4, 7, 2, 1, 5]
sage: f = Endofunction(L)
sage: f
Endofunction:
[0..8] -> [7, 0, 6, 1, 4, 7, 2, 1, 5]
```

3.1.2 Creation of a double rooted tree

```
sage: from slabbe import DoubleRootedTree
sage: L = [(0,6),(2,1),(3,1),(4,2),(5,7),(6,4),(7,0),(8,5)]
sage: D = DoubleRootedTree(L, 1, 7)
sage: D
Double rooted tree:
Edges: [(0, 6), (2, 1), (3, 1), (4, 2), (5, 7), (6, 4), (7, 0), (8, 5)]
RootA: 1
RootB: 7
```

3.1.3 Joyal's bijection

From the endofunction `f`, we get a double rooted tree:

```
sage: f.to_double_rooted_tree()
Double rooted tree:
Edges: [(0, 6), (2, 1), (3, 1), (4, 2), (5, 7), (6, 4), (7, 0), (8, 5)]
RootA: 1
RootB: 7
```

From the double rooted tree D, we get an endofunction:

```
sage: D.to_endofunction()
Endofunction:
[0..8] -> [7, 0, 6, 1, 4, 7, 2, 1, 5]
```

In fact, we got D from f and vice versa:

```
sage: D == f.to_double_rooted_tree()
True
sage: f == D.to_endofunction()
True
```

3.1.4 Endofunctions are defined on the set [0, 1, . . . , n-1]

As of now, the code supports only endofunctions defined on the set [0, 1, . . . , n-1]

```
sage: L = [1, 0, 3, 4, 5, 7, 1]
sage: f = Endofunction(L)
Traceback (most recent call last):
...
ValueError: images of [0..6] must be 0 <= i < 7
```

3.1.5 Another example

From a list L, we create an endofunction f

```
sage: L = [12, 7, 8, 3, 3, 11, 11, 9, 5, 12, 0, 10, 9]
sage: f = Endofunction(L)
sage: f
Endofunction:
[0..12] -> [12, 7, 8, 3, 3, 11, 11, 9, 5, 12, 0, 10, 9]
```

From f, we create a double rooted tree D:

```
sage: D = f.to_double_rooted_tree(); D
Double rooted tree:
Edges: [(0, 12), (1, 7), (2, 8), (3, 12), (4, 3), (5, 11),
(6, 11), (7, 9), (8, 5), (10, 0), (11, 10), (12, 9)]
RootA: 9
RootB: 3
```

And from D, we create an endofunction:

```
sage: D.to_endofunction()
Endofunction:
[0..12] -> [12, 7, 8, 3, 3, 11, 11, 9, 5, 12, 0, 10, 9]
```

We test that we recover the initial endofunction f:

```
sage: f == f.to_double_rooted_tree().to_endofunction()
True
```

3.1.6 A random example

We define the set of all endofunctions on [0..7]:

```
sage: from slabbe import Endofunctions
sage: E = Endofunctions(8)
sage: E
Endofunctions of [0..7]
```

We choose a random endofunction on the set [0..7]:

```
sage: f = E.random_element()
sage: f                                 # random
Endofunction:
[0..7] -> [5, 5, 0, 4, 5, 0, 1, 1]
```

We construct a double rooted tree from it:

```
sage: f.to_double_rooted_tree()         # random
Double rooted tree:
Edges: [(1, 5), (2, 0), (3, 4), (4, 5), (5, 0), (6, 1), (7, 1)]
RootA: 0
RootB: 5
```

We recover an endofunction from the double rooted tree:

```
sage: f.to_double_rooted_tree().to_endofunction()    # random
Endofunction:
[0..7] -> [5, 5, 0, 4, 5, 0, 1, 1]
```

Finally, we check the bijection:

```
sage: f == f.to_double_rooted_tree().to_endofunction()
True
```

3.1.7 Large random example

```
sage: E = Endofunctions(1000)
sage: f = E.random_element()
sage: f == f.to_double_rooted_tree().to_endofunction()
True
```

TESTS:

We test the limit cases:

```
sage: f = Endofunction([0])
sage: f == f.to_double_rooted_tree().to_endofunction()
True
sage: f = Endofunction([0,1])
sage: f == f.to_double_rooted_tree().to_endofunction()
True
sage: f = Endofunction([1,0])
sage: f == f.to_double_rooted_tree().to_endofunction()
True
```

More extensively:

```
sage: E = Endofunctions(1)
sage: all(f == f.to_double_rooted_tree().to_endofunction() for f in E)
```

(continues on next page)

(continued from previous page)

```
True
sage: E = Endofunctions(2)
sage: all(f == f.to_double_rooted_tree().to_endofunction() for f in E)
True
sage: E = Endofunctions(3)
sage: all(f == f.to_double_rooted_tree().to_endofunction() for f in E)
True
sage: E = Endofunctions(4)
sage: all(f == f.to_double_rooted_tree().to_endofunction() for f in E)
True
```

TIMING TESTS:

When the extension of the file is .sage:

```
sage: E = Endofunctions(3)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 0.02 s, Wall: 0.02 s
sage: E = Endofunctions(4)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 0.22 s, Wall: 0.22 s
sage: E = Endofunctions(5)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 2.82 s, Wall: 2.82 s
sage: E = Endofunctions(6)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 45.66 s, Wall: 45.74 s
```

When the extension of the file is .spyx:

```
sage: E = Endofunctions(3)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 0.02 s, Wall: 0.02 s
sage: E = Endofunctions(4)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 0.21 s, Wall: 0.21 s
sage: E = Endofunctions(5)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 2.71 s, Wall: 2.72 s
sage: E = Endofunctions(6)
sage: time all(f == f.to_double_rooted_tree().to_endofunction() for f in E) # not tested
True
Time: CPU 44.08 s, Wall: 44.17 s
```

When the extension of the file is .sage:

```
sage: E = Endofunctions(1000)
sage: f = E.random_element()
sage: time f == f.to_double_rooted_tree().to_endofunction() # not tested
True
Time: CPU 0.09 s, Wall: 0.09 s
sage: E = Endofunctions(10000)
sage: f = E.random_element()
sage: time f == f.to_double_rooted_tree().to_endofunction()  # not tested
True
Time: CPU 2.23 s, Wall: 2.24 s
```

When the extension of the file is .spyx:

```
sage: E = Endofunctions(1000)
sage: f = E.random_element()
sage: time f == f.to_double_rooted_tree().to_endofunction() # not tested
True
Time: CPU 0.11 s, Wall: 0.11 s
sage: E = Endofunctions(10000)
sage: f = E.random_element()
sage: time f == f.to_double_rooted_tree().to_endofunction() # not tested
True
Time: CPU 2.91 s, Wall: 2.93 s
```

class slabbe.joyal_bijection.**DoubleRootedTree**(*edges*, *rootA*, *rootB*)

Bases: `object`

Returns a double rooted tree.

INPUT:

- `edges` - list of edges
- `rootA` - root A
- `rootB` - root B

EXAMPLES:

```
sage: from slabbe import DoubleRootedTree
sage: edges = [(0,5),(1,2),(2,6),(3,2),(4,1),(5,7),(7,1),(8,2),(9,4)]
sage: D = DoubleRootedTree(edges, 6, 0)
sage: D
Double rooted tree:
Edges: [(0, 5), (1, 2), (2, 6), (3, 2), (4, 1), (5, 7), (7, 1), (8, 2), (9, 4)]
RootA: 6
RootB: 0
```

graph()

EXAMPLES:

```
sage: from slabbe import DoubleRootedTree
sage: edges = [(0,5),(1,2),(2,6),(3,2),(4,1),(5,7),(7,1),(8,2),(9,4)]
sage: D = DoubleRootedTree(edges, 6, 0)
sage: D.graph()
Graph on 10 vertices
```

skeleton()

EXAMPLES:

```
sage: from slabbe import DoubleRootedTree
sage: edges = [(0,5),(1,2),(2,6),(3,2),(4,1),(5,7),(7,1),(8,2),(9,4)]
sage: D = DoubleRootedTree(edges, 6, 0)
sage: D.skeleton()
[0, 5, 7, 1, 2, 6]
```

skeleton_cycles()

EXAMPLES:

```
sage: from slabbe import DoubleRootedTree
sage: edges = [(0,5),(1,2),(2,6),(3,2),(4,1),(5,7),(7,1),(8,2),(9,4)]
sage: D = DoubleRootedTree(edges, 6, 0)
sage: D.skeleton()
[0, 5, 7, 1, 2, 6]
sage: D.skeleton_cycles()
[(0,), (1, 5), (2, 7, 6)]
```

to_endofunction()

EXAMPLES:

```
sage: from slabbe import DoubleRootedTree
sage: edges = [(0,5),(1,2),(2,6),(3,2),(4,1),(5,7),(7,1),(8,2),(9,4)]
sage: D = DoubleRootedTree(edges, 6, 0)
sage: D.to_endofunction()
Endofunction:
[0..9] -> [0, 5, 7, 2, 1, 1, 2, 6, 2, 4]
```

TESTS:

```
sage: D = DoubleRootedTree([], 0, 0)
sage: D.to_endofunction()
Endofunction:
[0..0] -> [0]
```

class slabbe.joyal_bijection.**Endofunction**(*L*)

Bases: `object`

Returns an endofunction.

INPUT:

- L - list of length n containing images of the integers from 0 to n-1 where the images belong to the integers from 0 to n-1.

EXAMPLES:

```
sage: from slabbe import Endofunction
sage: L = [0, 5, 7, 2, 1, 1, 2, 6, 2, 4]
sage: f = Endofunction(L)
sage: f
Endofunction:
[0..9] -> [0, 5, 7, 2, 1, 1, 2, 6, 2, 4]
```

cycle_elements()

Returns the list of all elements in a cycle for this endofunction.

OUTPUT:

list

EXAMPLES:

```
sage: from slabbe import Endofunction
sage: L = [6, 5, 7, 2, 1, 1, 2, 6, 2, 4]
sage: f = Endofunction(L)
sage: f.cycle_elements()   # random order
[0, 6, 7, 2, 1, 5]
```

Note: `G.cycle_basis()` is not implemented for directed or multiedge graphs (in Networkx). Hence, the `cycle_basis` method is missing the 2-cycles.

skeleton()

Return the skeleton of the endofunction.

OUTPUT:

list

EXAMPLES:

```
sage: from slabbe import Endofunction
sage: L = [0, 5, 7, 2, 1, 1, 2, 6, 2, 4]
sage: f = Endofunction(L)
sage: f.skeleton()
[0, 5, 7, 1, 2, 6]
```

to_double_rooted_tree()

Return the double rooted tree following André Joyal Bijection.

OUTPUT:

Double rooted tree

EXAMPLES:

```
sage: from slabbe import Endofunction
sage: L = [0, 5, 7, 2, 1, 1, 2, 6, 2, 4]
sage: f = Endofunction(L)
sage: f.to_double_rooted_tree()
Double rooted tree:
Edges: [(0, 5), (1, 2), (2, 6), (3, 2), (4, 1), (5, 7), (7, 1), (8, 2), (9, 4)]
RootA: 6
RootB: 0
```

two_cycle_elements()

Iterator over elements in a two-cycle.

EXAMPLES:

```
sage: from slabbe import Endofunction
sage: L = [0, 5, 7, 2, 1, 1, 2, 6, 2, 4]
sage: f = Endofunction(L)
sage: list(f.two_cycle_elements())
[1, 5]
```

class slabbe.joyal_bijection.**Endofunctions**(*n*)

Bases: `object`

Returns the set of all endofunction on the set [0..n-1].

INPUT:

- `n` - positive integer

EXAMPLES:

```
sage: from slabbe import Endofunctions
sage: Endofunctions(10)
Endofunctions of [0..9]
```

random_element()

Return a random endofunction on [0..n-1].

EXAMPLES:

```
sage: from slabbe import Endofunctions
sage: E = Endofunctions(10)
sage: E.random_element()          # random
Endofunction:
[0..9] -> [2, 8, 7, 0, 0, 6, 2, 3, 5, 9]
sage: E.random_element()          # random
Endofunction:
[0..9] -> [8, 7, 7, 5, 4, 1, 0, 3, 8, 6]
```

3.2 Percolation in lattices

Bond Percolation

This is an implementation of bond percolation. See Chapter 3 of *[POG]*. See also my blog post Percolation and self-avoiding walks related to this code.

AUTHORS:

- Sebastien Labbe (2012-12-17): initial version, for pog lecture group

REFERENCES:

EXAMPLES:

3.2.1 Bond percolation sample

We construct a bond percolation sample in dimension d=2 with probability of open edges p=0.3:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(p=0.3, d=2)
sage: S
Bond percolation sample d=2 p=0.300
```

An edge is defined uniquely as a starting point in Z^d and an axis direction given by an integer i such that 1 <= i <= d. One may ask if a given edge is in the sample S:

```
sage: ((34,56), 2) in S     # random
False
```

The result is cached so the same answer is returned again:

```
sage: ((34,56), 2) in S     # random
False
sage: ((34,56), 2) in S     # random
False
```

The cluster containing the point zero is returned as an iterator:

```
sage: S.cluster()
<generator object at ...>
```

It may be finite of infinite. If you believe it is finite, you may compute its cardinality. If the cluster is infinite, it will not halt:

```
sage: S.cluster_cardinality() # not tested, might not halt
13
```

For larger values of p, the cluster might be larger if not infinite. In this case you may want to stop the computation at a certain point determined in advance. The following method does this. And it returns the cardinality if it is smaller than the stop value:

```
sage: S = BondPercolationSample(p=0.45, d=2)
sage: S.cluster_cardinality_stop_at(stop=10)          # random
'>=10'
sage: S.cluster_cardinality_stop_at(stop=100)          # random
'>=100'
sage: S.cluster_cardinality_stop_at(stop=1000)          # random
625
```

3.2.2 Bond percolation samples

Construction of 20 bond percolation samples. For each of them, compute the cardinality of the open cluster containing zero:

```
sage: from slabbe import BondPercolationSamples
sage: S20 = BondPercolationSamples(p=0.4, d=2, n=20)
sage: S20.cluster_cardinality(stop=100)              # random
[4, 2, 1, 4, 1, 10, 62, 71, 1, 25, 19, 2, 2, 42, '>=100', 1, 18, 2, '>=100', 20]
```

By considering "larger than 100" to be an infinite cluster, this gives a value of 2/20 = 0.10 for the percolation probability:

```
sage: S20.percolation_probability(stop=100)        # random
0.100
```

By increasing the stop value, the computations can be redone again *on the same samples*. In this case, by using a stop value of 1000, we get a value of 0 for the percolation probability of p=0.4:

```
sage: S20.cluster_cardinality(stop=1000)             # random
[4, 2, 1, 4, 1, 10, 62, 71, 1, 25, 19, 2, 2, 42, 176, 1, 18, 2, 186, 20]
sage: S20.percolation_probability(stop=1000)         # random
0.000
```

3.2.3 Percolation probability

One can define the percolation probability function for a given dimension d. It will generate n samples and consider the cluster to be infinite if its cardinality is larger than the given stop value:

```
sage: from slabbe import PercolationProbability
sage: T = PercolationProbability(d=2, n=10, stop=100)
sage: T
Percolation Probability $\theta(p)$
d = dimension = 2
n = # samples = 10
stop counting at = 100
```

Compute the value for a certain probability p:

```
sage: T(0.4534)             # random
0.300
```

Of course, this value will change for another equal percolation probability since it depends on the samples:

```
sage: T = PercolationProbability(d=2, n=10, stop=100)
sage: T(0.4534)              # random
0.600
```

Anyway, it is useful to draw the plot of the percolation probability:

```
sage: T = PercolationProbability(d=2, n=10, stop=100)
sage: T.return_plot((0,1), adaptive_recursion=4, plot_points=4)     # optional long
Graphics object consisting of 2 graphics primitives
```

Here we use Sage adaptative recursion algorithm for drawing plots which finds the particular important intervals to ask for more values of the function. See help section of plot function for details. Because T might be long to compute we start with only 4 points

3.2.4 TODO

- Make it 100% doctested (presently 21/24 = 87%)
- Base it on DiscreteSubset code
- Fix tikz2pdf use

Do we want to use?:

```
sage: from sage.sets.set_from_iterator import EnumeratedSetFromIterator
sage: from itertools import count
sage: S = EnumeratedSetFromIterator(count)
sage: S
{0, 1, 2, 3, 4, ...}
```

and ?:

```
sage: M = FiniteSetMaps(["a", "b"], [3, 4, 5]); M
Maps from {'a', 'b'} to {3, 4, 5}
```

3.2.5 Methods and classes

class slabbe.bond_percolation.**BondPercolationSample**(*p*, *d=2*)

Bases: `sage.structure.sage_object.SageObject`

Let $L^d = (Z^d,E^d)$ be the hypercubic lattice.

A sample contained in the set {0,1}^{E^d}.

An edge e in E is open (=1) in the sample with probability p.

Cached __contains__ method below does the job of memory.

children(*pt*)

Return an iterator over open neighbors of the point pt.

INPUT:

- `pt` - tuple, point in Z^d
- `m` - integer, limit

EXAMPLES:

The result is consistent:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(0.5)
sage: list(S.children((0,0)))        # random
[(1, 0), (-1, 0), (0, -1)]
```

Might be different for another sample:

```
sage: S = BondPercolationSample(0.5)
sage: list(S.children((0,0)))        # random
[(-1, 0)]
```

In dimension 3:

```
sage: S = BondPercolationSample(0.5,3)
sage: list(S.children((0,0,0)))        # random
[(1, 0, 0), (0, -1, 0), (0, 0, -1)]
sage: S = BondPercolationSample(0.5,3)
sage: list(S.children((0,0,0)))        # random
[(1, 0, 0), (-1, 0, 0)]
sage: list(S.children((0,0,0)))        # random
[(1, 0, 0), (-1, 0, 0)]
```

```
sage: S = BondPercolationSample(1,2)
sage: list(S.children((0,0)))        # random
[(1, 0), (-1, 0), (0, 1), (0, -1)]
sage: S = BondPercolationSample(1,3)
sage: list(S.children((0,0,0)))        # random
[(1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0), (0, 0, 1), (0, 0, -1)]
```

cluster(*pt=None*)

Return an iterator over the open cluster containing the point pt.

INPUT:

- `pt` - tuple, point in Z^d. If None, pt=zero is considered.

EXAMPLES:

sage: from slabbe import BondPercolationSample sage: S = BondPercolationSample(0.5) sage: it = S.cluster() sage: next(it) (0, 0)

cluster_cardinality(*pt=None*)

INPUT:

- `pt` - tuple, point in Z^d. If None, pt=zero is considered.

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: BondPercolationSample(0.01).cluster_cardinality() # random
1
sage: BondPercolationSample(0.4).cluster_cardinality()  # random
28
```

cluster_cardinality_stop_at(*stop*, *pt=None*)

Return the cardinality of the cluster or the strin ">=STOP" if the size is larger than stop value.

INPUT:

- `stop` - integer
- `pt` - tuple, point in Z^d. If None, pt=zero is considered.

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(0.3,2)
sage: S.cluster_cardinality()               # random
13
sage: S.cluster_cardinality_stop_at(1000)    # random
13
sage: S.cluster_cardinality_stop_at(100)     # random
13
sage: S.cluster_cardinality_stop_at(10)      # random
'>=10'
```

cluster_in_box(*m*, *pt=None*)

Return the cluster (as a list) in the primal box [-m,m]^d containing the point pt.

INPUT:

- `m` - integer
- `pt` - tuple, point in Z^d. If None, pt=zero is considered.

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(0.3,2)
sage: S.cluster_in_box(2)         # random
[(-2, -2), (-2, -1), (-1, -2), (-1, -1), (-1, 0), (0, 0)]
```

edges_in_box(*m*)

Return an iterator over all edges in the primal box [-m,m]^d.

INPUT:

- `m` - integer

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(1,2)
sage: for a in S.edges_in_box(1): print(a)
((-1, -1), (0, -1))
((-1, -1), (-1, 0))
((-1, 0), (0, 0))
((-1, 0), (-1, 1))
((0, -1), (1, -1))
((0, -1), (0, 0))
((0, 0), (1, 0))
((0, 0), (0, 1))
```

neighbor(*pt*, *d*)

Return the neighbors of the point pt in direction d.

INPUT:

- `pt` - tuple, point in Z^d
- `direction` - integer, possible values are 1, 2, . . . , d and -1, -2, . . . , -d.

EXAMPLES:

sage: from slabbe import BondPercolationSample sage: S = BondPercolationSample(0.5,2) sage: S.neighbor((2,3),1) (3, 3) sage: S.neighbor((2,3),2) (2, 4) sage: S.neighbor((2,3),-1) (1, 3) sage: S.neighbor((2,3),-2) (2, 2)

plot(*m*, *pointsize=100*, *thickness=3*, *axes=False*)

Return 2d graphics object contained in the primal box [-m,m]^d.

INPUT:

- `pointsize`, integer (default:`100`),
- `thickness`, integer (default:3),
- `axes`, bool (default:`False`),

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(0.5,2)
sage: S.plot(2)           # optional long
```

It works in 3d!!:

```
sage: S = BondPercolationSample(0.5,3)
sage: S.plot(3, pointsize=10, thickness=1)     # optional long
Graphics3d Object
```

save_pdf(*m*)

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: BondPercolationSample(0.3, d=2).save_pdf(20)    # optional long
Creation du fichier tikz_sample_d2_p300_m20.tikz
Using template '/Users/slabbe/.tikz2pdf.tex'.
tikz2pdf: calling pdflatex...
tikz2pdf: Output written to 'tikz_sample_d2_p300_m20.pdf'.
```

tikz(*m*)

Return tikz code.

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(0.5,2)
sage: S.tikz(2)
\begin{tikzpicture}
[inner sep=0pt,thick,
 reddot/.style={fill=red,draw=red,circle,minimum size=5pt}]
\clip (-2.4, -2.4) rectangle (2.4, 2.4);
\draw (..., ...) -- (..., ...);
...
\node[circle,fill=none,draw=red,minimum size=0.8cm,ultra thick,inner sep=0pt] at (0,0) {};
\node[above right] at (0,0) {$(0, 0)$};
\end{tikzpicture}
```

zero()

EXAMPLES:

```
sage: from slabbe import BondPercolationSample
sage: S = BondPercolationSample(0.5,3)
sage: S.zero()
(0, 0, 0)
```

```
sage: S = BondPercolationSample(0.5,5)
sage: S.zero()
(0, 0, 0, 0, 0)
```

class slabbe.bond_percolation.**BondPercolationSamples**(*p*, *d*, *n*)

Bases: `sage.structure.sage_object.SageObject`

Return a list of n BondPercolationSample of given parameter p and dimension d.

EXAMPLES:

```
sage: from slabbe import BondPercolationSamples
sage: BondPercolationSamples(0.2,2,3)
<slabbe.bond_percolation.BondPercolationSamples object at ...>
```

cluster_cardinality(*stop*)

Return the list of cardinality of the cluster for each sample or +Infinity if the size is larger than stop value.

INPUT:

- `stop` - integer

EXAMPLES:

```
sage: from slabbe import BondPercolationSamples
sage: S20 = BondPercolationSamples(p=0.2, d=2, n=20)
sage: S20.cluster_cardinality(100)       # random
[1, 4, 1, 2, 2, 6, 5, 1, 5, 9, 2, 2, 2, 1, 2, 4, 4, 3, 2, 1]
```

```
sage: d = 2
sage: n = 5
sage: for p in srange(0,1,0.1): print(p,BondPercolationSamples(p,d,n).cluster_cardinality(100)) #␣
↪optional long
0.000000000000000 [1, 1, 1, 1, 1]
0.100000000000000 [5, 1, 2, 1, 2]
0.200000000000000 [3, 1, 4, 1, 1]
0.300000000000000 [11, 7, 2, 1, 3]
0.400000000000000 [3, 3, 35, 10, '>=100']
0.500000000000000 ['>=100', '>=100', '>=100', '>=100', 26]
0.600000000000000 ['>=100', '>=100', '>=100', '>=100', '>=100']
0.700000000000000 ['>=100', '>=100', '>=100', '>=100', '>=100']
0.800000000000000 ['>=100', '>=100', '>=100', '>=100', '>=100']
0.900000000000000 ['>=100', '>=100', '>=100', '>=100', '>=100']
```

ntimes_over_size(*stop*)

EXAMPLES:

```
sage: from slabbe import BondPercolationSamples
sage: S = BondPercolationSamples(0.2,2,20)
sage: S.ntimes_over_size(100)     # random
0
sage: S = BondPercolationSamples(0.4,2,20)
sage: S.ntimes_over_size(100)     # random
1
sage: S = BondPercolationSamples(0.5,2,20)
sage: S.ntimes_over_size(100)     # random
17
```

percolation_probability(*stop*)

class slabbe.bond_percolation.**PercolationProbability**(*d*, *n*, *stop*, *verbose=False*)

Bases: `sage.structure.sage_object.SageObject`

EXAMPLES:

```
sage: from slabbe import PercolationProbability
sage: f = PercolationProbability(d=2, n=10, stop=100)
sage: f
Percolation Probability $\theta(p)$
d = dimension = 2
n = # samples = 10
stop counting at = 100
```

return_plot(*interval=(0, 1)*, *adaptive_recursion=4*, *plot_points=4*, *adaptive_tolerance=0.1*)

Return a plot of percolation probability using basic sage plot settings.

INPUT:

- `interval`, default=(0,1)
- `adaptive_recursion`, default=0
- `plot_points`, default=10
- `adaptive_tolerance` default=0.10

EXAMPLES:

```
sage: from slabbe import PercolationProbability
sage: T = PercolationProbability(d=2, n=10, stop=100)
sage: T.return_plot()           # optional long
Graphics object consisting of 1 graphics primitive
```

slabbe.bond_percolation.**compute_percolation_probability**(*range_p*, *d*, *n*, *stop*)

EXAMPLES:

```
sage: from slabbe.bond_percolation import compute_percolation_probability
sage: compute_percolation_probability(srange(0,0.8,0.1), d=2, n=5, stop=100) # random
d = 2, n = number of samples = 5
stop counting at = 100
p=0.0000, Theta=0.000, if |C|< 100 then max|C|=1
p=0.1000, Theta=0.000, if |C|< 100 then max|C|=1
p=0.2000, Theta=0.000, if |C|< 100 then max|C|=5
p=0.3000, Theta=0.000, if |C|< 100 then max|C|=6
p=0.4000, Theta=0.000, if |C|< 100 then max|C|=31
p=0.5000, Theta=1.00, if |C|< 100 then max|C|=-Infinity
p=0.6000, Theta=1.00, if |C|< 100 then max|C|=-Infinity
p=0.7000, Theta=1.00, if |C|< 100 then max|C|=-Infinity
```

```
sage: range_p = srange(0,0.8,0.1)
sage: compute_percolation_probability(range_p, d=2, n=5, stop=100) # not tested
d = 2, n = number of samples = 5
stop counting at = 100
p=0.0000, Theta=0.000, if |C|< 100 then max|C|=1
p=0.1000, Theta=0.000, if |C|< 100 then max|C|=1
p=0.2000, Theta=0.000, if |C|< 100 then max|C|=5
p=0.3000, Theta=0.000, if |C|< 100 then max|C|=6
p=0.4000, Theta=0.000, if |C|< 100 then max|C|=31
p=0.5000, Theta=1.00, if |C|< 100 then max|C|=-Infinity
p=0.6000, Theta=1.00, if |C|< 100 then max|C|=-Infinity
p=0.7000, Theta=1.00, if |C|< 100 then max|C|=-Infinity
```

```
sage: range_p = srange(0.45,0.55,0.01)
sage: compute_percolation_probability(range_p, d=2, n=10, stop=1000) # not tested
d = 2, n = number of samples = 10
stop counting at = 1000
p=0.4500, Theta=0.000, if |C|< 1000 then max|C|=378
p=0.4600, Theta=0.000, if |C|< 1000 then max|C|=475
p=0.4700, Theta=0.000, if |C|< 1000 then max|C|=514
p=0.4800, Theta=0.100, if |C|< 1000 then max|C|=655
p=0.4900, Theta=0.700, if |C|< 1000 then max|C|=274
p=0.5000, Theta=0.700, if |C|< 1000 then max|C|=975
p=0.5100, Theta=0.700, if |C|< 1000 then max|C|=16
p=0.5200, Theta=0.700, if |C|< 1000 then max|C|=125
p=0.5300, Theta=0.900, if |C|< 1000 then max|C|=4
p=0.5400, Theta=0.700, if |C|< 1000 then max|C|=6
```

```
sage: range_p = srange(0.475,0.485,0.001)
sage: compute_percolation_probability(range_p, d=2, n=10, stop=1000) # not tested
d = 2, n = number of samples = 10
stop counting at = 1000
p=0.4750, Theta=0.200, if |C|< 1000 then max|C|=718
p=0.4760, Theta=0.200, if |C|< 1000 then max|C|=844
p=0.4770, Theta=0.200, if |C|< 1000 then max|C|=566
p=0.4780, Theta=0.500, if |C|< 1000 then max|C|=257
p=0.4790, Theta=0.200, if |C|< 1000 then max|C|=566
p=0.4800, Theta=0.300, if |C|< 1000 then max|C|=544
p=0.4810, Theta=0.300, if |C|< 1000 then max|C|=778
p=0.4820, Theta=0.500, if |C|< 1000 then max|C|=983
p=0.4830, Theta=0.300, if |C|< 1000 then max|C|=473
p=0.4840, Theta=0.500, if |C|< 1000 then max|C|=411
```

```
sage: range_p = srange(0.47,0.48,0.001)
sage: compute_percolation_probability(range_p, d=2, n=20, stop=2000)  # not tested
d = 2, n = number of samples = 20
stop counting at = 2000
p=0.4700, Theta=0.0500, if |C|< 2000 then max|C|=1666
p=0.4710, Theta=0.100, if |C|< 2000 then max|C|=1665
p=0.4720, Theta=0.000, if |C|< 2000 then max|C|=1798
p=0.4730, Theta=0.0500, if |C|< 2000 then max|C|=1717
p=0.4740, Theta=0.150, if |C|< 2000 then max|C|=1924
p=0.4750, Theta=0.150, if |C|< 2000 then max|C|=1893
p=0.4760, Theta=0.150, if |C|< 2000 then max|C|=1458
p=0.4770, Theta=0.150, if |C|< 2000 then max|C|=1573
p=0.4780, Theta=0.200, if |C|< 2000 then max|C|=1762
p=0.4790, Theta=0.250, if |C|< 2000 then max|C|=951
```

slabbe.bond_percolation.**percolation_graphics_array**(*range_p, d, m, ncols=3*)

EXAMPLES:

```
sage: from slabbe.bond_percolation import percolation_graphics_array
sage: percolation_graphics_array(srange(0.1,1,0.1), d=2, m=5)     # optional long
sage: P = percolation_graphics_array(srange(0.45,0.55,0.01), d=2, m=5)  # optional long
sage: P.save('array_p45_p55_m5.png')     # not tested
sage: P = percolation_graphics_array(srange(0.45,0.55,0.01), d=2, m=10) # optional long
sage: P.save('array_p45_p55_m10.png')     # not tested
```

3.3 Dyck Word in 3d

3d DyckWords

Generalisation of Dyck Word to Surface of cubes in the n x n x n cube above the plane x+y+z=2n.

EXAMPLES:

```
sage: from slabbe.dyck_3d import DyckBlocks3d
sage: L = [len(DyckBlocks3d(i)) for i in range(1, 7)]     # not tested
[1, 2, 9, 96, 2498, 161422]
```

```
sage: L = [1, 2, 9, 96, 2498, 161422]
sage: oeis.find_by_subsequence(L)                        # not tested internet
0: A115965: Number of planar subpartitions of size n pyramidal planar partition.
```

AUTHOR:

- Sébastien Labbé, 31 october 2014

slabbe.dyck_3d.**DyckBlocks3d**(*n*)

EXAMPLES:

```
sage: from slabbe.dyck_3d import DyckBlocks3d
sage: L = [len(DyckBlocks3d(i)) for i in range(1, 6)]
sage: L
[1, 2, 9, 96, 2498]
```

slabbe.dyck_3d.**Possible**(*n*)

Possible stack of DyckWords inside a n x n cube.

EXAMPLES:

```
sage: from slabbe.dyck_3d import Possible
sage: Possible(1)
```

(continues on next page)

(continued from previous page)

```
The Cartesian product of ({[1, 0]},)
sage: Possible(2)
The Cartesian product of ({[1, 1, 0, 0]}, {[1, 0, 1, 0], [1, 1, 0, 0]})
sage: Possible(3).list()
[([1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 0, 0], [1, 0, 1, 0, 1, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 0, 0], [1, 0, 1, 1, 0, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 0, 0], [1, 1, 0, 0, 1, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 0, 0], [1, 1, 0, 1, 0, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 0, 0], [1, 1, 1, 0, 0, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 0, 1, 0, 1, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 0, 1, 1, 0, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 1, 0, 0, 1, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 1, 0, 1, 0, 0]),
 ([1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0], [1, 1, 1, 0, 0, 0])]
```

slabbe.dyck_3d.**is_larger_than**(*x*, *y*)

EXAMPLES:

```
sage: from slabbe.dyck_3d import is_larger_than
sage: w = DyckWord([1,1,1,0,0,0])
sage: w.heights()
(0, 1, 2, 3, 2, 1, 0)
sage: z = DyckWord([1,0,1,0,1,0])
sage: is_larger_than(w,z)
True
sage: is_larger_than(z,w)
False
sage: is_larger_than(w,w)
True
```

3.4 Combinatorics

Combinatorics methods

EXEMPLES:

```
sage: from slabbe.combinat import random_composition
sage: c = random_composition(24,9)
sage: c        # random
[1, 4, 2, 2, 3, 5, 4, 2, 1]
```

```
sage: from slabbe.combinat import random_simplex_point
sage: random_simplex_point(3)      # random
[0.2493321790694003, 0.5353600549544871, 0.21530776597611256]
```

```
sage: from slabbe.combinat import random_interior_point
sage: p = polytopes.hypercube(3)
sage: p = p + vector([20,0,0])
sage: random_interior_point(p)          # random
(19.33174562788114, -0.5428002756082744, -0.3568284089832092)
```

slabbe.combinat.**integral_points_count_union_of_polytopes**(*L*)

Return the cardinality of an union of polytopes.

See https://en.wikipedia.org/wiki/Inclusion–exclusion_principle

INPUT:

- L – list of polytopes

EXEMPLES:

```
sage: P = Polyhedron(ieqs=[[0,1,0],[0,0,1],[9,-1,0],[9,0,-1]])
sage: Q = Polyhedron(ieqs=[[-5,1,0],[-5,0,1],[14,-1,0],[14,0,-1]])
sage: P.integral_points_count()   # optional -- latte_int
100
sage: Q.integral_points_count()   # optional -- latte_int
100
sage: from slabbe.combinat import integral_points_count_union_of_polytopes
sage: integral_points_count_union_of_polytopes([P,Q])  # optional -- latte_int
175
```

slabbe.combinat.**intersection_of_polytopes**(*L*)

Return the intersection of a list of polytopes.

INPUT:

- L – list of polytopes

EXEMPLES:

```
sage: from slabbe.combinat import intersection_of_polytopes
sage: P = Polyhedron(ieqs=[[0,1,0],[0,0,1],[9,-1,0],[9,0,-1]])
sage: Q = Polyhedron(ieqs=[[-5,1,0],[-5,0,1],[14,-1,0],[14,0,-1]])
sage: I = intersection_of_polytopes([P,Q])
sage: I
A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 4 vertices
sage: I.integral_points_count()      # optional -- latte_int
25
```

TESTS:

```
sage: intersection_of_polytopes(iter([P,Q]))
A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 4 vertices
sage: intersection_of_polytopes(iter([]))
Traceback (most recent call last):
...
NotImplementedError: intersection of an empty list of polytopes not defined
```

slabbe.combinat.**non_uniform_randint**(*L*)

Return a random integer from 0 to len(L)-1 with probabilities proportional to the (integral) entries of L.

INPUT:

- L – list of integers

EXEMPLES:

```
sage: from slabbe.combinat import non_uniform_randint
sage: non_uniform_randint([2,3,5])    # random
1
sage: non_uniform_randint([2,3,5])    # random
2
sage: non_uniform_randint([2,3,5])    # random
2
sage: from collections import Counter
sage: L = [non_uniform_randint([2,3,5]) for _ in range(100000)]
sage: Counter(L)            # random
Counter({2: 49805, 1: 30228, 0: 19967})
```

slabbe.combinat.**random**()

random() -> x in the interval [0, 1).

slabbe.combinat.**random_composition**(*n*, *length*)

EXEMPLES:

```
sage: from slabbe.combinat import random_composition
sage: random_composition(4,2)   # random
[1, 3]
sage: random_composition(4,2)   # random
[2, 2]
sage: random_composition(4,2)   # random
[3, 1]
sage: c = random_composition(24,9)
sage: c  # random
[1, 4, 2, 2, 3, 5, 4, 2, 1]
sage: sum(c)
24
```

Because this is very slow!!:

```
sage: C = Compositions(24, length=9)
sage: %time C.random_element()     # not tested
CPU times: user 43.3 s, sys: 31.9 ms, total: 43.3 s
Wall time: 43.2 s
[2, 2, 5, 2, 8, 1, 1, 2, 1]
```

`slabbe.combinat.`**`random_interior_point`**(*self*, *a=10*, *integer=False*)

Return a random interior point of a polytope.

INPUT:

- `a` – number, amplitude of random deplacement in the direction of each ray.
- `integer` – bool, whether the output must be with integer coordinates

EXEMPLES:

```
sage: from slabbe.combinat import random_interior_point
sage: p = polytopes.hypercube(3)
sage: p = p + vector([20,0,0])
sage: p.center()
(20, 0, 0)
sage: random_interior_point(p)     # random
(19.33174562788114, -0.5428002756082744, -0.3568284089832092)
sage: random_interior_point(p)     # random
(20.039169786976075, -0.4121594862234468, -0.05623023234688396)
sage: random_interior_point(p, integer=True)     # random
(21, 0, 0)
```

`slabbe.combinat.`**`random_interior_point_compact_polytope`**(*self*, *uniform='simplex'*, *integer=False*)

Return a random interior point of a compact polytope.

INPUT:

- `uniform` – `'points'` (slow) or `'simplex'` (fast), whether to take the probability uniformly with respect to the set of integral points or with respect to the simplexes.
- `integer` – bool, whether the output must be with integer coordinates

EXEMPLES:

```
sage: from slabbe.combinat import random_interior_point_compact_polytope
sage: p = polytopes.hypercube(3)
sage: p = p + vector([30,20,10])
sage: p.center()
(30, 20, 10)
sage: random_interior_point_compact_polytope(p)     # random
(19.33174562788114, -0.5428002756082744, -0.3568284089832092)
sage: random_interior_point_compact_polytope(p)     # random
```

(continues on next page)

(continued from previous page)

```
(20.039169786976075, -0.4121594862234468, -0.05623023234688396)
sage: random_interior_point_compact_polytope(p, integer=True) # random
(30, 19, 9)
```

slabbe.combinat.**random_interior_point_simplex**(*self*, *integer=False*)

Return a random interior point of a simplex.

This method was based on the code `P.center()` of sage.

INPUT:

- `integer` – bool, whether the output must be with integer coordinates

EXEMPLES:

```
sage: from slabbe.combinat import random_interior_point_simplex
sage: P = 10 * polytopes.simplex(3)
sage: random_interior_point_simplex(P)          # random
(2.8787864522849462, 5.302173919578364, 1.7059355910006113, 0.11310403713607808)
sage: a = random_interior_point_simplex(P, integer=True)
sage: a             # random
(0, 7, 1, 2)
sage: a in P
True
```

slabbe.combinat.**random_simplex_point**(*d*)

Return a random vector of d positive real numbers summing to 1.

INPUT:

- d – integer

EXEMPLES:

```
sage: from slabbe.combinat import random_simplex_point
sage: random_simplex_point(7)          # random
[0.06137280030263492,
 0.08066113584919432,
 0.09019666554921013,
 0.24473802319989957,
 0.41761622259683495,
 0.10043545384643937,
 0.004979698655786735]
sage: random_simplex_point(2)          # random
[0.5677654878488222, 0.4322345121511778]
sage: random_simplex_point(3)          # random
[0.2493321790694003, 0.5353600549544871, 0.21530776597611256]
```

TESTS:

```
sage: sum(random_simplex_point(4))
1.0
sage: sum(random_simplex_point(7))
1.0
```

3.5 Graphs

Functions on graphs

slabbe.graph.**clean_sources_and_sinks**(*G*)

Return a copy of the graph where every vertices of the graph that have in or out degree 0 is removed (recursively).

EXAMPLES:

```
sage: from slabbe.graph import clean_sources_and_sinks
sage: L = [(0,1),(1,2),(2,3),(3,4),(4,5),(5,3)]
sage: G = DiGraph(L,format='list_of_edges')
sage: H = clean_sources_and_sinks(G)
sage: H
Digraph on 3 vertices
sage: H.vertices()
[3, 4, 5]
```

```
sage: L = [(0,1),(1,2),(2,3),(3,4),(4,5),(5,3),(1,0)]
sage: G = DiGraph(L, format='list_of_edges')
sage: H = clean_sources_and_sinks(G)
sage: H
Digraph on 6 vertices
sage: H.vertices()
[0, 1, 2, 3, 4, 5]
```

slabbe.graph.**digraph_move_label_to_edge**(*G*, *label_function=None*, *loops=True*, *multiedges=False*)

Return a digraph with labels moved from the arrival vertices to corresponding edges.

INPUT:

- `G` – graph, whose vertices are tuples of the form (vertex, label)
- `label_function` – function or None, a function to apply to each label
- `loops` – bool (default: True)
- `multiedges` – bool (default: False)

EXAMPLES:

```
sage: G = DiGraph()
sage: G.add_edges([((i, None), ((i+1)%10, 'plusone')) for i in range(10)])
sage: G.add_edges([((i, None), ((i+2)%10, 'plustwo')) for i in range(10)])
sage: G
Digraph on 30 vertices
sage: from slabbe.graph import digraph_move_label_to_edge
sage: digraph_move_label_to_edge(G)
Looped digraph on 10 vertices
```

Using a function to modify the labels:

```
sage: f = lambda label:"A"+label
sage: GG = digraph_move_label_to_edge(G, label_function=f)
sage: GG
Looped digraph on 10 vertices
sage: GG.edges()[0]
(0, 1, 'Aplusone')
```

slabbe.graph.**induced_subgraph**(*G*, *filter*)

Return the induced subdigraph of a digraph keeping only vertices that are map to `True` by the filter.

INPUT:

- `G` – graph
- `filter` – function, a function from vertices to boolean

EXAMPLES:

```
sage: from slabbe.graph import induced_subgraph
sage: G = DiGraph()
```

(continues on next page)

(continued from previous page)

```
sage: G.add_edges([((i, None), ((i+1)%10, 'plusone')) for i in range(10)])
sage: G.add_edges([((i, None), ((i+2)%10, 'plustwo')) for i in range(10)])
sage: G
Digraph on 30 vertices
sage: GG = induced_subgraph(G, lambda v: v[0]%2 == 0)
sage: GG
Digraph on 15 vertices
sage: GG.edges()[0]
((0, None), (2, 'plustwo'), None)
```

`slabbe.graph.`**`merge_multiedges`**(*G*, *label_function=<type 'tuple'>*)

Return the (di)graph where multiedges are merged into one.

INPUT:

- `G` – graph
- `label_function` – function (default:`tuple`), a function to apply to each list of labels

OUTPUT:

(looped) (di)graph

EXAMPLES:

A digraph:

```
sage: from slabbe.graph import merge_multiedges
sage: G = DiGraph(multiedges=True)
sage: G.add_edge(0,1,'one')
sage: G.add_edge(0,1,'two')
sage: G.add_edge(0,1,'alpha')
sage: GG = merge_multiedges(G)
sage: GG
Digraph on 2 vertices
sage: GG.edges()
[(0, 1, ('alpha', 'one', 'two'))]
```

A graph:

```
sage: G = Graph(multiedges=True)
sage: G.add_edge(0,1,'one')
sage: G.add_edge(0,1,'two')
sage: G.add_edge(0,1,'alpha')
sage: GG = merge_multiedges(G)
sage: GG
Graph on 2 vertices
sage: GG.edges()
[(0, 1, ('alpha', 'one', 'two'))]
```

Using `label_function`:

```
sage: fn = lambda L: LatexExpr(','.join(map(str, L)))
sage: GG = merge_multiedges(G, label_function=fn)
sage: GG.edges()
[(0, 1, alpha,one,two)]
```

`slabbe.graph.`**`projection_graph`**(*G*, *proj_fn*, *filename=None*, *verbose=False*)

Return the image of a graph under a function on vertices.

INPUT:

- `G` – graph
- `proj_fn` – function

- `filename` – integer (default:`None`), save the graph to this pdf filename if filename is not None
- `verbose` – bool (default:`False`), print a table of data about the projection

EXAMPLES:

```
sage: from slabbe.graph import projection_graph
sage: g = graphs.PetersenGraph()
sage: g.vertices()
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
sage: f = lambda i: i % 5
sage: projection_graph(g, f)
Looped multi-digraph on 5 vertices
```

With verbose information:

```
sage: projection_graph(g, lambda i:i%4, verbose=True)
  Number of vertices   Projected vertices
+--------------------+--------------------+
  2                    3
  2                    2
  3                    1
  3                    0
Looped multi-digraph on 4 vertices
```

3.6 Partial injections

Random partial injections and Stallings graphs

EXAMPLES:

```
sage: from slabbe import number_of_partial_injection
sage: number_of_partial_injection(10)
[1,
100,
4050,
86400,
1058400,
7620480,
31752000,
72576000,
81648000,
36288000,
3628800]
```

Random partial injections on `[0, 1, ..., 6]`:

```
sage: from slabbe import random_partial_injection
sage: random_partial_injection(7)
[None, None, 1, 3, None, 0, None]
sage: random_partial_injection(7)
[5, 1, 0, 3, None, 4, None]
```

Random Stallings graph on `[0, 1, ..., 19]` over 2 letters:

```
sage: from slabbe import random_cyclically_reduced_stallings_graph
sage: G,_,_ = random_cyclically_reduced_stallings_graph(20, 2)
sage: G
Looped multi-digraph on 20 vertices
```

Visualisation of the graph:

```
sage: from slabbe import TikzPicture
sage: tikz = TikzPicture.from_graph(G)
sage: path_to_file = tikz.pdf()    # not tested
```

REFERENCES:

- Bassino, Frédérique; Nicaud, Cyril; Weil, Pascal Random generation of finitely generated subgroups of a free group. Internat. J. Algebra Comput. 18 (2008), no. 2, 375–405.

`slabbe.partial_injection.`**`number_of_partial_injection`**(*n*, *algorithm='binomial'*)

Return the number of partial injections on an set of n elements defined on a subset of k elements for each k in $0, 1, ..., n$.

INPUT:

- `n` – integer
- `algorithm` – string (default: `'binomial'`), `'binomial'` or `'recursive'`. When n>50, the binomial coefficient approach is faster (linear time vs quadratic time).

OUTPUT:

list

Note: The recursive code of this function was originally written by Vincent Delecroix (Nov 30, 2017) the day after a discussion with Pascal Weil and me at LaBRI.

EXAMPLES:

```
sage: from slabbe import number_of_partial_injection
sage: number_of_partial_injection(0)
[1]
sage: number_of_partial_injection(1)
[1, 1]
sage: number_of_partial_injection(2)
[1, 4, 2]
sage: number_of_partial_injection(3)
[1, 9, 18, 6]
sage: number_of_partial_injection(4)
[1, 16, 72, 96, 24]
sage: number_of_partial_injection(5)
[1, 25, 200, 600, 600, 120]
sage: number_of_partial_injection(6)
[1, 36, 450, 2400, 5400, 4320, 720]
sage: number_of_partial_injection(7)
[1, 49, 882, 7350, 29400, 52920, 35280, 5040]
sage: number_of_partial_injection(8)
[1, 64, 1568, 18816, 117600, 376320, 564480, 322560, 40320]
```

TESTS:

```
sage: number_of_partial_injection(8, algorithm='recursive')
[1, 64, 1568, 18816, 117600, 376320, 564480, 322560, 40320]
```

REFERENCE:

https://oeis.org/A144084

`slabbe.partial_injection.`**`random_cyclically_reduced_stallings_graph`**(*n*, *r=2*, *verbose=False*, *merge=False*)

Return a uniformly chosen Stallings graph of n vertices over r letters.

INPUT:

- `n` – integer, size of graph
- `r` – integer (default: `2`), number of generators of the free group
- `verbose` – bool (default: `False`)

Note: The probability that G is connected is 1 - 2^r / n^(r-1) + o(1/n^(r-1)) which is approx. 1

OUTPUT:

digraph, integer, integer

EXAMPLES:

```
sage: from slabbe import random_cyclically_reduced_stallings_graph
sage: G,_,_ = random_cyclically_reduced_stallings_graph(20, 2)
sage: G
Looped multi-digraph on 20 vertices
```

```
sage: random_cyclically_reduced_stallings_graph(20, 5)[0]
Looped multi-digraph on 20 vertices
```

With verbose output:

```
sage: G = random_cyclically_reduced_stallings_graph(20, 2, verbose=True)   # random
rejecting because graph is not connected
rejecting because graph has a vertex of degree <=1
rejecting because graph has a vertex of degree <=1
rejecting because graph has a vertex of degree <=1
rejecting because graph has a vertex of degree <=1
```

For displaying purposes, the following merges the multiedges automatically:

```
sage: G,_,_ = random_cyclically_reduced_stallings_graph(20, 2)
sage: from slabbe import TikzPicture
sage: tikz = TikzPicture.from_graph(G)
sage: _ = tikz.pdf(view=False)
```

AUTHORS:

- Sébastien Labbé and Pascal Weil, Dec 14, 2017, Sage Thursdays at LaBRI

`slabbe.partial_injection.`**`random_partial_injection`**`(n)`

Return a uniformly chosen random partial injection on 0, 1, . . . , n-1.

INPUT:

- `n` – integer

OUTPUT:

list

EXAMPLES:

```
sage: from slabbe import random_partial_injection
sage: random_partial_injection(10)
[3, 5, 2, None, 1, None, 0, 8, 7, 6]
sage: random_partial_injection(10)
[1, 7, 4, 8, 3, 5, 9, None, 6, None]
sage: random_partial_injection(10)
[5, 6, 8, None, 7, 4, 0, 9, None, None]
```

TODO:

```
Adapt the code once this is merged:

https://trac.sagemath.org/ticket/24416
```

AUTHORS:

- Sébastien Labbé and Vincent Delecroix, Nov 30, 2017, Sage Thursdays at LaBRI

`slabbe.partial_injection.`**`reject_statistics`**(*n*, *r=2*, *sample_size=50*, *verbose=False*)

Return return reject statistics when randomly chosing Stallings graph of n vertices over r letters.

INPUT:

- `n` – integer, size of graph
- `r` – integer (default: `2`), number of generators of the free group
- `n` – integer (default: `50`), size of sample
- `verbose` – bool (default: `False`)

OUTPUT:

histogram

EXAMPLES:

```
sage: from slabbe.partial_injection import reject_statistics
sage: h = reject_statistics(50, verbose=True)   # random
not connected: Counter({0: 48, 1: 2})
has degree 1: Counter({0: 27, 1: 18, 2: 3, 3: 1, 4: 1})
sage: h.save('h_50.png', title='size of graph=50') # not tested
```

```
sage: h = reject_statistics(100, verbose=True)  # random
not connected: Counter({0: 48, 1: 2})
has degree 1: Counter({0: 41, 1: 8, 2: 1})
sage: h.save('h_100.png', title='size of graph=100') # not tested
```

```
sage: h = reject_statistics(500, verbose=True)       # not tested (30s)
not connected: Counter({2: 5, 4: 5, 5: 5, 0: 4, 1: 4, 8: 4, 3: 3, 16:
    3, 6: 2, 11: 2, 15: 2, 18: 2, 23: 2, 7: 1, 10: 1, 44: 1, 13: 1, 49:
    1, 19: 1, 21: 1})
has degree 1: Counter({0: 14, 1: 9, 3: 8, 2: 5, 4: 3, 5: 3, 6: 2, 9: 2,
    7: 1, 8: 1, 13: 1, 15: 1})
sage: h.save('h_500.png', title='size of graph=500') # not tested
```

```
sage: h = reject_statistics(1000, verbose=True)  # not tested (2min30s)
not connected: Counter({8: 4, 26: 3, 3: 2, 7: 2, 9: 2, 10: 2, 14: 2,
15: 2, 17: 2, 18: 2, 27: 2, 40: 2, 59: 2, 0: 1, 1: 1, 4: 1, 5: 1, 11:
1, 13: 1, 19: 1, 20: 1, 21: 1, 22: 1, 28: 1, 44: 1, 48: 1, 51: 1, 52:
1, 53: 1, 58: 1, 63: 1, 66: 1, 75: 1, 121: 1})
has degree 1: Counter({2: 9, 0: 7, 1: 6, 4: 6, 3: 4, 5: 4, 7: 4, 8: 2,
    6: 1, 9: 1, 11: 1, 12: 1, 13: 1, 15: 1, 17: 1, 26: 1})
sage: h.save('h_1000.png', title='size of graph=1000') # not tested
```

CHAPTER

FOUR

DYNAMICAL SYSTEMS

4.1 Matrix Cocycles

Matrix cocyles

EXAMPLES:

The 1-cylinders of ARP transformation given as matrices:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.ARP()
sage: zip(*ARP.n_cylinders_iterator(1))
[(word: 1,
  word: 2,
  word: 3,
  word: 123,
  word: 132,
  word: 213,
  word: 231,
  word: 312,
  word: 321),
 (
[1 1 1]  [1 0 0]  [1 0 0]  [1 0 1]  [1 1 0]  [1 1 1]  [2 1 1]  [1 1 1]
[0 1 0]  [1 1 1]  [0 1 0]  [1 1 1]  [1 2 1]  [0 1 1]  [1 1 0]  [1 2 1]
[0 0 1], [0 0 1], [1 1 1], [1 1 2], [1 1 1], [1 1 2], [1 1 1], [0 1 1],

[2 1 1]
[1 1 1]
[1 0 1]
)]
```

Ces calculs illustrent le bounded distorsion de ratio=4 pour ARP multiplicatif (2 avril 2014):

```
sage: T = cocycles.Sorted_ARPMulti(2)
sage: T.distorsion_max(1, p=oo)
5
sage: T.distorsion_max(2, p=oo)
7
sage: T.distorsion_max(3, p=oo)
22/3
sage: T.distorsion_max(4, p=oo)    # long time (4s)
62/17
```

```
sage: T = cocycles.Sorted_ARPMulti(3)
sage: T.distorsion_max(1, p=oo)
7
sage: T.distorsion_max(2, p=oo)
9
sage: T.distorsion_max(3, p=oo)
```

(continues on next page)

(continued from previous page)

```
19/2
sage: T.distorsion_max(4, p=oo)  # long time (47s)
161/43
```

class slabbe.matrix_cocycle.**MatrixCocycle**(*gens*, *cone=None*, *language=None*)

Bases: `object`

Matrix cocycle

INPUT:

- `gens` – list, tuple or dict; the matrices. Keys 0,. . . ,n-1 are used for list and tuple.
- `cone` – dict or matrix or None (default: None); the cone for each matrix generators. If it is a matrix, then it serves as the cone for all matrices. The cone is defined by the columns of the matrix. If None, then the cone is the identity matrix.
- `language` – regular language or None (default: None); if None, the language is the full shift.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import MatrixCocycle
sage: B1 = matrix(3, [1,0,0, 0,1,0, 0,1,1])
sage: B2 = matrix(3, [1,0,0, 0,0,1, 0,1,1])
sage: B3 = matrix(3, [0,1,0, 0,0,1, 1,0,1])
sage: gens = {'1':B1, '2':B2, '3':B3}
sage: cone = matrix(3, [1,1,1,0,1,1,0,0,1])
sage: MatrixCocycle(gens, cone)
Cocycle with 3 gens over Language of finite words over alphabet ['1', '2', '3']
```

cone(*key*)

cone_dict()

distorsion_argmax(*n*, *p=1*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.Sorted_ARP()
sage: ARP.distorsion_argmax(1)
(
          [1 0 0]
          [1 1 0]
word: A1, [3 2 1]
)
```

distorsion_max(*n*, *p=1*)

EXAMPLES:

Non borné:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: T = cocycles.Sorted_ARP()
sage: T.distorsion_max(1, p=oo)
1
sage: T.distorsion_max(2, p=oo)
3
sage: T.distorsion_max(3, p=oo)
5
sage: T.distorsion_max(4, p=oo)
7
```

gens()

gens_inverses()

Return a dictionary of the inverses of the generators.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: coc = cocycles.Brun()
sage: coc.gens_inverses().keys()
[321, 132, 231, 213, 312, 123]
sage: coc.gens_inverses().values()
[
[ 1 -1  0]  [ 1  0  0]  [ 1  0 -1]  [ 1  0  0]  [ 1  0  0]  [ 1  0  0]
[ 0  1  0]  [ 0  1 -1]  [ 0  1  0]  [ 0  1  0]  [-1  1  0]  [ 0  1  0]
[ 0  0  1], [ 0  0  1], [ 0  0  1], [-1  0  1], [ 0  0  1], [ 0 -1  1]
]
```

If possible, the ring is the Integer ring:

```
sage: coc = cocycles.Reverse()
sage: coc.gens_inverses().values()
[
[ 1 -1 -1]  [ 1  0  0]  [ 1  0  0]  [-1/2  1/2  1/2]
[ 0  1  0]  [-1  1 -1]  [ 0  1  0]  [ 1/2 -1/2  1/2]
[ 0  0  1], [ 0  0  1], [-1 -1  1], [ 1/2  1/2 -1/2]
]
sage: [m.parent() for m in _]
[Full MatrixSpace of 3 by 3 dense matrices over Integer Ring,
 Full MatrixSpace of 3 by 3 dense matrices over Integer Ring,
 Full MatrixSpace of 3 by 3 dense matrices over Integer Ring,
 Full MatrixSpace of 3 by 3 dense matrices over Rational Field]
```

identity_matrix()

EXAMPLES:

```
sage: class Foo:
....:     def __init__(self, x):
....:         self._x = x
....:     @cached_method
....:     def f(self):
....:         return self._x^2
sage: a = Foo(2)
sage: print(a.f.cache)
None
sage: a.f()
4
sage: a.f.cache
4
```

is_pisot(*w*)

language()

n_cylinders_edges(*n*)

Return the set of edges of the n-cylinders.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.ARP()
sage: ARP.n_cylinders_edges(1)
{frozenset({(1, 1, 0), (1, 1, 1)}),
 frozenset({(0, 1, 0), (1, 1, 0)}),
 frozenset({(1, 1, 1), (2, 1, 1)}),
 frozenset({(0, 0, 1), (1, 0, 1)}),
 frozenset({(0, 1, 0), (0, 1, 1)}),
```

(continues on next page)

(continued from previous page)

```
 frozenset({(0, 1, 1), (1, 0, 1)}),
 frozenset({(1, 0, 0), (1, 1, 0)}),
 frozenset({(1, 1, 0), (2, 1, 1)}),
 frozenset({(1, 0, 1), (1, 1, 2)}),
 frozenset({(1, 1, 0), (1, 2, 1)}),
 frozenset({(1, 0, 1), (2, 1, 1)}),
 frozenset({(0, 0, 1), (0, 1, 1)}),
 frozenset({(1, 0, 1), (1, 1, 1)}),
 frozenset({(0, 1, 1), (1, 2, 1)}),
 frozenset({(0, 1, 1), (1, 1, 2)}),
 frozenset({(1, 0, 0), (1, 0, 1)}),
 frozenset({(1, 1, 1), (1, 2, 1)}),
 frozenset({(1, 0, 1), (1, 1, 0)}),
 frozenset({(0, 1, 1), (1, 1, 1)}),
 frozenset({(0, 1, 1), (1, 1, 0)}),
 frozenset({(1, 1, 1), (1, 1, 2)})}
```

n_cylinders_iterator(*n*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: C = cocycles.ARP()
sage: it = C.n_cylinders_iterator(1)
sage: for w,cyl in it: print("{}\n{}".format(w,cyl))
1
[1 1 1]
[0 1 0]
[0 0 1]
2
[1 0 0]
[1 1 1]
[0 0 1]
3
[1 0 0]
[0 1 0]
[1 1 1]
123
[1 0 1]
[1 1 1]
[1 1 2]
132
[1 1 0]
[1 2 1]
[1 1 1]
213
[1 1 1]
[0 1 1]
[1 1 2]
231
[2 1 1]
[1 1 0]
[1 1 1]
312
[1 1 1]
[1 2 1]
[0 1 1]
321
[2 1 1]
[1 1 1]
[1 0 1]
```

n_matrices_distorsion_iterator(*n*, *p=1*)

Return the the distorsion of the n-cylinders.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: T = cocycles.Sorted_ARP()
sage: it =T.n_matrices_distorsion_iterator(1)
sage: list(it)
[(word: A1, 2),
 (word: A2, 2),
 (word: A3, 2),
 (word: P1, 3),
 (word: P2, 3),
 (word: P3, 3)]
```

n_matrices_eigenvalues_iterator(*n*)

Return the eigenvalues of the matrices of level n.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.ARP()
sage: list(ARP.n_matrices_eigenvalues_iterator(1))
[(word: 1, [1, 1, 1]),
 (word: 2, [1, 1, 1]),
 (word: 3, [1, 1, 1]),
 (word: 123, [1, 1, 1]),
 (word: 132, [1, 1, 1]),
 (word: 213, [1, 1, 1]),
 (word: 231, [1, 1, 1]),
 (word: 312, [1, 1, 1]),
 (word: 321, [1, 1, 1])]
```

```
sage: B = cocycles.Sorted_Brun()
sage: list(B.n_matrices_eigenvalues_iterator(1))
[(word: 1, [1, 1, 1]),
 (word: 2, [1, -0.618033988749895?, 1.618033988749895?]),
 (word: 3, [1.465571231876768?,
           -0.2327856159383841? - 0.7925519925154479?*I,
           -0.2327856159383841? + 0.7925519925154479?*I])]
```

n_matrices_eigenvectors(*n*, *verbose=False*)

Return the left and right eigenvectors of the matrices of level n.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: C = cocycles.ARP()
sage: C.n_matrices_eigenvectors(1)
[(word: 1, (1.0, 0.0, 0.0), (0.0, 0.0, 1.0)),
 (word: 2, (0.0, 1.0, 0.0), (1.0, 0.0, 0.0)),
 (word: 3, (0.0, 0.0, 1.0), (1.0, 0.0, 0.0)),
 (word: 123, (0.0, 0.0, 1.0), (1.0, 0.0, 0.0)),
 (word: 132, (0.0, 1.0, 0.0), (1.0, 0.0, 0.0)),
 (word: 213, (0.0, 0.0, 1.0), (0.0, 1.0, 0.0)),
 (word: 231, (1.0, 0.0, 0.0), (0.0, 1.0, 0.0)),
 (word: 312, (0.0, 1.0, 0.0), (0.0, 0.0, 1.0)),
 (word: 321, (1.0, 0.0, 0.0), (0.0, 0.0, 1.0))]
```

n_matrices_iterator(*n*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.Sorted_ARP()
sage: A,B = zip(*list(ARP.n_matrices_iterator(1)))
sage: A
(word: A1, word: A2, word: A3, word: P1, word: P2, word: P3)
sage: B
```

(continues on next page)

(continued from previous page)

```
(
[1 0 0]  [1 0 0]  [0 1 0]  [0 1 0]  [0 0 1]  [0 0 1]
[0 1 0]  [0 0 1]  [0 0 1]  [0 1 1]  [1 0 1]  [0 1 1]
[1 1 1], [1 1 1], [1 1 1], [1 1 1], [1 1 1], [1 1 1]
)
```

n_matrices_non_pisot(*n*, *verbose=False*)

Return the list of non pisot matrices (as list of indices of base matrices).

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.Sorted_ARP()
sage: ARP.n_matrices_non_pisot(1)
[word: A1, word: A2]
sage: ARP.n_matrices_non_pisot(2)   # long time (1s)
[word: A1,A1, word: A1,A2, word: A2,A1, word: A2,A2]
sage: ARP.n_matrices_non_pisot(3)   # long time (11s)
[word: A1,A1,A1,
 word: A1,A1,A2,
 word: A1,A2,A1,
 word: A1,A2,A2,
 word: A2,A1,A1,
 word: A2,A1,A2,
 word: A2,A2,A1,
 word: A2,A2,A2]
sage: len(ARP.n_matrices_non_pisot(4))  # long time
16
```

```
sage: from slabbe.matrix_cocycle import cocycles
sage: B = cocycles.Sorted_Brun()
sage: B.n_matrices_non_pisot(2)
[word: 11, word: 12, word: 21, word: 22]
sage: B.n_matrices_non_pisot(3)
[word: 111,
 word: 112,
 word: 121,
 word: 122,
 word: 211,
 word: 212,
 word: 221,
 word: 222]
```

n_matrices_pinching_iterator(*n*)

Return the pinching matrices of level n.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.ARP()
sage: list(ARP.n_matrices_pinching_iterator(0))
[]
sage: list(ARP.n_matrices_pinching_iterator(1))
[]
sage: list(ARP.n_matrices_pinching_iterator(2))
[]
sage: L = list(ARP.n_matrices_pinching_iterator(3))
sage: L[0]
(
               [4 5 2]
               [2 3 1]
word: 1,2,213, [1 1 1]
)
```

n_matrices_semi_norm_iterator(*n*, *p=2*)

EXAMPLES:

For the 1-norm, all matrices contracts the hyperplane:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: C = cocycles.ARP()
sage: it = C.n_matrices_semi_norm_iterator(1, p=1)
sage: for _ in range(5): print(next(it)) # tolerance 0.0001
(word: 1, 1.0, False)
(word: 2, 1.0, False)
(word: 3, 1.0, False)
(word: 123, 0.9999885582839877, False)
(word: 132, 0.9999854006354785, False)
```

For the 2-norm, AR matrices do not contract:

```
sage: it = C.n_matrices_semi_norm_iterator(1, p=2)
sage: for w,s,b in it: print(w,s,b)  # long time (6s)
A1 1.30656296488 False
A2 1.30656296486 False
A3 1.30656296475 False
P12 0.99999999996 False
P13 0.999999999967 False
P21 0.999999999967 False
P23 0.999999999997 False
P31 0.999999999769 False
P32 0.999999999839 False
```

When, the 1-norm is < 1, the product is pisot:

```
sage: it = C.n_matrices_semi_norm_iterator(2, p=1)
sage: for w,s,b in it: print(w,s,b)  # long time
A1,A1 1.0 False
A1,A2 1.0 False
A1,A3 1.0 False
A1,P12 0.999998922557 False
A1,P13 0.999997464905 False
A1,P21 0.999993244882 False
A1,P23 0.999999150973 True
A1,P31 0.999994030522 False
A1,P32 0.999998046513 True
A2,A1 1.0 False
A2,A2 1.0 False
A2,A3 1.0 False
A2,P12 0.99999375291 False
A2,P13 0.999995591588 True
...
P31,A3 0.999988326888 False
P31,P12 0.749998931902 True
P31,P23 0.799999157344 True
P31,P32 0.749993104833 True
P32,A1 0.999997170005 True
P32,A3 0.99999420509 False
P32,P13 0.666665046248 True
P32,P21 0.666665629351 True
P32,P31 0.666664488371 True
```

n_words_iterator(*n*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.Sorted_ARP()
sage: list(ARP.n_words_iterator(1))
[word: A1, word: A2, word: A3, word: P1, word: P2, word: P3]
```

non_pisot_automaton(*n*)
EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: C = cocycles.ARP()
sage: A = C.non_pisot_automaton(2)
sage: A
Automaton with 2 states
sage: A.graph().plot(edge_labels=True)   # not tested
```

plot_n_cylinders(*n*, *labels=True*)
EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: C = cocycles.Sorted_ARP()
sage: G = C.plot_n_cylinders(3)
```

plot_n_matrices_eigenvectors(*n*, *side='right'*, *color_index=0*, *draw_line=False*)
INPUT:

- `n` – integer, length
- `side` – `'left'` or `'right'`, drawing left or right eigenvectors
- `color_index` – 0 for first letter, -1 for last letter
- `draw_line` – boolean

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.ARP()
sage: G = ARP.plot_n_matrices_eigenvectors(2)
```

plot_pisot_conjugates(*n*)
EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: B = cocycles.Sorted_Brun()
sage: G = B.plot_pisot_conjugates(5)   # long time (8s)
```

Image envoyee a Timo (6 mai 2014):

```
sage: G = sum(B.plot_pisot_conjugates(i) for i in [1..6])  #not tested
```

tikz_n_cylinders(*n*, *labels=None*, *scale=1*)
INPUT:

- `n` – integer, for the nth-cylinders
- `labels` – None, True or False (default: None), if None, it takes value True if n is 1.
- `scale` – real (default: 1), scale value for tikzpicture

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: ARP = cocycles.ARP()
sage: t = ARP.tikz_n_cylinders(1, labels=True, scale=4)
sage: t
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\begin{document}
\begin{tikzpicture}
```

(continues on next page)

(continued from previous page)

```
[scale=4]
\draw (0.0000, -0.5000) -- (0.0000, 0.0000);
\draw (0.0000, -0.5000) -- (0.8660, -0.5000);
\draw (0.0000, 0.0000) -- (-0.2165, -0.1250);
...
... 23 lines not printed (1317 characters in total) ...
...
\node at (-0.1443, 0.1667) {$213$};
\node at (-0.2165, 0.0417) {$231$};
\node at (0.0722, -0.2083) {$312$};
\node at (-0.0722, -0.2083) {$321$};
\end{tikzpicture}
\end{document}
```

```
sage: from sage.misc.temporary_file import tmp_filename
sage: filename = tmp_filename('temp','.pdf')
sage: _ = t.pdf(filename)
```

word_to_matrix(*w*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: C = cocycles.Sorted_ARP()
sage: C.word_to_matrix(Word())
[1 0 0]
[0 1 0]
[0 0 1]
```

class slabbe.matrix_cocycle.**MatrixCocycleGenerator**

Bases: object

ARP()

ArnouxRauzy()

Brun()

Cassaigne()

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: c = cocycles.Cassaigne()
sage: list(m for (w,m) in c.n_cylinders_iterator(2))
[
[1 1 1]  [1 1 0]  [0 0 1]  [1 0 0]
[0 1 0]  [0 1 1]  [1 1 0]  [0 1 0]
[0 0 1], [1 0 0], [0 1 1], [1 1 1]
]
```

Cassaigne_accelerated(*order=3*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cocycles
sage: c = cocycles.Cassaigne_accelerated(order=3)
sage: c
Cocycle with 6 gens over Language of finite words over
alphabet ['11', '121', '12^{2}1', '212', '21^{2}2', '22']
```

FullySubtractive()

Poincare()

Reverse()

Selmer()

Sorted_ARP()

Sorted_ARPMulti(*order=3*)

Sorted_Brun()

slabbe.matrix_cocycle.**arp_polyhedron**(*d=3*)

Return the d-dimensional 1-cylinders of the ARP algorithm.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import arp_polyhedron
sage: A,P,L = arp_polyhedron(3)
sage: A.vertices_list()
[[0, 0, 0], [1/2, 1/2, 0], [1/2, 1/4, 1/4], [1, 0, 0]]
sage: P.vertices_list()
[[0, 0, 0], [1/2, 1/2, 0], [1/2, 1/4, 1/4], [1/3, 1/3, 1/3]]
```

```
sage: A,P,L = arp_polyhedron(4)
sage: A.vertices_list()
[[0, 0, 0, 0],
 [1/2, 1/2, 0, 0],
 [1/2, 1/6, 1/6, 1/6],
 [1/2, 1/4, 1/4, 0],
 [1, 0, 0, 0]]
sage: P.vertices_list()
[[0, 0, 0, 0],
 [1/2, 1/2, 0, 0],
 [1/2, 1/4, 1/4, 0],
 [1/2, 1/6, 1/6, 1/6],
 [1/4, 1/4, 1/4, 1/4],
 [1/3, 1/3, 1/3, 0]]
```

```
sage: A,P,L = arp_polyhedron(5)
sage: A.vertices_list()
[[0, 0, 0, 0, 0],
 [1/2, 1/2, 0, 0, 0],
 [1/2, 1/8, 1/8, 1/8, 1/8],
 [1/2, 1/6, 1/6, 1/6, 0],
 [1/2, 1/4, 1/4, 0, 0],
 [1, 0, 0, 0, 0]]
sage: P.vertices_list()
[[0, 0, 0, 0, 0],
 [1/2, 1/2, 0, 0, 0],
 [1/2, 1/6, 1/6, 1/6, 0],
 [1/2, 1/8, 1/8, 1/8, 1/8],
 [1/2, 1/4, 1/4, 0, 0],
 [1/3, 1/3, 1/3, 0, 0],
 [1/5, 1/5, 1/5, 1/5, 1/5],
 [1/4, 1/4, 1/4, 1/4, 0]]
```

slabbe.matrix_cocycle.**cassaigne_polyhedron**(*d=3*)

Return the d-dimensional 1-cylinders of the Cassaigne algorithm.

(of the dual!)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import cassaigne_polyhedron
sage: L,La,Lb = cassaigne_polyhedron(3)
sage: L.vertices_list()
[[0, 0, 0], [0, 1/2, 1/2], [1/3, 1/3, 1/3], [1/2, 1/2, 0]]
sage: La.vertices_list()
```

(continues on next page)

(continued from previous page)

```
[[0, 0, 0], [0, 1/2, 1/2], [1/3, 1/3, 1/3], [1/4, 1/2, 1/4]]
sage: Lb.vertices_list()
[[0, 0, 0], [1/3, 1/3, 1/3], [1/2, 1/2, 0], [1/4, 1/2, 1/4]]
```

```
sage: L,La,Lb = cassaigne_polyhedron(4)
sage: L.vertices_list()
[[0, 0, 0, 0],
 [0, 1/3, 1/3, 1/3],
 [1/3, 1/3, 1/3, 0],
 [1/4, 1/4, 1/4, 1/4],
 [1/5, 2/5, 1/5, 1/5],
 [1/5, 1/5, 2/5, 1/5]]
```

```
sage: L,La,Lb = cassaigne_polyhedron(5)
sage: L.vertices_list()
[[0, 0, 0, 0, 0],
 [0, 1/4, 1/4, 1/4, 1/4],
 [1/4, 1/4, 1/4, 1/4, 0],
 [1/6, 1/6, 1/3, 1/6, 1/6],
 [1/5, 1/5, 1/5, 1/5, 1/5],
 [1/6, 1/3, 1/6, 1/6, 1/6],
 [1/7, 2/7, 2/7, 1/7, 1/7],
 [1/7, 2/7, 1/7, 2/7, 1/7],
 [1/7, 1/7, 2/7, 2/7, 1/7],
 [1/6, 1/6, 1/6, 1/3, 1/6]]
```

slabbe.matrix_cocycle.**distorsion**(*M*, *p=1*)

1 Avril 2014. L'ancien ratio n'était pas le bon. Je n'utilisais pas les bonnes normes.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import distorsion
sage: M = matrix(3, (1,2,3,4,5,6,7,8,9))
sage: M
[1 2 3]
[4 5 6]
[7 8 9]
sage: distorsion(M)
3/2
sage: (3+6+9) / (1+4+7)
3/2
sage: distorsion(M, p=oo)
9/7
```

slabbe.matrix_cocycle.**is_pisot**(*m*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import is_pisot
sage: M = matrix(3, (1,2,3,4,5,6,7,8,9))
sage: is_pisot(M)
False
```

slabbe.matrix_cocycle.**perron_right_eigenvector**(*M*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import perron_right_eigenvector
sage: m = matrix(2,[-11,14,-26,29])
sage: perron_right_eigenvector(m)           # tolerance 0.00001
(15.0000000000000, (0.35, 0.6499999999999999))
```

slabbe.matrix_cocycle.**rounded_string_vector**(*v*, *digits=4*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import rounded_string_vector
sage: v = (-0.144337567297406, 0.166666666666667)
sage: rounded_string_vector(v)
'(-0.1443, 0.1667)'
sage: rounded_string_vector(v, digits=6)
'(-0.144338, 0.166667)'
```

slabbe.matrix_cocycle.**semi_norm_D**(*v*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import semi_norm_D
sage: semi_norm_D((1,2,3,-5))
8
```

slabbe.matrix_cocycle.**semi_norm_cone**(*M*, *cone*, *p=2*, *verbose=False*)

Return the semi norm on the hyperplane orthogonal to v where v lives in the cone.

EXAMPLES:

For Arnoux-Rauzy, only the 1-norm works:

```
sage: from slabbe.matrix_cocycle import semi_norm_cone
sage: A1 = matrix(3, [1,1,1, 0,1,0, 0,0,1])
sage: cone = A1
sage: semi_norm_cone(A1.transpose(), cone, p=1)    # tolerance 0.00001
0.9999999999999998
sage: semi_norm_cone(A1.transpose(), cone, p=oo)   # tolerance 0.0001
1.9999757223144654
sage: semi_norm_cone(A1.transpose(), cone, p=2)   # tolerance 0.00001
1.3065629648763757
```

For Poincaré, all norms work:

```
sage: P21 = matrix(3, [1,1,1, 0,1,1, 0,0,1])
sage: H21 = matrix(3, [1,0,0, 0,1,0, 1,0,1])
sage: cone = P21 * H21
sage: semi_norm_cone(P21.transpose(), cone, p=1)   # tolerance 0.00001
0.9999957276014074
sage: semi_norm_cone(P21.transpose(), cone, p=oo)   # tolerance 0.00001
1.0
sage: semi_norm_cone(P21.transpose(), cone, p=2)   # tolerance 0.00001
0.9999999999670175
```

For Poincaré on the whole cone, it works for some norms:

```
sage: P21 = matrix(3, [1,1,1, 0,1,1, 0,0,1])
sage: cone = P21
sage: semi_norm_cone(P21.transpose(), cone, p=1)   # tolerance 0.0001 # known bug
1.9999675644077723
sage: semi_norm_cone(P21.transpose(), cone, p=2)   # tolerance 0.00001
1.6180339887021953
sage: semi_norm_cone(P21.transpose(), cone, p=oo)   # tolerance 0.00001
1.0
```

For a product, all norms work:

```
sage: A1 = matrix(3, [1,1,1, 0,1,0, 0,0,1])
sage: P21 = matrix(3, [1,1,1, 0,1,1, 0,0,1])
sage: H21 = matrix(3, [1,0,0, 0,1,0, 1,0,1])
sage: M = A1 * P21
sage: cone = A1 * P21 * H21
sage: semi_norm_cone(M.transpose(), cone, p=1)   # tolerance 0.00001
0.999993244882415
```

(continues on next page)

(continued from previous page)

```
sage: semi_norm_cone(M.transpose(), cone, p=oo)   # tolerance 0.00001
0.9999935206958908
sage: semi_norm_cone(M.transpose(), cone, p=2)   # tolerance 0.00001
0.7529377601317161
```

```
sage: M = cone = matrix(3,[2,3,2, 2,2,1, 1,2,1])
sage: semi_norm_cone(M.T, cone, p='D')  # tolerance 0.00001
0.7499977852638109
```

`slabbe.matrix_cocycle.semi_norm_v`(*M*, *v*, *p=2*, *verbose=False*)

Return the semi norm on the hyperplane orthogonal to v.

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import semi_norm_v
sage: A1 = matrix(3, [1,-1,-1, 0,1,0, 0,0,1]).inverse()
sage: semi_norm_v(A1, vector( (1,1,1)))[0]      # tolerance 0.0001
0.9999999999890247
sage: semi_norm_v(A1, vector( (1,1,1)), p=1)[0]   # tolerance 0.0001
0.9999394820959548
sage: semi_norm_v(A1, vector( (1,1,1)), p=oo)[0]   # tolerance 0.0001
1.0
```

```
sage: m = matrix(3,[0,0,0, 1,0,1, 0,-1,0])
sage: semi_norm_v(m, vector((1,1,1)), p='D')[0]  # tolerance 0.0001
0.6666436827952827
```

4.2 Multidimensional Continued Fraction Algorithms

Multidimensional Continued Fraction Algorithms (Python code)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: algo = Brun()
```

Drawing the natural extension:

```
sage: fig = algo.natural_extension_plot(3000, norm_xyz=1, axis_off=True)
sage: fig
<Figure size 1200x200 with 4 Axes>
sage: fig.savefig('a.png')  # not tested
```

Drawing the invariant measure:

```
sage: fig = algo.invariant_measure_wireframe_plot(10^6, 50)
sage: fig
<Figure size 640x480 with 1 Axes>
sage: fig.savefig('a.png')  # not tested
```

Word with given frequencies:

```
sage: algo.s_adic_word((1,e,pi))
word: 1232323123233231232332312323123232312323...
```

Construction of the same s-adic word from the substitutions and the coding iterator:

```
sage: from itertools import repeat
sage: D = algo.substitutions()
sage: it = algo.coding_iterator((1,e,pi))
sage: words.s_adic(it, repeat(1), D)
word: 1232323123233323123233323123231232323312323...
```

AUTHORS:

- Sébastien Labbé, Externalize Python only functions (pip install takes now 33s instead of 51s), August 2016

slabbe.mult_cont_frac.**ARP**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ARP
sage: ARP()
Arnoux-Rauzy-Poincar\'e 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**ArnouxRauzy**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import ArnouxRauzy
sage: ArnouxRauzy()
ArnouxRauzy 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**Brun**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: Brun()
Brun 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**Cassaigne**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Cassaigne
sage: Cassaigne()
Cassaigne 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**FullySubtractive**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import FullySubtractive
sage: FullySubtractive()
Fully Subtractive 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**JacobiPerron**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import JacobiPerron
sage: JacobiPerron()
JacobiPerron 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**JacobiPerronAdditif**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import JacobiPerronAdditif
sage: JacobiPerronAdditif()
JacobiPerronAdditif 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**JacobiPerronAdditifv2**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import JacobiPerronAdditifv2
sage: JacobiPerronAdditifv2()
JacobiPerronAdditifv2 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**Poincare**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Poincare
sage: Poincare()
Poincar\'e 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**Reverse**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Reverse
sage: Reverse()
Reverse 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**Selmer**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Selmer
sage: Selmer()
Selmer 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**Sorted_ARMonteil**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_ARMonteil
sage: Sorted_ARMonteil()
Sorted_ARMonteil 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**Sorted_ARP**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_ARP
sage: Sorted_ARP()
Sorted_ARP 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**Sorted_ARPMulti**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_ARPMulti
sage: Sorted_ARPMulti()
Sorted_ARPMulti 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**Sorted_ARrevert**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_ARrevert
sage: Sorted_ARrevert()
Sorted_ARrevert 3-dimensional continued fraction algorithm
```

slabbe.mult_cont_frac.**Sorted_ARrevertMulti**()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_ARrevertMulti
sage: Sorted_ARrevertMulti()
Sorted_ARrevertMulti 3-dimensional continued fraction algorithm
```

`slabbe.mult_cont_frac.`**`Sorted_ArnouxRauzy`**`()`

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_ArnouxRauzy
sage: Sorted_ArnouxRauzy()
Sorted_ArnouxRauzy 3-dimensional continued fraction algorithm
```

`slabbe.mult_cont_frac.`**`Sorted_ArnouxRauzyMulti`**`()`

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_ArnouxRauzyMulti
sage: Sorted_ArnouxRauzyMulti()
Sorted_ArnouxRauzyMulti 3-dimensional continued fraction algorithm
```

`slabbe.mult_cont_frac.`**`Sorted_Brun`**`()`

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_Brun
sage: Sorted_Brun()
Sorted_Brun 3-dimensional continued fraction algorithm
```

`slabbe.mult_cont_frac.`**`Sorted_BrunMulti`**`()`

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_BrunMulti
sage: Sorted_BrunMulti()
Sorted_BrunMulti 3-dimensional continued fraction algorithm
```

`slabbe.mult_cont_frac.`**`Sorted_Delaunay`**`()`

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_Delaunay
sage: Sorted_Delaunay()
Sorted_Delaunay 3-dimensional continued fraction algorithm
```

`slabbe.mult_cont_frac.`**`Sorted_FullySubtractive`**`()`

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_FullySubtractive
sage: Sorted_FullySubtractive()
Sorted_FullySubtractive 3-dimensional continued fraction algorithm
```

`slabbe.mult_cont_frac.`**`Sorted_Poincare`**`()`

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_Poincare
sage: Sorted_Poincare()
Sorted_Poincare 3-dimensional continued fraction algorithm
```

`slabbe.mult_cont_frac.`**`Sorted_Selmer`**`()`

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Sorted_Selmer
sage: Sorted_Selmer()
Sorted_Selmer 3-dimensional continued fraction algorithm
```

Multidimensional Continued Fraction Algorithms (Cython code)

See also the Python code which provides more methods.

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: algo = Brun()
```

Orbit in the cone (with dual coordinates):

```
sage: algo.cone_orbit_list((10,23,15), 6)
[(((10.0, 8.0, 15.0), (1.0, 1.0, 2.0)), 132),
 (((10.0, 8.0, 5.0), (3.0, 1.0, 2.0)), 213),
 (((2.0, 8.0, 5.0), (3.0, 4.0, 2.0)), 321),
 (((2.0, 3.0, 5.0), (3.0, 4.0, 6.0)), 132),
 (((2.0, 3.0, 2.0), (3.0, 10.0, 6.0)), 123),
 (((2.0, 1.0, 2.0), (3.0, 10.0, 16.0)), 132)]
```

Orbit in the simplex:

```
sage: algo.simplex_orbit_list((10,23,15), 3)
[(0.30303030303030304,
  0.24242424242424246,
  0.45454545454545453,
  0.25,
  0.25,
  0.5,
  132),
 (0.43478260869565216,
  0.3478260869565218,
  0.21739130434782603,
  0.5,
  0.16666666666666666,
  0.3333333333333333,
  213),
 (0.13333333333333328,
  0.5333333333333334,
  0.3333333333333333,
  0.33333333333333337,
  0.4444444444444445,
  0.22222222222222224,
  321)]
```

BENCHMARKS:

With slabbe-0.2 or earlier, 68.6 ms on my machine. With slabbe-0.3.b1, 62.2 ms on my machine. With slabbe-0.3.b2, 28.6 ms on my machine. With slabbe-0.3.b2, 13.3 ms on priminfo in Liège:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: %time Brun().lyapunov_exponents(n_iterations=10^6)  # not tested
(0.3049429393152174, -0.1120652699014143, 1.367495867105725)
```

With slabbe-0.3.b1, 74ms on my machine. With slabbe-0.3.b2, 35ms on my machine. With slabbe-0.3.b2, 17ms on priminfo in Liège:

```
sage: from slabbe.mult_cont_frac_pyx import ARP
sage: %time ARP().lyapunov_exponents(n_iterations=10^6)  # not tested
(0.443493194984839, -0.17269097306340797, 1.3893881011394358)
```

With slabbe-0.2 or earlier, 3.71s at liafa, 4.58s on my machine. With slabbe-0.3.b1, 3.93s on my machine. With slabbe-0.3.b2, 1.93s on my machine. With slabbe-0.3.b2, 1.22s on priminfo in Liège:

```
sage: %time Brun().lyapunov_exponents(n_iterations=67000000) # not tested
(0.30456433843239084, -0.1121770192467067, 1.36831961293987303)
```

With slabbe-0.3.b1, 4.83 s on my machine: With slabbe-0.3.b2, 2.33 s on my machine: With slabbe-0.3.b2, 1.56 s on priminfo in Liège:

```
sage: %time ARP().lyapunov_exponents(n_iterations=67*10^6)   # not tested
(0.44296596371477626, -0.17222952278277034, 1.3888098339168744)
```

With slabbe-0.2 or earlier, 660 ms on my machine. With slabbe-0.3.b1, 640 ms on my machine (maybe this test could be made much faster without using list. . .). With slabbe-0.3.b2, 215 ms on priminfo in Liège:

```
sage: %time L = Brun().simplex_orbit_list(n_iterations=10^6)   # not tested
```

Question:

- Comment factoriser le code sans utiliser les yield?
- Comment faire un appel de fonction rapide (pour factoriser le code)

AUTHORS:

- Sébastien Labbé, Invariant measures, Lyapounov exponents and natural extensions for a dozen of algorithms, October 2013.
- Sébastien Labbé, Cleaning the code, Fall 2015
- Sébastien Labbé, Making use of PairPoint to prepare for higher dimension, Fall 2016

class slabbe.mult_cont_frac_pyx.ARP

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import ARP
sage: algo = ARP()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import ARP
sage: ARP().dual_substitutions()
{1: WordMorphism: 1->123, 2->2, 3->3,
 2: WordMorphism: 1->1, 2->231, 3->3,
 3: WordMorphism: 1->1, 2->2, 3->312,
 123: WordMorphism: 1->1, 2->21, 3->321,
 132: WordMorphism: 1->1, 2->231, 3->31,
 213: WordMorphism: 1->12, 2->2, 3->312,
 231: WordMorphism: 1->132, 2->2, 3->32,
 312: WordMorphism: 1->13, 2->213, 3->3,
 321: WordMorphism: 1->123, 2->23, 3->3}
```

name()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import ARP
sage: ARP().name()
"Arnoux-Rauzy-Poincar\\'e"
```

substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import ARP
sage: ARP().substitutions()
{1: WordMorphism: 1->1, 2->21, 3->31,
```

(continues on next page)

(continued from previous page)

```
 2: WordMorphism: 1->12, 2->2, 3->32,
 3: WordMorphism: 1->13, 2->23, 3->3,
 123: WordMorphism: 1->123, 2->23, 3->3,
 132: WordMorphism: 1->132, 2->2, 3->32,
 213: WordMorphism: 1->13, 2->213, 3->3,
 231: WordMorphism: 1->1, 2->231, 3->31,
 312: WordMorphism: 1->12, 2->2, 3->312,
 321: WordMorphism: 1->1, 2->21, 3->321}
```

class slabbe.mult_cont_frac_pyx.**ArnouxRauzy**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import ArnouxRauzy
sage: algo = ArnouxRauzy()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import ArnouxRauzy
sage: ArnouxRauzy().dual_substitutions()
{1: WordMorphism: 1->123, 2->2, 3->3,
 2: WordMorphism: 1->1, 2->231, 3->3,
 3: WordMorphism: 1->1, 2->2, 3->312}
```

substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import ArnouxRauzy
sage: ArnouxRauzy().substitutions()
{1: WordMorphism: 1->1, 2->21, 3->31,
 2: WordMorphism: 1->12, 2->2, 3->32,
 3: WordMorphism: 1->13, 2->23, 3->3}
```

class slabbe.mult_cont_frac_pyx.**Brun**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: algo = Brun()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: Brun().dual_substitutions()
{123: WordMorphism: 1->1, 2->2, 3->32,
 132: WordMorphism: 1->1, 2->23, 3->3,
 213: WordMorphism: 1->1, 2->2, 3->31,
 231: WordMorphism: 1->13, 2->2, 3->3,
 312: WordMorphism: 1->1, 2->21, 3->3,
 321: WordMorphism: 1->12, 2->2, 3->3}
```

substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: Brun().substitutions()
{123: WordMorphism: 1->1, 2->23, 3->3,
 132: WordMorphism: 1->1, 2->2, 3->32,
 213: WordMorphism: 1->13, 2->2, 3->3,
 231: WordMorphism: 1->1, 2->2, 3->31,
 312: WordMorphism: 1->12, 2->2, 3->3,
 321: WordMorphism: 1->1, 2->21, 3->3}
```

class slabbe.mult_cont_frac_pyx.**Cassaigne**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Cassaigne
sage: algo = Cassaigne()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Cassaigne
sage: Cassaigne().dual_substitutions()
{1: WordMorphism: 1->12, 2->3, 3->2,
 2: WordMorphism: 1->2, 2->1, 3->23}
```

substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Cassaigne
sage: Cassaigne().substitutions()
{1: WordMorphism: 1->1, 2->13, 3->2,
 2: WordMorphism: 1->2, 2->13, 3->3}
```

class slabbe.mult_cont_frac_pyx.**FullySubtractive**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import FullySubtractive
sage: algo = FullySubtractive()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import FullySubtractive
sage: FullySubtractive().dual_substitutions()
{1: WordMorphism: 1->1, 2->21, 3->31,
 2: WordMorphism: 1->12, 2->2, 3->32,
 3: WordMorphism: 1->13, 2->23, 3->3}
```

name()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import FullySubtractive
sage: FullySubtractive().name()
'Fully Subtractive'
```

substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import FullySubtractive
sage: FullySubtractive().substitutions()
{1: WordMorphism: 1->123, 2->2, 3->3,
 2: WordMorphism: 1->1, 2->231, 3->3,
 3: WordMorphism: 1->1, 2->2, 3->312}
```

class slabbe.mult_cont_frac_pyx.**JacobiPerron**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**JacobiPerronAdditif**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**JacobiPerronAdditifv2**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**MCFAlgorithm**

Bases: `object`

branches()

Returns the branches labels of the algorithm.

This method is an heuristic and should be implemented in the inherited classes.

EXAMPLES:

```
sage: import slabbe.mult_cont_frac_pyx as mcf
sage: mcf.Brun().branches()
{123, 132, 213, 231, 312, 321}
sage: mcf.ARP().branches()
{1, 2, 3, 123, 132, 213, 231, 312, 321}
```

class_name()

The name of the class.

Note: This might not be the same as the name of the algorithm.

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Reverse, Brun, ARP
sage: Reverse().class_name()
'Reverse'
sage: Brun().class_name()
'Brun'
sage: ARP().class_name()
'ARP'
```

coding_iterator()

INPUT:

- `start` – iterable of three real numbers

OUTPUT:

iterator

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import ARP
sage: it = ARP().coding_iterator((1,e,pi))
sage: [next(it) for _ in range(20)]
[123, 2, 1, 123, 1, 231, 3, 3, 3, 3, 123, 1, 1, 1, 231, 2, 321, 2, 3, 312]
```

```
sage: from slabbe.mult_cont_frac_pyx import Poincare
sage: algo = Poincare(4)
sage: it = algo.coding_iterator((1,e,pi,sqrt(2)))
sage: [next(it) for _ in range(10)]
[1423, 4312, 3241, 3412, 3142, 3214, 4312, 1342, 3412, 1342]
```

cone_orbit_iterator()

INPUT:

- `start` - initial vector (default: `None`), if None, then initial point is random

NOTE:

This iterator is 10x slower because of the yield statement. So avoid using this when writing fast code. Just copy paste the loop or use simplex_orbit_list or simplex_orbit_filtered_list method.

OUTPUT:

iterator of tuples (PairPoint, integer)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: it = Brun().cone_orbit_iterator((13,17,29))
sage: for _ in range(10): next(it)
(((13.0, 17.0, 12.0), (1.0, 2.0, 1.0)), 123)
(((13.0, 4.0, 12.0), (3.0, 2.0, 1.0)), 312)
(((1.0, 4.0, 12.0), (3.0, 2.0, 4.0)), 231)
(((1.0, 4.0, 8.0), (3.0, 6.0, 4.0)), 123)
(((1.0, 4.0, 4.0), (3.0, 10.0, 4.0)), 123)
(((1.0, 4.0, 0.0), (3.0, 14.0, 4.0)), 123)
(((1.0, 3.0, 0.0), (17.0, 14.0, 4.0)), 312)
(((1.0, 2.0, 0.0), (31.0, 14.0, 4.0)), 312)
(((1.0, 1.0, 0.0), (45.0, 14.0, 4.0)), 312)
(((1.0, 0.0, 0.0), (59.0, 14.0, 4.0)), 312)
```

cone_orbit_list()

INPUT:

- `start` - initial vector (default: `None`), if None, then initial point is random
- `n_iterations` - integer, number of iterations

OUTPUT:

list of tuples (PairPoint, integer)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: L = Brun().cone_orbit_list((10, 21, 37), 20)
sage: L[-1]
(((1.0, 0.0, 0.0), (68.0, 55.0, 658.0)), 231)
```

dual_substitutions()

This method must be implemented in the inherited classes.

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: Brun().dual_substitutions()
{123: WordMorphism: 1->1, 2->2, 3->32,
 132: WordMorphism: 1->1, 2->23, 3->3,
 213: WordMorphism: 1->1, 2->2, 3->31,
 231: WordMorphism: 1->13, 2->2, 3->3,
 312: WordMorphism: 1->1, 2->21, 3->3,
 321: WordMorphism: 1->12, 2->2, 3->3}
```

`image()`

Return the image of a vector in R^3 after n iterations.

INPUT:

- `start` - initial vector
- `n_iterations` - integer, number of iterations (default: 1)

OUTPUT:

tuple of three floats

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: Brun().image((10, 21, 37))
(10.0, 21.0, 16.0)
sage: Brun().image((10, 21, 37), 2)
(10.0, 5.0, 16.0)
sage: Brun().image((10, 21, 37), 3)
(10.0, 5.0, 6.0)
sage: Brun().image((10, 21, 37), 10)
(1.0, 1.0, 0.0)
```

`lyapunov_exponents()`

Return the lyapunov exponents (theta1, theta2, 1-theta2/theta1)

See also the module `slabbe.lyapunov` for parallel computations.

INPUT:

- `start` - initial vector (default: `None`), if None, then initial point is random
- `n_iterations` – integer
- `verbose` – bool (default: `False`)

OUTPUT:

tuple of the first two liapounov exponents and the uniform approximation exponent:

(theta1, theta2, 1-theta2/theta1)

Note: the code of this method was translated from C to cython. The C version is from Vincent Delecroix.

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: Brun().lyapunov_exponents(n_iterations=1000000)  # tol 0.02
(0.3049429393152174, -0.1120652699014143, 1.367495867105725)
```

```
sage: start = (0.2134134, 0.31618415, 0.414514985)
sage: Brun().lyapunov_exponents(start=start, n_iterations=10^6)  # tol 0.01
(0.3046809303742965, -0.1121152799778245, 1.3679760326322108)
```

`name()`

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Reverse, Brun, ARP
sage: Reverse().name()
'Reverse'
sage: Brun().name()
'Brun'
sage: ARP().name()
"Arnoux-Rauzy-Poincar\\'e"
```

`nsmall_entries_list()`

INPUT:

- `ratio` - real number, 0 < ratio < 1
- `start` - initial vector (default: `None`), if None, then initial point is random
- `n_iterations` – integer
- `p` – integer, p-norm

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Poincare
sage: algo = Poincare(4)
sage: algo.nsmall_entries_list(.1, (1,e,pi,sqrt(2)), n_iterations=20)
[0, 1, 1, 1, 1, 0, 0, 1, 0, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3]
```

```
sage: from slabbe.finite_word import run_length_encoding
sage: L = algo.nsmall_entries_list(.01, (1,e,pi,sqrt(2)), n_iterations=1000)
sage: run_length_encoding(L)
[(0, 1), (1, 1), (0, 7), (1, 1), (0, 3), (1, 2), (2, 1), (3, 984)]
```

`return_time_to_nsmall_entries()`

INPUT:

- `ratio` - real number, 0 < ratio < 1
- `n` - integer, number of small entries
- `start` - initial vector (default: `None`), if None, then initial point is random
- `p` – integer, p-norm

OUTPUT:

a tuple (integer, PairPoint)

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Poincare
sage: algo = Poincare(4)
sage: algo.return_time_to_nsmall_entries(.05, 0, (1,e,pi,sqrt(2)))
(3,
 ((0.31830988618379064, 0.41509782135371204,
   0.13474402056773493, 0.1318482718947624),
  (0.0, 0.0, 0.0, 0.0)))
```

```
sage: algo = Poincare(6)
sage: start = (1,e,pi,sqrt(2),sqrt(3),sqrt(5))
sage: algo.return_time_to_nsmall_entries(.05, 0, start)
(5,
 ((0.3183098861837907, 0.153493436015088, 0.134744020567735,
 0.1318482718947624, 0.1011707373432389, 0.16043364799538504),
 (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)))
```

`simplex_orbit_filtered_list()`

Return a list of the orbit filtered to fit into a rectangle.

INPUT:

- `start` - initial vector (default: `None`), if None, then initial point is random
- `n_iterations` - integer, number of iterations
- `norm_xyz` – integer (default: `1`), either `0` or `1`, the norm used for the orbit of points (x, y, z) of the algo
- `norm_uvw` – integer (default: `1`), either `0` or `1` or `'hypersurfac'`, the norm used for the orbit of dual coordinates (u, v, w).
- `xmin` - double
- `ymin` - double
- `umin` - double
- `vmin` - double
- `xmax` - double
- `ymax` - double
- `umax` - double
- `vmax` - double
- `ndvis` - integer, number of divisions

OUTPUT:

list

BENCHMARK:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: %time D = Brun().simplex_orbit_filtered_list(10^6) # not tested
CPU times: user 366 ms, sys: 203 ms, total: 568 ms
Wall time: 570 ms
```

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: start=(.414578,.571324,.65513)
sage: D = Brun().simplex_orbit_filtered_list(start, 3)
sage: D      # random
[(0.3049590483124023,
  -0.36889249928767137,
  -0.21650635094610976,
  -0.125,
  312,
  312),
 (0.08651831333735083,
  -0.31784823591841554,
  -0.34641016151377557,
  -0.2,
  312,
  312),
 (-0.41045591033143647,
  -0.20171750067080554,
  -0.4330127018922195,
  -0.25000000000000006,
  312,
  231)]
```

```
sage: Brun().simplex_orbit_filtered_list(n_iterations=3, norm_xyz=1,ndivs=1000)
Traceback (most recent call last):
...
ValueError: when ndivs is specified, you must provide a value
for xmin, xmax, ymin, ymax, umin, umax, vmin and vmax
```

```
sage: Brun().simplex_orbit_filtered_list(n_iterations=7,  # random
....:       norm_xyz=1, ndivs=100,
....:       xmin=-.866, xmax=.866, ymin=-.5, ymax=1.,
....:       umin=-.866, umax=.866, vmin=-.5, vmax=1.)
[(30, 47, 50, 50, 132, 213),
 (15, 83, 33, 66, 213, 231),
 (18, 80, 38, 44, 231, 231),
 (22, 75, 41, 33, 231, 231),
 (30, 68, 43, 26, 231, 231),
 (44, 53, 44, 22, 231, 213),
 (41, 78, 24, 56, 213, 321)]
```

simplex_orbit_iterator()

INPUT:

- start - initial vector (default: None), if None, then initial point is random
- norm_xyz – integer (default: 0), either 0 or 1, the norm used for the orbit of points (x, y, z) of the algo
- norm_uvw – integer (default: 1), either 0 or 1 or 'hypersurfac', the norm used for the orbit of dual coordinates (u, v, w).

NOTE:

This iterator is 10x slower because of the yield statement. So avoid using this when writing fast code. Just copy paste the loop or use simplex_orbit_list or simplex_orbit_filtered_list method.

OUTPUT:

iterator

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: it = Brun().simplex_orbit_iterator((.414578,.571324,.65513))
sage: for _ in range(4): next(it)
((0.7256442929056017, 1.0, 0.14668734378391243),
 (0.25, 0.5, 0.25),
 123)
((1.0, 0.37808566783572695, 0.20214772612150184),
 (0.5, 0.3333333333333333, 0.16666666666666666),
 312)
((1.0, 0.6079385025908344, 0.32504111204194974),
 (0.3333333333333333, 0.5555555555555555, 0.1111111111111111),
 321)
((0.6449032192209051, 1.0, 0.534661171576946),
 (0.25, 0.6666666666666666, 0.08333333333333333),
 321)
```

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: it = Brun().simplex_orbit_iterator((.414578,.571324,.65513), norm_xyz=1)
sage: for _ in range(4): next(it)
((0.3875618393056797, 0.5340934161472103, 0.07834474454711005),
 (0.25, 0.5, 0.25),
 123)
((0.6328179140018012, 0.23925938363378257, 0.12792270236441622),
 (0.5, 0.3333333333333333, 0.16666666666666666),
```

(continues on next page)

(continued from previous page)

```
 312)
((0.5173360300491189, 0.3145084914443481, 0.16815547850653312),
 (0.3333333333333333, 0.5555555555555555, 0.1111111111111111),
 321)
((0.2958862889959549, 0.45880727553726447, 0.24530643546678058),
 (0.25, 0.6666666666666666, 0.08333333333333333),
 321)
```

simplex_orbit_list()

INPUT:

- `start` - initial vector (default: `None`), if None, then initial point is random
- `n_iterations` - integer, number of iterations
- `norm_xyz` – integer (default: `1`), either `0` or `1`, the norm used for the orbit of points (x, y, z) of the algo
- `norm_uvw` – integer (default: `1`), either `0` or `1` or `'hypersurfac'`, the norm used for the orbit of dual coordinates (u, v, w).

OUTPUT:

list

Note: It could be 10 times faster because 10^6 iterations can be done in about 60ms on this machine. But for drawing images, it does not matter to be 10 times slower:

```
sage: %time L = Brun().simplex_orbit_list(10^6)   # not tested
CPU times: user 376 ms, sys: 267 ms, total: 643 ms
Wall time: 660 ms
```

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: L = Brun().simplex_orbit_list(n_iterations=10^5)
sage: L[-1]    # random
(0.7307002153148079,
 1.0,
 0.31588474491578816,
 0.29055326655584235,
 0.4690741038784866,
 0.24037262956567113,
 321)
```

substitutions()

This method must be implemented in the inherited classes.

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Brun
sage: Brun().substitutions()
{123: WordMorphism: 1->1, 2->23, 3->3,
 132: WordMorphism: 1->1, 2->2, 3->32,
 213: WordMorphism: 1->13, 2->2, 3->3,
 231: WordMorphism: 1->1, 2->2, 3->31,
 312: WordMorphism: 1->12, 2->2, 3->3,
 321: WordMorphism: 1->1, 2->21, 3->3}
```

class slabbe.mult_cont_frac_pyx.**PairPoint**

Bases: `object`

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import PairPoint
sage: PairPoint(3, (.2,.3,.4))
((0.2, 0.3, 0.4), (..., ..., ...))
sage: PairPoint(3, a=(.2,.3,.4))
((..., ..., ...), (0.2, 0.3, 0.4))
```

number_small_entries()

Returns the number of indices i such that x[i]/||x|| < ratio.

permutation()

http://stackoverflow.com/questions/17554242/how-to-obtain-the-index-permutation-after-the-sorting

OUTPUT:

int (the permutation, works well if self.dim < 10)

Permutation gets written to self.perm

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import PairPoint
sage: P = PairPoint(4, [.4, .2, .3, .1], [4,3,2,1])
sage: P.permutation()
4231
```

sort_x()

Sort array x independently of array a.

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import PairPoint
sage: P = PairPoint(4, [.4, .2, .3, .1], [4,3,2,1])
sage: P.sort_x()
sage: P
((0.1, 0.2, 0.3, 0.4), (4.0, 3.0, 2.0, 1.0))
```

to_dict()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import PairPoint
sage: PairPoint(3, [1,2,3], [4,5,6]).to_dict()
{'a': [4.0, 5.0, 6.0], 'x': [1.0, 2.0, 3.0]}
```

to_tuple()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import PairPoint
sage: PairPoint(3, [1,2,3], [4,5,6]).to_tuple()
((1.0, 2.0, 3.0), (4.0, 5.0, 6.0))
```

class slabbe.mult_cont_frac_pyx.**Poincare**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Poincare
sage: algo = Poincare()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Poincare
sage: Poincare().dual_substitutions()
{123: WordMorphism: 1->1, 2->21, 3->321,
 132: WordMorphism: 1->1, 2->231, 3->31,
 213: WordMorphism: 1->12, 2->2, 3->312,
 231: WordMorphism: 1->132, 2->2, 3->32,
 312: WordMorphism: 1->13, 2->213, 3->3,
 321: WordMorphism: 1->123, 2->23, 3->3}
```

name()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Poincare
sage: Poincare().name()
"Poincar\\'e"
```

substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Poincare
sage: Poincare().substitutions()
{123: WordMorphism: 1->123, 2->23, 3->3,
 132: WordMorphism: 1->132, 2->2, 3->32,
 213: WordMorphism: 1->13, 2->213, 3->3,
 231: WordMorphism: 1->1, 2->231, 3->31,
 312: WordMorphism: 1->12, 2->2, 3->312,
 321: WordMorphism: 1->1, 2->21, 3->321}
```

class slabbe.mult_cont_frac_pyx.**Reverse**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Reverse
sage: algo = Reverse()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Reverse
sage: Reverse().dual_substitutions()
{1: WordMorphism: 1->123, 2->2, 3->3,
 2: WordMorphism: 1->1, 2->231, 3->3,
 3: WordMorphism: 1->1, 2->2, 3->312,
 4: WordMorphism: 1->23, 2->13, 3->12}
```

substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Reverse
sage: Reverse().substitutions()
{1: WordMorphism: 1->1, 2->21, 3->31,
 2: WordMorphism: 1->12, 2->2, 3->32,
 3: WordMorphism: 1->13, 2->23, 3->3,
 4: WordMorphism: 1->23, 2->31, 3->12}
```

class slabbe.mult_cont_frac_pyx.**Selmer**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Selmer
sage: algo = Selmer()
sage: TestSuite(algo).run()
sage: algo._test_dual_substitution_definition()
sage: algo._test_coherence()
sage: algo._test_definition()
```

dual_substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Selmer
sage: Selmer().dual_substitutions()
{123: WordMorphism: 1->1, 2->2, 3->31,
 132: WordMorphism: 1->1, 2->21, 3->3,
 213: WordMorphism: 1->1, 2->2, 3->32,
 231: WordMorphism: 1->12, 2->2, 3->3,
 312: WordMorphism: 1->1, 2->23, 3->3,
 321: WordMorphism: 1->13, 2->2, 3->3}
```

substitutions()

EXAMPLES:

```
sage: from slabbe.mult_cont_frac_pyx import Selmer
sage: Selmer().substitutions()
{123: WordMorphism: 1->13, 2->2, 3->3,
 132: WordMorphism: 1->12, 2->2, 3->3,
 213: WordMorphism: 1->1, 2->23, 3->3,
 231: WordMorphism: 1->1, 2->21, 3->3,
 312: WordMorphism: 1->1, 2->2, 3->32,
 321: WordMorphism: 1->1, 2->2, 3->31}
```

class slabbe.mult_cont_frac_pyx.**Sorted_ARMonteil**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**Sorted_ARP**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**Sorted_ARPMulti**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**Sorted_ARrevert**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**Sorted_ARrevertMulti**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**Sorted_ArnouxRauzy**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**Sorted_ArnouxRauzyMulti**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**Sorted_Brun**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**Sorted_BrunMulti**

Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**Sorted_Delaunay**
    Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**Sorted_FullySubtractive**
    Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**Sorted_Poincare**
    Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

class slabbe.mult_cont_frac_pyx.**Sorted_Selmer**
    Bases: *slabbe.mult_cont_frac_pyx.MCFAlgorithm*

4.3 Diophantine Approximation

Simultaneous diophantine approximation

EXAMPLES:

```
sage: from slabbe.diophantine_approx import best_simultaneous_convergents
sage: it = best_simultaneous_convergents([e, pi])
sage: dirichlet10 = [next(it) for _ in range(10)]
sage: dirichlet10
[(3, 3, 1),
 (19, 22, 7),
 (1843, 2130, 678),
 (51892, 59973, 19090),
 (113018, 130618, 41577),
 (114861, 132748, 42255),
 (166753, 192721, 61345),
 (446524, 516060, 164267),
 (1174662, 1357589, 432134),
 (3970510, 4588827, 1460669)]
```

The above Dirichlet simultaneous diophantine approximations appear as columns of matrices generated by multidimensional continued fraction algorithms:

```
sage: from slabbe.diophantine_approx import mult_cont_frac_vs_dirichlet
sage: from slabbe.mult_cont_frac import Brun,ARP,Reverse,Selmer,Cassaigne
sage: algos = [Brun(), ARP(), Reverse(), Selmer(),Cassaigne()]
sage: mult_cont_frac_vs_dirichlet([e,pi], dirichlet10, algos)
  Dirichlet                     Brun       ARP        Reverse   Selmer     Cassaigne
+-----------------------------+----------+----------+---------+----------+-----------+
  (3, 3, 1)                     [4, 5]     [3]        []        [8, 12]    []
  (19, 22, 7)                   [9, 16]    [6, 11]    [7, 33]   [32]       [15, 25]
  (1843, 2130, 678)             [22, 27]   [16, 17]   []        [44, 48]   [36, 39]
  (51892, 59973, 19090)         []         []         []        [56]       []
  (113018, 130618, 41577)       []         []         []        []         []
  (114861, 132748, 42255)       [33, 35]   [22, 24]   []        [62, 66]   [51, 53]
  (166753, 192721, 61345)       []         []         []        []         []
  (446524, 516060, 164267)      [36]       [25]       []        []         [56, 57]
  (1174662, 1357589, 432134)    [39, 44]   [26, 29]   []        [68]       [61, 66]
  (3970510, 4588827, 1460669)   []         [28]       []        []         []
```

The indices in the table are the i-th matrices. For example, the first 3 Dirichlet approximations appear in the convergents of ARP algorithm, but not the 4th neither the 5th):

```
sage: algo = ARP()
sage: algo.n_matrix((e,pi,1), 3)
[3 3 2]
[3 4 2]
[1 1 1]
```

(continues on next page)

(continued from previous page)

```
sage: algo.n_matrix((e,pi,1), 6)
[33 19  8]
[38 22  9]
[12  7  3]
sage: algo.n_matrix((e,pi,1), 16)
[1631 2498 1843]
[1885 2887 2130]
[ 600  919  678]
sage: algo.n_matrix((e,pi,1), 22)
[114861 101941  64812]
[132748 117816  74905]
[ 42255  37502  23843]
sage: algo.n_matrix((e,pi,1), 25)
[446524 331663 842999]
[516060 383312 974277]
[164267 122012 310122]
sage: algo.n_matrix((e,pi,1), 26)
[1621186  331663 1174662]
[1873649  383312 1357589]
[ 596401  122012  432134]
sage: algo.n_matrix((e,pi,1), 28)
[3970510 2680987 1174662]
[4588827 3098490 1357589]
[1460669  986280  432134]
```

The Dirichlet vectors can be computed from the precedent ones. So one could expect a perfect MCF algorithm based on 3x3 matrices:

```
sage: from slabbe.diophantine_approx import dirichlet_convergents_dependance
sage: dirichlet_convergents_dependance([e,pi], 8)
  i   vi                          lin. rec.     remainder
+---+---------------------------+-------------+-----------+
  0   (3, 3, 1)                   []            (3, 3, 1)
  1   (19, 22, 7)                 [6]           (1, 4, 1)
  2   (1843, 2130, 678)           [96, 6]       (1, 0, 0)
  3   (51892, 59973, 19090)       [28, 15, 1]   (0, 0, 0)
  4   (113018, 130618, 41577)     [2, 5, 1]     (0, 0, 0)
  5   (114861, 132748, 42255)     [1, 0, 1]     (0, 0, 0)
  6   (166753, 192721, 61345)     [1, 0, 1]     (0, 0, 0)
  7   (446524, 516060, 164267)    [2, 0, 1]     (0, 0, 0)
```

But, looking further, it is not true anymore, because v_10 is not a greedy linear combination of the previous three:

```
sage: dirichlet_convergents_dependance([e,pi], 18)   # long time (7.4s)
  i    vi                                   lin. rec.         remainder
+----+------------------------------------+-----------------+-----------+
  0    (3, 3, 1)                            []                (3, 3, 1)
  1    (19, 22, 7)                          [6]               (1, 4, 1)
  2    (1843, 2130, 678)                    [96, 6]           (1, 0, 0)
  3    (51892, 59973, 19090)                [28, 15, 1]       (0, 0, 0)
  4    (113018, 130618, 41577)              [2, 5, 1]         (0, 0, 0)
  5    (114861, 132748, 42255)              [1, 0, 1]         (0, 0, 0)
  6    (166753, 192721, 61345)              [1, 0, 1]         (0, 0, 0)
  7    (446524, 516060, 164267)             [2, 0, 1]         (0, 0, 0)
  8    (1174662, 1357589, 432134)           [2, 1, 1]         (0, 0, 0)
  9    (3970510, 4588827, 1460669)          [3, 1]            (0, 0, 0)
  10   (21640489, 25010505, 7961091)        [5, 1, 1, 1]      (0, 0, 0)
  11   (25610999, 29599332, 9421760)        [1, 1]            (0, 0, 0)
  12   (47251488, 54609837, 17382851)       [1, 1]            (0, 0, 0)
  13   (117318127, 135587768, 43158927)     [2, 0, 1, 0, 1]   (0, 0, 0)
  14   (142929126, 165187100, 52580687)     [1, 0, 1]         (0, 0, 0)
  15   (164569615, 190197605, 60541778)     [1, 0, 0, 0, 1]   (0, 0, 0)
  16   (307498741, 355384705, 113122465)    [1, 1]            (0, 0, 0)
  17   (779567097, 900967015, 286786708)    [2, 1]            (0, 0, 0)
```

Remark: matrices of independant lines is 1 (with Thomas Garrity, 27oct 2016). He kind of have a proof of that…

This used to give a Value Error:

```
sage: dirichlet_convergents_dependance([e,pi], 19) # not tested (1min 12s)
  i    vi                                      lin. rec.             remainder
+----+--------------------------------------+---------------------+-----------+
  0    (3, 3, 1)                               []                    (3, 3, 1)
  1    (19, 22, 7)                             [6]                   (1, 4, 1)
  2    (1843, 2130, 678)                       [96, 6]               (1, 0, 0)
  3    (51892, 59973, 19090)                   [28, 15, 1]           (0, 0, 0)
  4    (113018, 130618, 41577)                 [2, 5, 1]             (0, 0, 0)
  5    (114861, 132748, 42255)                 [1, 0, 1]             (0, 0, 0)
  6    (166753, 192721, 61345)                 [1, 0, 1]             (0, 0, 0)
  7    (446524, 516060, 164267)                [2, 0, 1]             (0, 0, 0)
  8    (1174662, 1357589, 432134)              [2, 1, 1]             (0, 0, 0)
  9    (3970510, 4588827, 1460669)             [3, 1]                (0, 0, 0)
  10   (21640489, 25010505, 7961091)           [5, 1, 1, 1]          (0, 0, 0)
  11   (25610999, 29599332, 9421760)           [1, 1]                (0, 0, 0)
  12   (47251488, 54609837, 17382851)          [1, 1]                (0, 0, 0)
  13   (117318127, 135587768, 43158927)        [2, 0, 1, 0, 1]       (0, 0, 0)
  14   (142929126, 165187100, 52580687)        [1, 0, 1]             (0, 0, 0)
  15   (164569615, 190197605, 60541778)        [1, 0, 0, 0, 1]       (0, 0, 0)
  16   (307498741, 355384705, 113122465)       [1, 1]                (0, 0, 0)
  17   (779567097, 900967015, 286786708)       [2, 1]                (0, 0, 0)
  18   (8457919940, 9775049397, 3111494861)    [10, 2, 0, 0, 0, 1]   (0, 0, 0)
```

BENCHMARKS:

```
sage: it = best_simultaneous_convergents([e,pi])
sage: %time L = [next(it) for _ in range(15)]   # not tested (4.66s)
sage: it = best_simultaneous_convergents([e,pi])
sage: %time L = [next(it) for _ in range(18)]   # not tested (52s)
```

AUTHORS:

- Sébastien Labbé, September 22, 2016
- Sébastien Labbé, October 10, 2016
- Sébastien Labbé, October 19, 2016, Cython improvements (10000x faster)
- Sébastien Labbé, October 21, 2016, Parallelization of computations

TODO:

- In general, how many of the dirichlet approximations are found by the MCF algos?
- Code Szekeres MCF algorithm

REFERENCES:

Szekeres, G. Multidimensional continued fractions. Ann. Univ. Sci. Budapest. Eötvös Sect. Math. 13 (1970), 113–140 (1971).

Cusick, T. W. The Szekeres multidimensional continued fraction. Math. Comp. 31 (1977), no. 137, 280–317.

`slabbe.diophantine_approx.`**`best_simultaneous_convergents`**`(`*v*`)`

Return an iterator of best convergents to a vector of real number according to Dirichlet theorem on simultaneous approximations.

INPUT:

- v – list of real numbers

OUTPUT:

- iterator

EXAMPLES:

```
sage: from slabbe.diophantine_approx import best_simultaneous_convergents
sage: it = best_simultaneous_convergents([e, pi])
sage: [next(it) for _ in range(5)]
[(3, 3, 1),
 (19, 22, 7),
 (1843, 2130, 678),
 (51892, 59973, 19090),
 (113018, 130618, 41577)]
```

TESTS:

Correspondance with continued fraction when d=1:

```
sage: it = best_simultaneous_convergents([e])
sage: [next(it) for _ in range(10)]
[(3, 1),
 (8, 3),
 (11, 4),
 (19, 7),
 (87, 32),
 (106, 39),
 (193, 71),
 (1264, 465),
 (1457, 536),
 (2721, 1001)]
sage: continued_fraction(e).convergents()[:11].list()
[2, 3, 8/3, 11/4, 19/7, 87/32, 106/39, 193/71, 1264/465, 1457/536, 2721/1001]
```

slabbe.diophantine_approx.**best_simultaneous_convergents_upto**(*v*, *Q*, *start=1*, *verbose=False*)

Return a list of all best simultaneous diophantine approximations p,q of vector `v` at distance `|qv-p|<=1/Q` such that $1 <= q < Q^d$.

INPUT:

- `v` – list of real numbers
- `Q` – real number, Q>1
- `start` – integer (default: 1), starting value to check
- `verbose` – boolean (default: `False`)

EXAMPLES:

```
sage: from slabbe.diophantine_approx import best_simultaneous_convergents_upto
sage: best_simultaneous_convergents_upto([e,pi], 2)
[((3, 3, 1), 3.549646778303845)]
sage: best_simultaneous_convergents_upto([e,pi], 4)
[((19, 22, 7), 35.74901433260719)]
sage: best_simultaneous_convergents_upto([e,pi], 36, start=4**2)
[((1843, 2130, 678), 203.23944293852406)]
sage: best_simultaneous_convergents_upto([e,pi], 204, start=36**2)
[((51892, 59973, 19090), 266.16775098010373),
 ((113018, 130618, 41577), 279.18598227531174)]
sage: best_simultaneous_convergents_upto([e,pi], 280, start=204**2)
[((114861, 132748, 42255), 412.7859137824949),
 ((166753, 192721, 61345), 749.3634909055199)]
sage: best_simultaneous_convergents_upto([e,pi], 750, start=280**2)
[((446524, 516060, 164267), 896.4734658499202),
 ((1174662, 1357589, 432134), 2935.314937919726)]
sage: best_simultaneous_convergents_upto([e,pi], 2936, start=750**2)
[((3970510, 4588827, 1460669), 3654.2989332956854),
 ((21640489, 25010505, 7961091), 6257.014011585661)]
```

TESTS:

```
sage: best_simultaneous_convergents_upto([e,pi], 102300.1, start=10^9) # not tested (1 min)
[((8457919940, 9775049397, 3111494861), 194686.19839453633)]
```

`slabbe.diophantine_approx.`**`dirichlet_convergents_dependance`**(*v*, *n*, *verbose=False*)

INPUT:

- `v` – list of real numbers
- `n` – integer, number of iterations
- `verbose` – bool (default: `False`),

OUTPUT:

- table of linear combinaisons of dirichlet approximations in terms of previous dirichlet approximations

EXAMPLES:

```
sage: from slabbe.diophantine_approx import dirichlet_convergents_dependance
sage: dirichlet_convergents_dependance([e,pi], 4)
  i   vi                      lin. rec.     remainder
+---+-----------------------+-------------+-----------+
  0   (3, 3, 1)               []            (3, 3, 1)
  1   (19, 22, 7)             [6]           (1, 4, 1)
  2   (1843, 2130, 678)       [96, 6]       (1, 0, 0)
  3   (51892, 59973, 19090)   [28, 15, 1]   (0, 0, 0)
```

The last 3 seems enough:

```
sage: dirichlet_convergents_dependance([e,pi], 8)
  i   vi                         lin. rec.     remainder
+---+--------------------------+-------------+-----------+
  0   (3, 3, 1)                  []            (3, 3, 1)
  1   (19, 22, 7)                [6]           (1, 4, 1)
  2   (1843, 2130, 678)          [96, 6]       (1, 0, 0)
  3   (51892, 59973, 19090)      [28, 15, 1]   (0, 0, 0)
  4   (113018, 130618, 41577)    [2, 5, 1]     (0, 0, 0)
  5   (114861, 132748, 42255)    [1, 0, 1]     (0, 0, 0)
  6   (166753, 192721, 61345)    [1, 0, 1]     (0, 0, 0)
  7   (446524, 516060, 164267)   [2, 0, 1]     (0, 0, 0)
```

But not in this case:

```
sage: dirichlet_convergents_dependance([pi,sqrt(3)], 12)
  i    vi                       lin. rec.                  remainder
+----+-----------------------+--------------------------+-----------+
  0    (3, 2, 1)               []                         (3, 2, 1)
  1    (22, 12, 7)             [6]                        (4, 0, 1)
  2    (47, 26, 15)            [2, 1]                     (0, 0, 0)
  3    (69, 38, 22)            [1, 1]                     (0, 0, 0)
  4    (176, 97, 56)           [2, 0, 1, 4]               (4, 1, 1)
  5    (223, 123, 71)          [1, 0, 1]                  (0, 0, 0)
  6    (399, 220, 127)         [1, 1]                     (0, 0, 0)
  7    (1442, 795, 459)        [3, 1, 0, 0, 0, 1]         (0, 0, 0)
  8    (6390, 3523, 2034)      [4, 1, 1]                  (0, 0, 0)
  9    (26603, 14667, 8468)    [4, 0, 2, 1, 0, 0, 0, 1]   (0, 0, 0)
  10   (32993, 18190, 10502)   [1, 1]                     (0, 0, 0)
  11   (40825, 22508, 12995)   [1, 0, 1, 1]               (0, 0, 0)
```

The v4 is not a lin. comb. of the previous four:

```
sage: dirichlet_convergents_dependance([e,pi,sqrt(3)], 5)
  i   vi                                lin. rec.        remainder
+---+---------------------------------+----------------+--------------+
  0   (3, 3, 2, 1)                      []               (3, 3, 2, 1)
```

(continues on next page)

(continued from previous page)

```
  1   (19, 22, 12, 7)                  [6]              (1, 4, 0, 1)
  2   (193, 223, 123, 71)              [10, 1]          (0, 0, 1, 0)
  3   (5529, 6390, 3523, 2034)         [28, 6, 3]       (2, 5, 1, 1)
  4   (163067, 188461, 103904, 59989)  [29, 14, 1, 1]   (2, 4, 1, 1)
```

slabbe.diophantine_approx.**distance_to_nearest_integer**(*x*)

slabbe.diophantine_approx.**frac**(*x*)

Return the fractional part of real number x.

Not always perfect…

EXAMPLES:

```
sage: from slabbe.diophantine_approx import frac
sage: frac(3.2)
0.200000000000000
sage: frac(-3.2)
0.800000000000000
sage: frac(pi)
pi - 3
sage: frac(pi).n()
0.141592653589793
```

This looks suspicious…:

```
sage: frac(pi*10**15).n()
0.000000000000000
```

slabbe.diophantine_approx.**mult_cont_frac_vs_dirichlet**(*v*, *dirichlet*, *algos*)

Returns the indices i such that dirichlet approximations appears as columns of the i-th matrix obtained from mult. dim. cont. frac. algorithms.

INPUT:

- v – list of real numbers
- `dirichlet` – list, first dirichlet approximations
- `algos` – list, list of mult. cont. frac. algorithms

OUTPUT:

table

EXAMPLES:

```
sage: from slabbe.diophantine_approx import best_simultaneous_convergents
sage: from slabbe.diophantine_approx import mult_cont_frac_vs_dirichlet
sage: from slabbe.mult_cont_frac import Brun,ARP,Reverse,Selmer,Cassaigne
sage: v = [e, pi]
sage: it = best_simultaneous_convergents(v)
sage: dirichlet = [next(it) for _ in range(10)]
sage: algos = [Brun(), ARP(), Reverse(), Selmer(),Cassaigne()]
sage: mult_cont_frac_vs_dirichlet(v, dirichlet, algos)
  Dirichlet                     Brun       ARP        Reverse   Selmer     Cassaigne
+-----------------------------+----------+----------+---------+----------+-----------+
  (3, 3, 1)                     [4, 5]     [3]        []        [8, 12]    []
  (19, 22, 7)                   [9, 16]    [6, 11]    [7, 33]   [32]       [15, 25]
  (1843, 2130, 678)             [22, 27]   [16, 17]   []        [44, 48]   [36, 39]
  (51892, 59973, 19090)         []         []         []        [56]       []
  (113018, 130618, 41577)       []         []         []        []         []
  (114861, 132748, 42255)       [33, 35]   [22, 24]   []        [62, 66]   [51, 53]
  (166753, 192721, 61345)       []         []         []        []         []
```

(continues on next page)

(continued from previous page)

```
(446524, 516060, 164267)      [36]        [25]        []        []        [56, 57]
(1174662, 1357589, 432134)    [39, 44]    [26, 29]    []        [68]      [61, 66]
(3970510, 4588827, 1460669)   []          [28]        []        []        []
```

slabbe.diophantine_approx.**mult_cont_frac_vs_dirichlet_dict**(*v*, *dirichlet*, *algos*)

INPUT:

- v – list of real numbers
- dirichlet – list, first dirichlet approximations
- algos – list, list of mult. cont. frac. algorithms

OUTPUT:

dict

EXAMPLES:

```
   sage: from slabbe.diophantine_approx import best_simultaneous_convergents
   sage: from slabbe.diophantine_approx import mult_cont_frac_vs_dirichlet_dict
   sage: from slabbe.mult_cont_frac import ARP, Brun
   sage: v = [e, pi]
   sage: it = best_simultaneous_convergents(v)
   sage: dirichlet = [next(it) for _ in range(3)]
   sage: mult_cont_frac_vs_dirichlet_dict([e,pi], dirichlet, [Brun(), ARP()])
   {Arnoux-Rauzy-Poincar\'e 3-dimensional continued fraction algorithm:
    defaultdict(<type 'list'>, {(19, 22, 7): [6, 7, 8, 9, 10, 11],
    (3, 3, 1): [3], (1843, 2130, 678): [16, 17]}),
    Brun 3-dimensional continued fraction algorithm:
    defaultdict(<type 'list'>, {(19, 22, 7): [9, 10, 11, 12, 13,
    14, 15, 16], (3, 3, 1): [4, 5], (1843, 2130, 678): [22, 23,
    24, 25, 26, 27]})}

Or from precomputed Dirichlet approximations::

   sage: dirichlet = [(3, 3, 1), (19, 22, 7), (1843, 2130, 678), (51892, 59973, 19090)]
   sage: mult_cont_frac_vs_dirichlet_dict([e,pi], dirichlet, [Brun(), ARP()])
   {Arnoux-Rauzy-Poincar\'e 3-dimensional continued fraction algorithm:
    defaultdict(<type 'list'>, {(19, 22, 7): [6, 7, 8, 9, 10, 11],
    (3, 3, 1): [3], (1843, 2130, 678): [16, 17]}),
    Brun 3-dimensional continued fraction algorithm:
    defaultdict(<type 'list'>, {(19, 22, 7): [9, 10, 11, 12, 13,
    14, 15, 16], (3, 3, 1): [4, 5], (1843, 2130, 678): [22, 23,
    24, 25, 26, 27]})}
```

Simultaneous diophantine approximation

EXAMPLES:

The code gets between 1000x faster and 50000x faster compared to the same function in Python code (diophantine_approximation.py):

```
sage: from slabbe.diophantine_approx_pyx import good_simultaneous_convergents_upto
sage: from slabbe.diophantine_approx import _best_simultaneous_convergents_upto
sage: good_simultaneous_convergents_upto([e,pi], 203)     # 493 µs
[678, 19090, 19768]
sage: _best_simultaneous_convergents_upto([e,pi], 203)        # 905 ms
((1843, 2130, 678), 203.239442934072)
sage: 905/493. * 1000
1835.69979716024
```

```
sage: good_simultaneous_convergents_upto([e,pi], 204)    # 493 µs
[19090, 19768, 41577]
```

(continues on next page)

(continued from previous page)

```
sage: _best_simultaneous_convergents_upto([e,pi], 204)        # 25s (not tested)
((51892, 59973, 19090), 266.17)
sage: 25 / 493. * 1000000
50709.9391480730
```

AUTHORS:

- Sébastien Labbé, October 19, 2016

`slabbe.diophantine_approx_pyx.`**`good_simultaneous_convergents_upto`**`()`

Return a list of potential best simultaneous diophantine approximation of vector `v` at distance `1/Q`.

It searches for all possibilities of denominators in the interval [1, Q^d] starting at start by step. Argument step is used for parallel computations purposes.

INPUT:

- `v` – list of real numbers to approximate
- `Q` – real number, Q>1
- `start` – integer (default: 1), starting value to check
- `step` – integer (default: 1), step

OUTPUT:

- list of integers q such that entries of q*v are at most 1/Q from the integer lattice

EXAMPLES:

```
sage: from slabbe.diophantine_approx_pyx import good_simultaneous_convergents_upto
sage: good_simultaneous_convergents_upto([e,pi], 2)
[1, 2, 3, 4]
sage: good_simultaneous_convergents_upto([e,pi], 4)
[7, 14, 15]
sage: good_simultaneous_convergents_upto([e,pi], 35)
[7, 678, 685]
sage: good_simultaneous_convergents_upto([e,pi], 36)
[678, 685]
sage: good_simultaneous_convergents_upto([e,pi], 203)
[678, 19090, 19768]
sage: good_simultaneous_convergents_upto([e,pi], 204)
[19090, 19768, 41577]
```

The previous results mean that these integers are good approximations:

```
sage: v = [e,pi]
sage: [(7 * a).n() for a in v]
[19.0279727992133, 21.9911485751286]
sage: [(678 * a).n() for a in v]
[1842.99507969523, 2129.99981913388]
sage: [(19090 * a).n() for a in v]
[51892.0001052832, 59973.0037570292]
```

Knowing the error of previous results, we can start the next computation at step 678:

```
sage: min(1/abs(678*a - p) for a,p in zip([e,pi], (1843, 2130))).n()
203.239442934072
sage: good_simultaneous_convergents_upto([e,pi], 204, start=678)
[19090, 19768, 41577]
```

Use start and step for parallel computation purposes:

```
sage: good_simultaneous_convergents_upto([e,pi], 2935, start=1)
[432134, 1460669, 7961091, 8393225]
sage: good_simultaneous_convergents_upto([e,pi], 2935, start=1, step=3)
[]
sage: good_simultaneous_convergents_upto([e,pi], 2935, start=2, step=3)
[432134, 1460669, 8393225]
sage: good_simultaneous_convergents_upto([e,pi], 2935, start=3, step=3)
[7961091]
```

TESTS:

```
sage: good_simultaneous_convergents_upto([1,e,pi], 1)
Traceback (most recent call last):
...
ValueError: argument Q(=1.0) must be > 1
```

If the interval is too short, then noting is found:

```
sage: good_simultaneous_convergents_upto([e,pi], 204, start=42000, step=2)
[]
```

No solution was found when the q was $cdef int$ instead of $cdef long$:

```
sage: good_simultaneous_convergents_upto([e,pi], 136500, start=1)  # not tested (2min)
[3111494861]
```

This used to give a Value Error:

```
sage: good_simultaneous_convergents_upto([e,pi], 102300, start=10^9, step=2) # not tested (1min)
[]
sage: good_simultaneous_convergents_upto([e,pi], 102300, start=10^9+1, step=2) # not tested (1min)
[3111494861, 3398281569]
```

4.4 Lyapunov exponents (comparison)

Lyapunov parallel computation for MCF algorithms

slabbe.lyapunov.**lyapunov_comparison_table**(*L*, *n_orbits=100*, *n_iterations=10000*)

Return a table of values of Lyapunov exponents for many algorithm.

INPUT:

- L – list of algorithms
- `n_orbits` – integer
- `n_iterations` – integer

OUTPUT:

table

EXAMPLES:

```
sage: import slabbe.mult_cont_frac as mcf
sage: from slabbe.lyapunov import lyapunov_comparison_table
sage: algos = [mcf.Brun(), mcf.ARP()]
sage: lyapunov_comparison_table(algos)    # abs tol 0.01
  Algorithm                 \#Orbits   $\theta_1$ (std)   $\theta_2$ (std)   $1-\theta_2/\theta_1$ (std)
+-------------------------+----------+------------------+------------------+-----------------------------
↪+
```

(continues on next page)

(continued from previous page)

```
  Arnoux-Rauzy-Poincar\'e   100        0.44 (0.012)       -0.172 (0.0060)     1.388 (0.0054)
  Brun                      100        0.30 (0.011)       -0.113 (0.0049)     1.370 (0.0070)
```

slabbe.lyapunov.**lyapunov_sample**(*algo*, *n_orbits*, *n_iterations=1000*, *verbose=False*)

Return lists of values for theta1, theta2 and 1-theta2/theta1 computed on many orbits.

This is computed in parallel.

INPUT:

- `n_orbits` – integer, number of orbits
- `n_iterations` – integer, length of each orbit
- `verbose` – bool (default: `False`)

OUTPUT:

tuple of three lists

EXAMPLES:

```
sage: from slabbe.lyapunov import lyapunov_sample
sage: from slabbe.mult_cont_frac import Brun
sage: lyapunov_sample(Brun(), 5, 1000000) # abs tol 0.01
[(0.3027620661266397,
  0.3033468535021702,
  0.3044950176856005,
  0.3030531162480779,
  0.30601169862996064),
 (-0.11116236859835525,
  -0.11165563059874498,
  -0.1122595926203868,
  -0.11190323336181864,
  -0.11255687513610782),
 (1.367160820443926,
  1.3680790794750939,
  1.3686746452327765,
  1.3692528714016428,
  1.3678188632657973)]
```

slabbe.lyapunov.**lyapunov_table**(*algo*, *n_orbits*, *n_iterations=1000*)

Return a table of values of Lyapunov exponents for this algorithm.

INPUT:

- `n_orbits` – integer, number of orbits
- `n_iterations` – integer, length of each orbit

OUTPUT:

table of liapounov exponents

EXAMPLES:

```
sage: from slabbe.mult_cont_frac import Brun
sage: from slabbe.lyapunov import lyapunov_table
sage: lyapunov_table(Brun(), 10, 1000000) # random
  10 succesful orbits     min       mean      max       std
+-----------------------+---------+---------+---------+---------+
  $\theta_1$              0.303     0.305     0.307     0.0013
  $\theta_2$              -0.1131   -0.1124   -0.1115   0.00051
  $1-\theta_2/\theta_1$   1.3678    1.3687    1.3691    0.00043
```

4.5 Markov transformations

Markov transformation

EXAMPLES:

...

TODO:

- Remove cylinder code from matrix cocycle
- Remove rounded_string_vector from matrix cocycle

AUTHORS:

- Sébastien Labbé, initial version, January 2016

class `slabbe.markov_transformation.`**`MarkovTransformation`**(*partition*, *transitions*, *linear_maps*)

Bases: `object`

Markov Transformation

INPUT:

- `partition` – dict, mapping each key to a cone (matrix)
- `transitions` – dict, mapping each key to set of keys
- `linear_maps` – dict, mapping each key to a linear map (matrix)

EXAMPLES:

Brun MCF algorithm is a Markov transformation:

```
sage: import itertools
sage: B12 = matrix(3, [1,0,0, 1,1,0, 0,0,1])
sage: B13 = matrix(3, [1,0,0, 0,1,0, 1,0,1])
sage: B21 = matrix(3, [1,1,0, 0,1,0, 0,0,1])
sage: B23 = matrix(3, [1,0,0, 0,1,0, 0,1,1])
sage: B31 = matrix(3, [1,0,1, 0,1,0, 0,0,1])
sage: B32 = matrix(3, [1,0,0, 0,1,1, 0,0,1])
sage: gens = (B23, B32, B13, B31, B12, B21)
sage: alphabet = [123, 132, 213, 231, 312, 321]
sage: partition = dict(zip(alphabet, gens))
sage: def B(i,j,k): return int('{}{}{}'.format(i,j,k))
sage: transitions = {B(i,j,k):[B(i,j,k), B(i,k,j), B(k,i,j)]
....:         for i,j,k in itertools.permutations((1,2,3))}
sage: linear_maps = partition
sage: from slabbe.markov_transformation import MarkovTransformation
sage: T = MarkovTransformation(partition, transitions, linear_maps)
```

`automaton`()

EXAMPLES:

```
sage: from slabbe.markov_transformation import markov_transformations
sage: T = markov_transformations.Selmer()
sage: T.automaton()
Automaton with 12 states
```

`identity_matrix`()

EXAMPLES:

```
sage: from slabbe.markov_transformation import markov_transformations
sage: T = markov_transformations.Selmer()
sage: T.identity_matrix()
[1 0 0]
[0 1 0]
[0 0 1]
```

language()

EXAMPLES:

```
sage: from slabbe.markov_transformation import markov_transformations
sage: T = markov_transformations.Selmer()
sage: T.language()
Regular language over [321, 132, -123, 231, -312, -213, 213,
312, -231, 123, -132, -321]
defined by: Automaton with 12 states
```

n_cylinders_edges(*n*)

EXAMPLES:

```
sage: from slabbe.markov_transformation import markov_transformations
sage: T = markov_transformations.Selmer()
sage: E = T.n_cylinders_edges(1)
sage: len(E)
39
```

n_cylinders_iterator(*n*)

EXAMPLES:

```
sage: from slabbe.markov_transformation import markov_transformations
sage: T = markov_transformations.Selmer()
sage: A,B = zip(*list(T.n_cylinders_iterator(1)))
sage: A[:5]
(word: 321, word: 321, word: 132, word: 132, word: -123)
sage: B
(
[1 3 1]  [2 3 1]  [0 1 0]  [1 1 0]  [1 1 0]  [1 1 1]  [1 3 1]  [2 3 1]
[0 1 1]  [1 1 1]  [1 3 1]  [2 3 1]  [1 2 1]  [1 2 1]  [0 1 0]  [1 1 0]
[0 1 0], [1 1 0], [0 1 1], [1 1 1], [2 2 1], [2 2 1], [0 1 1], [1 1 1],

[1 2 1]  [1 2 1]  [1 2 1]  [1 2 1]  [0 1 1]  [1 1 1]  [0 1 1]  [1 1 1]
[2 2 1]  [2 2 1]  [1 1 0]  [1 1 1]  [0 1 0]  [1 1 0]  [1 3 1]  [2 3 1]
[1 1 0], [1 1 1], [2 2 1], [2 2 1], [1 3 1], [2 3 1], [0 1 0], [1 1 0],

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

n_matrices_iterator(*n*)

EXAMPLES:

```
sage: from slabbe.markov_transformation import markov_transformations
sage: T = markov_transformations.Selmer()
sage: A,B = zip(*list(T.n_matrices_iterator(1)))
sage: A
(word: 321, word: 132, word: -123, word: 231, word: -312, word:
-213, word: 213, word: 312, word: -231, word: 123, word: -132,
word: -321)
sage: B
(
[1 0 1]  [1 0 0]  [1 0 0]  [1 1 0]  [1 0 0]  [1 0 0]  [1 0 0]  [1 0 0]
[0 1 0]  [1 1 0]  [0 1 0]  [0 1 0]  [0 1 1]  [0 1 0]  [0 1 0]  [0 1 1]
```

(continues on next page)

(continued from previous page)

```
[0 0 1], [0 0 1], [1 0 1], [0 0 1], [0 0 1], [0 1 1], [0 1 1], [0 0 1],

[1 1 0]  [1 0 0]  [1 0 0]  [1 0 1]
[0 1 0]  [0 1 0]  [1 1 0]  [0 1 0]
[0 0 1], [1 0 1], [0 0 1], [0 0 1]
)
```

TESTS:

```
sage: list(T.n_matrices_iterator(0))
[(
        [1 0 0]
        [0 1 0]
word: , [0 0 1]
)]
```

n_words_iterator(*n*)

EXAMPLES:

```
sage: from slabbe.markov_transformation import markov_transformations
sage: T = markov_transformations.Selmer()
sage: list( T.n_words_iterator(1))
[word: 321, word: 132, word: -123, word: 231, word: -312, word:
-213, word: 213, word: 312, word: -231, word: 123, word: -132,
word: -321]
```

TESTS:

```
sage: list(T.n_words_iterator(0))
[word: ]
```

plot_n_cylinders(*n*, *labels=True*)

EXAMPLES:

```
sage: from slabbe.markov_transformation import markov_transformations
sage: T = markov_transformations.Selmer()
sage: G = T.plot_n_cylinders(3)
```

TESTS:

```
sage: G = T.plot_n_cylinders(0)
```

tikz_n_cylinders(*n*, *labels=None*, *scale=1*)

INPUT:

- `labels` – None, True or False (default: None), if None, it takes value True if n is 1.

EXAMPLES:

```
sage: from slabbe.markov_transformation import markov_transformations
sage: T = markov_transformations.Selmer()
sage: t = T.tikz_n_cylinders(1, labels=True, scale=4)
sage: t
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\begin{document}
\begin{tikzpicture}
[scale=4]
\draw (-0.8660, -0.5000) -- (-0.3464, -0.2000);
\draw (0.0000, 0.0000) -- (-0.2165, -0.1250);
\draw (0.0000, -0.2000) -- (0.0000, 0.0000);
...
```

(continues on next page)

(continued from previous page)

```
... 56 lines not printed (2702 characters in total) ...
...
\node at (0.2742, 0.0750) {$-132$};
\node at (0.1299, -0.0083) {$-132$};
\node at (-0.0722, -0.2750) {$-321$};
\node at (-0.0722, -0.1083) {$-321$};
\end{tikzpicture}
\end{document}
```

```
sage: from sage.misc.temporary_file import tmp_filename
sage: filename = tmp_filename('temp','.pdf')
sage: _ = t.pdf(filename)
```

word_to_matrix(*w*)

EXAMPLES:

```
sage: from slabbe.markov_transformation import markov_transformations
sage: T = markov_transformations.Selmer()
sage: T.word_to_matrix([123,321,-231])
[1 1 1]
[0 1 0]
[1 1 2]
```

Empty word:

```
sage: T.word_to_matrix([])
[1 0 0]
[0 1 0]
[0 0 1]
```

class slabbe.markov_transformation.**MarkovTransformationGenerators**

Bases: object

Brun()

Selmer()

EXAMPLES:

```
sage: from slabbe.markov_transformation import markov_transformations
sage: T = markov_transformations.Selmer()
sage: T
Markov Transformation with transitions
{321: [321, -321], 132: [132, -132], -123: [-132, -312], 231:
[231, -231], -312: [-321, -231], -213: [-231, -321], 213: [213,
-213], 312: [312, -312], -231: [-213, -123], 123: [123, -123],
-132: [-123, -213], -321: [-312, -132]}
```

slabbe.markov_transformation.**rounded_string_vector**(*v*, *digits=4*)

EXAMPLES:

```
sage: from slabbe.matrix_cocycle import rounded_string_vector
sage: v = (-0.144337567297406, 0.166666666666667)
sage: rounded_string_vector(v)
'(-0.1443, 0.1667)'
sage: rounded_string_vector(v, digits=6)
'(-0.144338, 0.166667)'
```

4.6 Matrices

Matrix functions

EXAMPLES:

```
...
```

TODO:

- Discrete geometry code should use projection_matrix from here

`slabbe.matrices.`**`Minkowski_embedding_without_sqrt2`**`(`*self*, *B=None*, *prec=None*`)`

This method is a modification of the `Minkowski_embedding` method of NumberField in sage (without sqrt2).

EXAMPLES:

```
sage: from slabbe.matrices import Minkowski_embedding_without_sqrt2
sage: F.<alpha> = NumberField(x^3+2)
sage: F.minkowski_embedding()
[ 1.00000000000000 -1.25992104989487  1.58740105196820]
[ 1.41421356237309 0.890898718140339 -1.12246204830937]
[0.000000000000000  1.54308184421705  1.94416129723967]
sage: Minkowski_embedding_without_sqrt2(F)
[  1.00000000000000  -1.25992104989487   1.58740105196820]
[  1.00000000000000  0.629960524947437 -0.793700525984099]
[ 0.000000000000000   1.09112363597172   1.37472963699860]
sage: Minkowski_embedding_without_sqrt2(F, [1, alpha+2, alpha^2-alpha])
[ 1.00000000000000 0.740078950105127  2.84732210186307]
[ 1.00000000000000  2.62996052494744 -1.42366105093154]
[0.000000000000000  1.09112363597172 0.283606001026881]
sage: Minkowski_embedding_without_sqrt2(F) * (alpha + 2).vector().column()
[0.740078950105127]
[ 2.62996052494744]
[ 1.09112363597172]
```

Tribo:

```
sage: F.<beta> = NumberField(x^3-x^2-x-1)
sage: F.minkowski_embedding()
[  1.00000000000000   1.83928675521416   3.38297576790624]
[  1.41421356237309 -0.593465355971987 -0.270804762516626]
[ 0.000000000000000  0.857424571985895 -0.719625086862932]
sage: Minkowski_embedding_without_sqrt2(F)
[  1.00000000000000   1.83928675521416   3.38297576790624]
[  1.00000000000000 -0.419643377607080 -0.191487883953119]
[ 0.000000000000000  0.606290729207199 -0.508851778832738]
```

Comprendre le problème de norme:

```
sage: norme = lambda v:abs(v[0]) * (v[1]^2 + v[2]^2)
sage: F.<beta> = NumberField(x^3-x^2-x-1)
sage: M = Minkowski_embedding_without_sqrt2(F)
sage: norme(M*vector((1,0,0)))
1.00000000000000
sage: norme(M*vector((1,0,-1)))
4.00000000000000
```

`slabbe.matrices.`**`Minkowski_projection_pair`**`(`*self*, *B=None*, *prec=None*`)`

Return the projections to the expanding and contracting spaces.

OUTPUT:

- tuple (A, B) of matrices

EXAMPLES:

```
sage: from slabbe.matrices import Minkowski_projection_pair
sage: F.<alpha> = NumberField(x^3+2)
```

(continues on next page)

(continued from previous page)

```
sage: Minkowski_projection_pair(F)
(
[  1.00000000000000  -1.25992104989487   1.58740105196820]
[  1.00000000000000  0.629960524947437 -0.793700525984099]
[ 0.000000000000000   1.09112363597172   1.37472963699860], []
)
sage: Minkowski_projection_pair(F, [1, alpha+2, alpha^2-alpha])
(
[ 1.00000000000000 0.740078950105127  2.84732210186307]
[ 1.00000000000000  2.62996052494744 -1.42366105093154]
[0.000000000000000  1.09112363597172 0.283606001026881], []
)
```

Tribo:

```
sage: F.<beta> = NumberField(x^3-x^2-x-1)
sage: Minkowski_projection_pair(F)
(
[1.00000000000000000000000000000 1.83928675521416113255185256467l
3.38297576790623749412270853652l],
[  1.00000000000000 -0.419643377607080 -0.191487883953119]
[ 0.000000000000000  0.606290729207199 -0.508851778832738]
)
```

slabbe.matrices.**conjugate_matrix_Z**(*M*)

Return the conjugate matrix Z as defined in [1].

EXAMPLES:

```
sage: from slabbe.matrices import conjugate_matrix_Z
sage: M = matrix(2, [11,29,14,-1])
sage: conjugate_matrix_Z(M)       # abs tol 1e-8
[11.674409930010482  27.69820597163912]
[14.349386111618157  -1.67440993001048]
sage: conjugate_matrix_Z(M)^2     # abs tol 1e-8
[533.7440993001048 276.9820597163913]
[143.4938611161816 400.2559006998952]
```

```
sage: M = matrix(2, [-11,14,-26,29])
sage: conjugate_matrix_Z(M)      # abs tol 1e-8
[ 7.200000000000004  4.199999999999998]
[ 7.799999999999995 10.800000000000002]
sage: conjugate_matrix_Z(M) * 5      # abs tol 1e-8
[ 36.00000000000002 20.999999999999993]
[ 38.99999999999998 54.000000000000014]
```

```
sage: M = matrix(2, [-11,26,-14,29]) / 15
sage: conjugate_matrix_Z(M)      # abs tol 1e-8
[ 0.5999999999999999  0.3999999999999999]
[0.39999999999999986  0.5999999999999999]
```

REFERENCES:

[1] Labbé, Jean-Philippe, et Sébastien Labbé. « A Perron theorem for matrices with negative entries and applications to Coxeter groups ». arXiv:1511.04975 [math], 16 novembre 2015. http://arxiv.org/abs/1511.04975.

slabbe.matrices.**is_nonnegative**(*M*)

EXAMPLES:

```
sage: from slabbe.matrices import is_nonnegative
sage: m = matrix(4, range(-8,8))
sage: is_nonnegative(m)
```

(continues on next page)

(continued from previous page)

```
False
sage: m = matrix(4, range(16))
sage: is_nonnegative(m)
True
```

slabbe.matrices.**is_pisot**(*M*)

EXAMPLES:

```
sage: from slabbe.matrices import is_pisot
sage: M = matrix(2,[1,1,0,1])
sage: is_pisot(M)
False
```

```
sage: M = matrix(2,[0,1,1,1])
sage: is_pisot(M)
True
```

slabbe.matrices.**is_positive**(*M*)

EXAMPLES:

```
sage: from slabbe.matrices import is_positive
sage: m = matrix(4, range(16))
sage: is_positive(m)
False
sage: m = matrix(4, range(1,17))
sage: is_positive(m)
True
```

slabbe.matrices.**is_primitive**(*M*)

EXAMPLES:

```
sage: from slabbe.matrices import is_primitive
sage: m = matrix(2, [0,1,1,1])
sage: is_primitive(m)
True
sage: m = matrix(2, [1,1,0,1])
sage: is_primitive(m)
False
```

slabbe.matrices.**map_coefficients_to_variable_index**(*M*, *x*)

INPUT:

- `M` – matrix
- `x` – string, variable

EXAMPLES:

```
sage: from slabbe.matrices import map_coefficients_to_variable_index
sage: M = matrix(2, range(4))
sage: map_coefficients_to_variable_index(M, 's')
[s_0 s_1]
[s_2 s_3]
sage: latex(_)
\left(\begin{array}{rr}
s_{0} & s_{1} \\
s_{2} & s_{3}
\end{array}\right)
```

slabbe.matrices.**perron_right_eigenvector**(*M*)

EXAMPLES:

```
sage: from slabbe.matrices import perron_right_eigenvector
sage: m = matrix(2,[-11,14,-26,29])
sage: perron_right_eigenvector(m)    # abs tol 0.0000001
(15.0, (0.35, 0.65))
```

`slabbe.matrices.`**`projection_matrix`**(*dim_from=3*, *dim_to=2*)

Return a projection matrix from R^d to R^l.

INPUT:

- `dim_from` -- integer (default: ``3)`
- `dim_to` -- integer (default: ``2)`

OUTPUT:

matrix

EXAMPLES:

```
sage: from slabbe.matrices import projection_matrix
sage: projection_matrix(3,2)
[-0.866025403784439  0.866025403784439  0.000000000000000]
[-0.500000000000000 -0.500000000000000   1.00000000000000]
sage: projection_matrix(2,3)
[-0.577350269189626 -0.333333333333333]
[ 0.577350269189626 -0.333333333333333]
[ 0.000000000000000  0.666666666666667]
```

`slabbe.matrices.`**`rauzy_projection`**(*M*, *beta=None*, *prec=53*)

Returns a projection matrix of the canonical basis using the Minkowski embedding associated to the left eigenvector of the given eigenvalue.

INPUT:

- `beta` - a real element of `QQbar` of degree >= 2 (default: `None`). The eigenvalue used for the projection. It must be an eigenvalue of `M`. The one used by default is the maximal eigenvalue of `M` (usually a Pisot number), but matrices of order larger than 3 letters other interesting choices are sometimes possible.
- `prec` - integer (default: `53`), the number of bits used in the floating point representations of the coordinates.

OUTPUT:

matrix

EXAMPLES:

Fibonacci:

```
sage: from slabbe.matrices import rauzy_projection
sage: m = matrix(2,(1,1,1,0))
sage: m
[1 1]
[1 0]
sage: rauzy_projection(m)
[ 1.00000000000000000000000000000000 -1.61803398874989484820458683436]
[ 1.00000000000000000000000000000000 0.618033988749894848204586834365]
```

Tribonacci:

```
sage: m = matrix(3, [1,1,1, 1,0,0, 0,1,0])
sage: rauzy_projection(m)
[  1.00000000000000  0.839286755214161  0.543689012692076]
[  1.00000000000000  -1.41964337760708 -0.771844506346038]
[ 0.000000000000000  0.606290729207199  -1.11514250803994]
```

(continues on next page)

(continued from previous page)

```
sage: matrix(2,(0,1,0, 0,0,-1))*rauzy_projection(m)
[  1.00000000000000  -1.41964337760708 -0.771844506346038]
[ 0.000000000000000 -0.606290729207199   1.11514250803994]
```

which corresponds to the Rauzy fractal projection coded by Timo:

```
sage: s = WordMorphism('1->12,2->13,3->1')
sage: s.rauzy_fractal_projection()
{'1': (1.00000000000000, 0.000000000000000),
 '2': (-1.41964337760708, -0.606290729207199),
 '3': (-0.771844506346038, 1.11514250803994)}
```

TESTS:

```
sage: t = WordMorphism('1->12,2->3,3->45,4->5,5->6,6->7,7->8,8->1')
sage: m = matrix(t)
sage: rauzy_projection(m).T
[  1.00000000000000   1.00000000000000  0.000000000000000]
[ 0.324717957244746  -1.66235897862237  0.562279512062301]
[ 0.430159709001947  0.784920145499027  -1.30714127868205]
[ 0.245122333753307   1.87743883312335  0.744861766619744]
[ 0.324717957244746  -1.66235897862237  0.562279512062301]
[ 0.430159709001947  0.784920145499027  -1.30714127868205]
[ 0.569840290998053  0.215079854500973   1.30714127868205]
[ 0.754877666246693 -0.877438833123346 -0.744861766619744]
sage: t.rauzy_fractal_projection()
{'1': (1.00000000000000, 0.000000000000000),
 '2': (-1.66235897862237, -0.562279512062301),
 '3': (0.784920145499027, 1.30714127868205),
 '4': (1.87743883312335, -0.744861766619744),
 '5': (-1.66235897862237, -0.562279512062301),
 '6': (0.784920145499027, 1.30714127868205),
 '7': (0.215079854500973, -1.30714127868205),
 '8': (-0.877438833123346, 0.744861766619744)}
```

```
sage: E = t.incidence_matrix().eigenvalues()
sage: x = [x for x in E if -0.8 < x < -0.7][0]
sage: x
-0.7548776662466928?
sage: rauzy_projection(m, beta=x).T
[  1.00000000000000   1.00000000000000  0.000000000000000]
[ -1.75487766624669 -0.122561166876654  0.744861766619744]
[  1.32471795724475 -0.662358978622373  0.562279512062301]
[ -4.07959562349144 -0.460202188254281  0.182582254557443]
[  3.07959562349144 -0.539797811745719 -0.182582254557443]
[ -2.32471795724475 -0.337641021377627 -0.562279512062301]
[  1.75487766624669  0.122561166876654 -0.744861766619744]
[ -1.32471795724475  0.662358978622373 -0.562279512062301]
sage: t.rauzy_fractal_projection(eig=x)
{'1': (1.00000000000000, 0.000000000000000),
 '2': (-0.122561166876654, -0.744861766619744),
 '3': (-0.662358978622373, -0.562279512062301),
 '4': (-0.460202188254281, -0.182582254557443),
 '5': (-0.539797811745719, 0.182582254557443),
 '6': (-0.337641021377627, 0.562279512062301),
 '7': (0.122561166876654, 0.744861766619744),
 '8': (0.662358978622373, 0.562279512062301)}
```

AUTHORS:

- Timo Jolivet (2012-06-16) – for substitutions in Sage
- Sébastien Labbé (2018-03-08) – for matrices, using Minkowski embedding

slabbe.matrices.**recurrence_matrix**(*coeffs*)

Return the recurrence matrix of a relation, for example:

INPUT:

- `coeffs` – list of integers, for example if R(n) = R(n-1) + 2 R(n-2) + 3R(n-3) + 4R(n-4) + 5R(n-5) then coeff must be [1,2,3,4,5]

EXAMPLES:

```
sage: from slabbe.matrices import recurrence_matrix
sage: recurrence_matrix([1,2,3,4,5])
[1 2 3 4 5]
[1 0 0 0 0]
[0 1 0 0 0]
[0 0 1 0 0]
[0 0 0 1 0]
```

slabbe.matrices.**spectrum**(*M*)

EXAMPLES:

```
sage: from slabbe.matrices import spectrum, recurrence_matrix
sage: M = recurrence_matrix([1,2,3,4,5])
sage: spectrum(M)
2.576021761956651?
```

4.7 Polyhedron partition and Induction

Polyhedron partition, polyhedron exchange transformations and induced transformations

EXAMPLES:

A polyhedron partition:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/3
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (h,0)])
sage: r = Polyhedron([(h,1), (1,1), (1,h), (h,0)])
sage: s = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s})
sage: P.is_pairwise_disjoint()
True
sage: list(P)
[(0, A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 3 vertices),
 (1, A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 4 vertices),
 (2, A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 4 vertices),
 (3, A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 3 vertices)]
sage: G = P.plot()
```

Applying a rationnal rotation:

```
sage: from slabbe import PolyhedronExchangeTransformation as PET
sage: base = identity_matrix(2)
sage: translation = vector((2/3, 0))
sage: u = PET.toral_translation(base, translation)
sage: Q = u(P)
sage: Q
Polyhedron partition of 4 atoms with 4 letters
```

Inducing an irrationnal rotation on a subdomain:

```
sage: z = polygen(QQ, 'z') #z = QQ['z'].0 # same as
sage: K = NumberField(z**2-z-1, 'phi', embedding=RR(1.6))
sage: phi = K.gen()
sage: h = 1/phi^2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (h,0)])
sage: r = Polyhedron([(h,1), (1,1), (1,h), (h,0)])
sage: s = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s}, base_ring=K)
sage: base = identity_matrix(2)
sage: translation = vector((1/phi, 0))
sage: u = PET.toral_translation(base, translation)
sage: ieq = [h, -1, 0]   # x0 <= h
sage: P1,sub01 = u.induced_partition(ieq, P)
sage: P1
Polyhedron partition of 7 atoms with 7 letters
sage: sub01
{0: [0, 2],
 1: [1, 2],
 2: [1, 3],
 3: [0, 2, 2],
 4: [1, 2, 2],
 5: [1, 3, 2],
 6: [1, 3, 3]}
```

AUTHORS:

- Sébastien Labbé, November 2017, initial version of polyhedron partitions
- Sébastien Labbé, January 2019, added a class for polyhedron exchange transformations

class `slabbe.polyhedron_partition.`**`PolyhedronExchangeTransformation`**(*partition*, *translations*)

Bases: `object`

Polyhedron Exchange Transformation (PET).

INPUT:

- `partition` – a polyhedron partition
- `translations` – list or dict

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition, PolyhedronExchangeTransformation
sage: h = 1/3
sage: p = Polyhedron([(0,0),(h,0),(h,1),(0,1)])
sage: q = Polyhedron([(1,0),(h,0),(h,1),(1,1)])
sage: P = PolyhedronPartition({0:p, 1:q})
sage: T = {0:(1-h,0), 1:(-h,0)}
sage: PolyhedronExchangeTransformation(P, T)
Polyhedron Exchange Transformation of
Polyhedron partition of 2 atoms with 2 letters
with translations {0: (2/3, 0), 1: (-1/3, 0)}
```

REFERENCES:

- Schwartz, Richard Evan. The Octagonal PETs. First Edition edition. Providence, Rhode Island: American Mathematical Society, 2014.

`ambient_space`()

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition, PolyhedronExchangeTransformation
sage: h = 1/3
sage: p = Polyhedron([(0,0),(h,0),(h,1),(0,1)])
```

(continues on next page)

(continued from previous page)

```
sage: q = Polyhedron([(1,0),(h,0),(h,1),(1,1)])
sage: P = PolyhedronPartition({0:p, 1:q})
sage: T = {0:(1-h,0), 1:(-h,0)}
sage: F = PolyhedronExchangeTransformation(P, T)
sage: F.ambient_space()
Vector space of dimension 2 over Rational Field
```

cylinder(*word*, *partition=None*, *key_fn=None*)

Return the region associated to the coding word.

INPUT:

- `word` – list
- `partition` – polyhedron partition (default:`None`), if None, it uses the domain partition of the transformation
- `key_fn` – function (default:`lambda a,b:(a,b)`), the concatenation function

OUTPUT:

polyhedron partition

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
```

```
sage: from slabbe import PolyhedronExchangeTransformation as PET
sage: base = identity_matrix(2)
sage: translation = vector((1/3, 0))
sage: u = PET.toral_translation(base, translation)
sage: c = u.cylinder([2,2], P); c
Polyhedron partition of 1 atoms with 1 letters
sage: c.alphabet()
{(2, 2)}
```

```
sage: u.cylinder([1,1], P)
Polyhedron partition of 2 atoms with 1 letters
sage: u.cylinder([1], P)
Polyhedron partition of 1 atoms with 1 letters
```

Cylinders of words of length 0:

```
sage: u.cylinder([], P).volume()
1
```

Cylinders of words of length 1:

```
sage: C1 = [u.cylinder([a], P).volume() for a in range(3)]
sage: C1
[1/8, 3/4, 1/8]
sage: sum(C1)
1
```

Cylinders of words of length 2:

```
sage: import itertools
sage: L2 = itertools.product(range(3),repeat=2)
sage: C2 = [u.cylinder([a,b], P).volume() for (a,b) in L2]
sage: C2
[1/72, 1/9, 0, 1/9, 19/36, 1/9, 0, 1/9, 1/72]
sage: sum(C2)
1
```

Cylinders of words of length 3:

```
sage: L3 = itertools.product(range(3),repeat=3)
sage: C3 = [u.cylinder([a,b,c], P).volume() for (a,b,c) in L3]
sage: sum(C3)
1
```

TESTS:

```
sage: u.cylinder([0,0,0], P)
Polyhedron partition of 0 atoms with 0 letters
sage: u.cylinder([2,3], P)
Polyhedron partition of 0 atoms with 0 letters
sage: u.cylinder([2,1], P)
Polyhedron partition of 1 atoms with 1 letters
sage: u.cylinder([], P)
Polyhedron partition of 3 atoms with 3 letters
```

cylinders(*size*, *partition=None*, *key_fn=None*)

Return the cylinders of given size.

INPUT:

- `size` – nonnegative integer
- `partition` – polyhedron partition (default:`None`), if None, it uses the domain partition of the transformation
- `key_fn` – function (default:`lambda a,b:a+b` and every key of atoms of the partition is changed into a singleton tuple), the concatenation function

OUTPUT:

polyhedron partition

EXAMPLES:

```
sage: from slabbe import PolyhedronExchangeTransformation as PET
sage: base = identity_matrix(2)
sage: translation = vector((1/3, 0))
sage: u = PET.toral_translation(base, translation)
sage: [u.cylinders(i) for i in range(5)]
[Polyhedron partition of 1 atoms with 1 letters,
 Polyhedron partition of 2 atoms with 2 letters,
 Polyhedron partition of 3 atoms with 3 letters,
 Polyhedron partition of 3 atoms with 3 letters,
 Polyhedron partition of 3 atoms with 3 letters]
sage: [u.cylinders(i).alphabet() for i in range(5)]
[{()},
 {(0,), (1,)},
 {(0, 0), (0, 1), (1, 0)},
 {(0, 0, 1), (0, 1, 0), (1, 0, 0)},
 {(0, 0, 1, 0), (0, 1, 0, 0), (1, 0, 0, 1)}]
```

domain()

Return the domain of the exchange transformation.

OUTPUT:

a polyhedron

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition, PolyhedronExchangeTransformation
sage: h = 1/3
sage: p = Polyhedron([(0,0),(h,0),(h,1),(0,1)])
sage: q = Polyhedron([(1,0),(h,0),(h,1),(1,1)])
sage: P = PolyhedronPartition({0:p, 1:q})
sage: T = {0:(1-h,0), 1:(-h,0)}
sage: F = PolyhedronExchangeTransformation(P, T)
sage: F.domain()
A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 4 vertices
sage: F.domain().vertices()
(A vertex at (0, 0),
 A vertex at (0, 1),
 A vertex at (1, 0),
 A vertex at (1, 1))
```

image_partition()

Return the partition of the image.

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition, PolyhedronExchangeTransformation
sage: h = 1/3
sage: p = Polyhedron([(0,0),(h,0),(h,1),(0,1)])
sage: q = Polyhedron([(1,0),(h,0),(h,1),(1,1)])
sage: P = PolyhedronPartition({0:p, 1:q})
sage: T = {0:(1-h,0), 1:(-h,0)}
sage: F = PolyhedronExchangeTransformation(P, T)
sage: F.image_partition()
Polyhedron partition of 2 atoms with 2 letters
```

induced_in_partition(*ieq*, *partition=None*)

Returns the partition of the induced transformation on the domain. given by an inequality.

INPUT:

- `ieq` – list, an inequality. An entry equal to "[-1,7,3,4]" represents the inequality 7x_1+3x_2+4x_3>= 1.
- `partition` – polyhedron partition (default:`None`), if None, it uses the domain partition of the transformation

OUTPUT:

dict of polyhedron partitions with keys giving the return time

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/3
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (h,0)])
sage: r = Polyhedron([(h,1), (1,1), (1,h), (h,0)])
sage: s = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s})
```

```
sage: from slabbe import PolyhedronExchangeTransformation as PET
sage: base = identity_matrix(2)
sage: translation = vector((1/3, 0))
sage: u = PET.toral_translation(base, translation)
```

(continues on next page)

(continued from previous page)

```
sage: ieq = [h, -1, 0]   # x0 <= h
sage: u.induced_in_partition(ieq, P)
{3: Polyhedron partition of 4 atoms with 4 letters}
```

```
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s})
sage: ieq2 = [1/2, -1, 0]   # x0 <= 1/2
sage: d = u.induced_in_partition(ieq2, P)
sage: d
{1: Polyhedron partition of 2 atoms with 2 letters,
 2: Polyhedron partition of 3 atoms with 3 letters,
 3: Polyhedron partition of 4 atoms with 4 letters}
```

induced_out_partition(*ieq*, *partition=None*)

Returns the output partition obtained as the induction of the transformation on the domain given by an inequality.

Note: the output partition corresponds to the arrival partition in the domain, not the initial one.

INPUT:

- `ieq` – list, an inequality. An entry equal to "[-1,7,3,4]" represents the inequality 7x_1+3x_2+4x_3>= 1.
- `partition` – polyhedron partition (default:`None`), if None, it uses the domain partition of the transformation

OUTPUT:

dict of polyhedron partitions with keys giving the return time

EXAMPLES:

```
sage: from slabbe import PolyhedronExchangeTransformation as PET
sage: base = identity_matrix(2)
sage: translation = vector((1/3, 0))
sage: u = PET.toral_translation(base, translation)
sage: ieq = [1/2, -1, 0]   # x0 <= 1/2
sage: d = u.induced_out_partition(ieq)
sage: [(i, d[i], d[i].alphabet()) for i in d]
[(1, Polyhedron partition of 1 atoms with 1 letters, {(0,)}),
 (2, Polyhedron partition of 1 atoms with 1 letters, {(0, 1)}),
 (3, Polyhedron partition of 1 atoms with 1 letters, {(0, 0, 1)})]
```

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/3
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (h,0)])
sage: r = Polyhedron([(h,1), (1,1), (1,h), (h,0)])
sage: s = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s})
sage: ieq = [h, -1, 0]   # x0 <= h
sage: d = u.induced_out_partition(ieq, P)
sage: [(i, d[i], d[i].alphabet()) for i in d]
[(3,
  Polyhedron partition of 4 atoms with 4 letters,
  {(0, 2, 2), (1, 2, 2), (1, 2, 3), (1, 3, 3)})]
```

```
sage: ieq2 = [1/2, -1, 0]   # x0 <= 1/2
sage: d = u.induced_out_partition(ieq2, P)
sage: [(i, d[i], d[i].alphabet()) for i in d]
[(1, Polyhedron partition of 2 atoms with 2 letters, {(0,), (1,)}),
 (2, Polyhedron partition of 3 atoms with 3 letters, {(2, 2), (2, 3), (3, 3)}),
 (3,
```

(continues on next page)

(continued from previous page)

```
  Polyhedron partition of 4 atoms with 4 letters,
  {(0, 2, 2), (1, 2, 2), (1, 2, 3), (1, 3, 3)})]
sage: Q = PolyhedronPartition(d[1].atoms()+d[2].atoms()+d[3].atoms())
sage: Q.is_pairwise_disjoint()
True
```

```
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s})
sage: ieq3 = [-1/2, 1, 0]   # x0 >= 1/2
sage: u.induced_out_partition(ieq3, P)
{1: Polyhedron partition of 2 atoms with 2 letters,
 2: Polyhedron partition of 3 atoms with 3 letters,
 3: Polyhedron partition of 4 atoms with 4 letters}
```

It is an error if the induced region is empty:

```
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s})
sage: ieq4 = [-1/2, -1, 0]   # x0 <= -1/2
sage: u.induced_out_partition(ieq4, P)
Traceback (most recent call last):
...
ValueError: Inequality An inequality (-2, 0) x - 1 >= 0 does
not intersect P (=Polyhedron partition of 4 atoms with 4
letters)
```

The whole domain:

```
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s})
sage: ieq5 = [1/2, 1, 0]   # x0 >= -1/2
sage: d = u.induced_out_partition(ieq5, P)
sage: [(i, d[i], d[i].alphabet()) for i in d]
[(1,
  Polyhedron partition of 6 atoms with 4 letters,
  {(0,), (1,), (2,), (3,)})]
```

An irrational rotation:

```
sage: z = polygen(QQ, 'z') #z = QQ['z'].0 # same as
sage: K = NumberField(z**2-z-1, 'phi', embedding=RR(1.6))
sage: phi = K.gen()
sage: h = 1/phi^2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (h,0)])
sage: r = Polyhedron([(h,1), (1,1), (1,h), (h,0)])
sage: s = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s}, base_ring=K)
sage: base = identity_matrix(2)
sage: translation = vector((1/phi, 0))
sage: u = PET.toral_translation(base, translation)
sage: ieq = [phi^-4, -1, 0]   # x0 <= phi^-4
sage: d = u.induced_out_partition(ieq, P)
sage: d
{5: Polyhedron partition of 6 atoms with 6 letters,
 8: Polyhedron partition of 9 atoms with 9 letters}
```

induced_partition(*ieq*, *partition=None*, *substitution_type='dict'*)

Returns the partition of the induced transformation on the domain.

INPUT:

- `ieq` – list, an inequality. An entry equal to "[-1,7,3,4]" represents the inequality 7x_1+3x_2+4x_3>= 1.
- `partition` – polyhedron partition (default:`None`), if None, it uses the domain partition of the transformation

- substitution_type – string (default:'dict'), if 'column' or 'row', it returns a substitution2d, otherwise it returns a dict.

OUTPUT:

- a polyhedron partition
- a substitution2d or a dict

EXAMPLES:

```
sage: from slabbe import PolyhedronExchangeTransformation as PET
sage: base = identity_matrix(2)
sage: translation = vector((1/3, 0))
sage: u = PET.toral_translation(base, translation)
```

We compute the induced partition of a polyhedron exchange transformation on a subdomain given by an inequality:

```
sage: ieq = [1/3, -1, 0]   # x0 <= 1/3
sage: u.induced_partition(ieq)
(Polyhedron partition of 1 atoms with 1 letters,
 {0: [0, 0, 1]})
sage: ieq = [1/2, -1, 0]   # x0 <= 1/2
sage: u.induced_partition(ieq)
(Polyhedron partition of 3 atoms with 3 letters,
 {0: [0], 1: [0, 1], 2: [0, 0, 1]})
```

The second output can be turned into a column or a row Substitution2d if desired:

```
sage: u.induced_partition(ieq, substitution_type='row')
(Polyhedron partition of 3 atoms with 3 letters,
 Substitution 2d: {0: [[0]], 1: [[0], [1]], 2: [[0], [0], [1]]})
```

Now we construct a another coding partition:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/3
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (h,0)])
sage: r = Polyhedron([(h,1), (1,1), (1,h), (h,0)])
sage: s = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s})
```

We use this other partition to compute the induced partition:

```
sage: ieq = [h, -1, 0]   # x0 <= h
sage: Q,sub = u.induced_partition(ieq, P)
sage: Q
Polyhedron partition of 4 atoms with 4 letters
sage: sub
{0: [0, 2, 2], 1: [1, 2, 2], 2: [1, 2, 3], 3: [1, 3, 3]}
```

```
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s})
sage: ieq2 = [1/2, -1, 0]   # x0 <= 1/2
sage: Q,sub = u.induced_partition(ieq2, P)
sage: Q
Polyhedron partition of 9 atoms with 9 letters
sage: sub
{0: [0],
 1: [1],
 2: [2, 2],
 3: [2, 3],
 4: [3, 3],
```

(continues on next page)

(continued from previous page)

```
 5: [0, 2, 2],
 6: [1, 2, 2],
 7: [1, 2, 3],
 8: [1, 3, 3]}
```

Irrationnal rotations:

```
sage: z = polygen(QQ, 'z') #z = QQ['z'].0 # same as
sage: K = NumberField(z**2-z-1, 'phi', embedding=RR(1.6))
sage: phi = K.gen()
sage: h = 1/phi^2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (h,0)])
sage: r = Polyhedron([(h,1), (1,1), (1,h), (h,0)])
sage: s = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s}, base_ring=K)
sage: base = identity_matrix(2)
sage: translation = vector((1/phi, 0))
sage: u = PET.toral_translation(base, translation)
sage: ieq = [h, -1, 0]   # x0 <= h
sage: P1,sub01 = u.induced_partition(ieq, P)
sage: P1
Polyhedron partition of 7 atoms with 7 letters
sage: sub01
{0: [0, 2],
 1: [1, 2],
 2: [1, 3],
 3: [0, 2, 2],
 4: [1, 2, 2],
 5: [1, 3, 2],
 6: [1, 3, 3]}
```

We do the induction on a smaller domain:

```
sage: ieq2 = [1/phi^3, -1, 0]   # x0 <= h
sage: P2,sub02 = u.induced_partition(ieq2, P)
sage: P2
Polyhedron partition of 10 atoms with 10 letters
sage: sub02
{0: [0, 2, 2],
 1: [1, 2, 2],
 2: [1, 3, 2],
 3: [1, 3, 3],
 4: [0, 2, 0, 2, 2],
 5: [0, 2, 1, 2, 2],
 6: [1, 2, 1, 2, 2],
 7: [1, 2, 1, 3, 2],
 8: [1, 3, 1, 3, 2],
 9: [1, 3, 1, 3, 3]}
```

We check that inductions commute:

```
sage: base = diagonal_matrix((phi^-2,1))
sage: translation = vector((phi^-3, 0))
sage: u1 = PET.toral_translation(base, translation)
sage: P2_alt,sub12 = u1.induced_partition(ieq2, P1)
sage: P2_alt
Polyhedron partition of 10 atoms with 10 letters
sage: P2_alt == P2
True
```

Up to a permutation of the alphabet, `sub02` and `sub01*sub12` are equal:

```
sage: s01 = WordMorphism(sub01)
sage: s12 = WordMorphism(sub12)
sage: s02 = WordMorphism(sub02)
sage: s02
WordMorphism: 0->022, 1->122, 2->132, 3->133, 4->02022, 5->02122, 6->12122, 7->12132, 8->13132, 9->
↪13133
sage: s01*s12 == s02
True
```

By chance, the above is true, but in general, we have:

```
sage: perm = WordMorphism(P2.keys_permutation(P2_alt))
sage: perm
WordMorphism: 0->0, 1->1, 2->2, 3->3, 4->4, 5->5, 6->6, 7->7, 8->8, 9->9
sage: s01*s12*perm == s02
True
```

induced_transformation(*ieq*)

Return the induced transformation on the domain.

INPUT:

- `ieq` – list, an inequality. An entry equal to "[-1,7,3,4]" represents the inequality 7x_1+3x_2+4x_3>= 1.

OUTPUT:

- a polyhedron exchange transformation on the subdomain
- a substitution (dict)

EXAMPLES:

```
sage: from slabbe import PolyhedronExchangeTransformation as PET
sage: base = identity_matrix(2)
sage: translation = vector((1/3, 0))
sage: u = PET.toral_translation(base, translation)
```

We compute the induced transformation of a polyhedron exchange transformation on a subdomain given by an inequality:

```
sage: ieq = [1/2, -1, 0]   # x0 <= 1/2
sage: T,sub = u.induced_transformation(ieq)
sage: T
Polyhedron Exchange Transformation of
Polyhedron partition of 3 atoms with 3 letters
with translations {0: (1/3, 0), 1: (-1/3, 0), 2: (0, 0)}
sage: sub
{0: (0,), 1: (0, 1), 2: (0, 0, 1)}
```

inverse()

Return the inverse of self.

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition, PolyhedronExchangeTransformation
sage: h = 1/3
sage: p = Polyhedron([(0,0),(h,0),(h,1),(0,1)])
sage: q = Polyhedron([(1,0),(h,0),(h,1),(1,1)])
sage: P = PolyhedronPartition({0:p, 1:q})
sage: T = {0:(1-h,0), 1:(-h,0)}
sage: F = PolyhedronExchangeTransformation(P, T)
sage: F
Polyhedron Exchange Transformation of
```

(continues on next page)

(continued from previous page)

```
Polyhedron partition of 2 atoms with 2 letters
with translations {0: (2/3, 0), 1: (-1/3, 0)}
```

```
sage: F.inverse()
Polyhedron Exchange Transformation of
Polyhedron partition of 2 atoms with 2 letters
with translations {0: (-2/3, 0), 1: (1/3, 0)}
```

merge_atoms_with_same_translation()

Return a new partition into convex polyhedrons where atoms mapped by the same translation are merged if their union is convex.

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition, PolyhedronExchangeTransformation
sage: h = 1/3
sage: p = Polyhedron([(0,0),(h,0),(h,h),(0,h)])
sage: q = Polyhedron([(0,h),(h,h),(h,1),(0,1)])
sage: r = Polyhedron([(1,0),(h,0),(h,1),(1,1)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r})
sage: d = {0:(1-h,0), 1:(1-h,0), 2:(-h,0)}
sage: T = PolyhedronExchangeTransformation(P, d)
sage: T
Polyhedron Exchange Transformation of
Polyhedron partition of 3 atoms with 3 letters
with translations {0: (2/3, 0), 1: (2/3, 0), 2: (-1/3, 0)}
sage: T.merge_atoms_with_same_translation()
Polyhedron Exchange Transformation of
Polyhedron partition of 2 atoms with 2 letters
with translations {0: (2/3, 0), 2: (-1/3, 0)}
```

partition()

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition, PolyhedronExchangeTransformation
sage: h = 1/3
sage: p = Polyhedron([(0,0),(h,0),(h,1),(0,1)])
sage: q = Polyhedron([(1,0),(h,0),(h,1),(1,1)])
sage: P = PolyhedronPartition({0:p, 1:q})
sage: d = {0:(1-h,0), 1:(-h,0)}
sage: T = PolyhedronExchangeTransformation(P, d)
sage: T.partition()
Polyhedron partition of 2 atoms with 2 letters
```

plot()

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition, PolyhedronExchangeTransformation
sage: h = 1/3
sage: p = Polyhedron([(0,0),(h,0),(h,1),(0,1)])
sage: q = Polyhedron([(1,0),(h,0),(h,1),(1,1)])
sage: P = PolyhedronPartition({0:p, 1:q})
sage: d = {0:(1-h,0), 1:(-h,0)}
sage: T = PolyhedronExchangeTransformation(P, d)
sage: T.plot()
Graphics object consisting of 16 graphics primitives
```

classmethod toral_translation(*base*, *translation*, *fundamental_domain=None*)

Return a polyhedron exchange transformation defined by a translation on a d-dimensional torus.

INPUT:

- `base` – matrix, the columns are the base of a lattice

- `translation` – vector, translation vector
- `fundamental_domain` – polyhedron or `None` (default: `None`), if `None` the parallelotope defined by `base` is used.

OUTPUT:

a polyhedron exchange transformation on the fundamental domain of the lattice

EXAMPLES:

```
sage: from slabbe import PolyhedronExchangeTransformation as PET
sage: base = diagonal_matrix((1,1))
sage: translation = vector((1/5, 1/3))
sage: T = PET.toral_translation(base, translation)
sage: T
Polyhedron Exchange Transformation of
Polyhedron partition of 4 atoms with 4 letters
with translations {0: (1/5, 1/3), 1: (1/5, -2/3), 2: (-4/5, 1/3), 3: (-4/5, -2/3)}
sage: T.partition()
Polyhedron partition of 4 atoms with 4 letters
```

Some preliminary definitions:

```
sage: z = polygen(QQ, 'z') #z = QQ['z'].0 # same as
sage: K = NumberField(z**2-z-1, 'phi', embedding=RR(1.6))
sage: phi = K.gen()
sage: vertices = ((-phi + 2, phi - 1), (-phi + 2, 1), (phi - 1, 1))
sage: p = Polyhedron(vertices, base_ring=K)
```

A translation +1 modulo phi on the x coordinate:

```
sage: base = diagonal_matrix((phi,phi))
sage: translation = vector((1, 0))
sage: t0 = PET.toral_translation(base, translation)
sage: t0
Polyhedron Exchange Transformation of
Polyhedron partition of 2 atoms with 2 letters
with translations {0: (1, 0), 1: (-phi + 1, 0)}
sage: t0(p).vertices()
(A vertex at (-phi + 3, phi - 1),
 A vertex at (-phi + 3, 1),
 A vertex at (phi, 1))
```

The inverse map:

```
sage: t0.inverse()
Polyhedron Exchange Transformation of
Polyhedron partition of 2 atoms with 2 letters
with translations {0: (-1, 0), 1: (phi - 1, 0)}
sage: t0(p) == p
False
sage: t0.inverse()(t0(p)) == p
True
```

A rotation modulo 1 on the y coordinate:

```
sage: base = diagonal_matrix((phi,phi))
sage: translation = vector((0, 1))
sage: t1 = PET.toral_translation(base, translation)
sage: t1(p).vertices()
(A vertex at (-phi + 2, 0),
 A vertex at (-phi + 2, -phi + 2),
 A vertex at (phi - 1, -phi + 2))
```

It works if the translation is larger than the fundamental domain:

```
sage: base = diagonal_matrix((1,1))
sage: translation = vector((phi, 0))
sage: t2 = PET.toral_translation(base, translation)
sage: t2(p).vertices()
(A vertex at (0, phi - 1),
 A vertex at (0, 1),
 A vertex at (2*phi - 3, 1))
```

The domain is the fundamental domain of the given lattice:

```
sage: base = diagonal_matrix((phi^-2,1))
sage: translation = vector((phi^-3, 0))
sage: t3 = PET.toral_translation(base, translation)
sage: t3.domain().vertices()
(A vertex at (-phi + 2, 0),
 A vertex at (-phi + 2, 1),
 A vertex at (0, 0),
 A vertex at (0, 1))
```

The fundamental domain can be given as input. For example, it can be a translated copy of the base parallelotope:

```
sage: base = diagonal_matrix((1,1))
sage: translation = vector((1/5, 1/3))
sage: F = polytopes.parallelotope(base)
sage: T = PET.toral_translation(base, translation, F-vector((1/10,1/10)))
```

But it does not always work well yet, for example for other shape of fundamental domains:

```
sage: m = matrix(2, (1,1,0,1))
sage: T = PET.toral_translation(base, translation, m*F)
Traceback (most recent call last):
...
NotImplementedError: Volume of the partition is 41/45 but the
fundamental domain as volume 1. The code does not handle this
case properly yet.
```

translations()
EXAMPLES:

```
sage: from slabbe import PolyhedronPartition, PolyhedronExchangeTransformation
sage: h = 1/3
sage: p = Polyhedron([(0,0),(h,0),(h,1),(0,1)])
sage: q = Polyhedron([(1,0),(h,0),(h,1),(1,1)])
sage: P = PolyhedronPartition({0:p, 1:q})
sage: T = {0:(1-h,0), 1:(-h,0)}
sage: F = PolyhedronExchangeTransformation(P, T)
sage: F.translations()
{0: (2/3, 0), 1: (-1/3, 0)}
```

class `slabbe.polyhedron_partition.`**PolyhedronPartition**(*atoms*, *base_ring=None*)
Bases: `object`

Return a partition into polyhedron.

Note: Many atoms may share the same key.

INPUT:

- `atoms` – list of polyhedron or dict of key -> polyhedron or list of (key, polyhedron)
- `base_ring` – base ring (default: `None`) of the vertices

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: P
Polyhedron partition of 3 atoms with 3 letters
```

```
sage: P.is_pairwise_disjoint()
True
sage: P.volume()
1
sage: G = P.plot()
```

From a dict:

```
sage: PolyhedronPartition(dict(a=p,b=q,c=r))
Polyhedron partition of 3 atoms with 3 letters
```

From a list of (key, polyhedron):

```
sage: PolyhedronPartition([(9,p),(8,q),(9,r)])
Polyhedron partition of 3 atoms with 2 letters
```

alphabet()

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([(3,p), (5,q), (9,r)])
sage: P.alphabet()
{3, 5, 9}
sage: P = PolyhedronPartition([(3,p), (5,q), (3,r)])
sage: P.alphabet()
{3, 5}
```

alphabet_size()

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([(3,p), (5,q), (9,r)])
sage: P.alphabet_size()
3
sage: P = PolyhedronPartition([(3,p), (5,q), (3,r)])
sage: P.alphabet_size()
2
```

ambient_space()

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
```

(continues on next page)

(continued from previous page)

```
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: P.ambient_space()
Vector space of dimension 2 over Rational Field
```

atoms()

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: P.atoms()
[A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 3 vertices,
 A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 6 vertices,
 A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 3 vertices]
```

base_ring()

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: P.base_ring()
Rational Field
```

cached_atoms_set()

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: P.cached_atoms_set()
{A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 3 vertices,
 A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 3 vertices,
 A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 6 vertices}
```

code(*p*)

Returns in which atom the polyhedron lives in.

INPUT:

- p – a polyhedron

OUTPUT:

integer (for the i-th atom)

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/3
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (h,0)])
sage: r = Polyhedron([(h,1), (1,1), (1,h), (h,0)])
```

(continues on next page)

(continued from previous page)

```
sage: s = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s})
sage: P.code(p)
0
sage: P.code(q)
1
sage: t = Polyhedron([(0, 8/9), (0, 1), (1/9, 1)])
sage: P.code(t)
0
```

TESTS:

```
sage: t = Polyhedron([(0, 1/9), (0, 1), (1/9, 1)])
sage: P.code(t)
Traceback (most recent call last):
...
ValueError: polyhedron p whose vertices are (A vertex at (0,
1), A vertex at (0, 1/9), A vertex at (1/9, 1)) lies in no atom
```

domain()

Return the domain of the partition.

OUTPUT:

a polyhedron

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/3
sage: p = Polyhedron([(0,0),(h,0),(h,1),(0,1)])
sage: q = Polyhedron([(1,0),(h,0),(h,1),(1,1)])
sage: P = PolyhedronPartition({0:p, 1:q})
sage: P.domain()
A 2-dimensional polyhedron in QQ^2 defined as the convex hull of 4 vertices
sage: P.domain().vertices()
(A vertex at (0, 0),
 A vertex at (0, 1),
 A vertex at (1, 0),
 A vertex at (1, 1))
```

edges()

Return the edges of partition (one copy of each edge).

Note: If there are vertices of atoms on the interior of the edge of another atom, then, the overlapping edges will be repeated.

OUTPUT:

- set of sorted pair of immutable vectors

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: sorted(P.edges())
[((0, 0), (0, 1/2)),
```

(continues on next page)

(continued from previous page)

```
 ((0, 0), (1/2, 0)),
 ((0, 1/2), (0, 1)),
 ((0, 1/2), (1/2, 1)),
 ((0, 1), (1/2, 1)),
 ((1/2, 0), (1, 0)),
 ((1/2, 0), (1, 1/2)),
 ((1/2, 1), (1, 1)),
 ((1, 0), (1, 1/2)),
 ((1, 1/2), (1, 1))]
```

Irrational partition:

```
sage: z = polygen(QQ, 'z') #z = QQ['z'].0 # same as
sage: K = NumberField(z**2-z-1, 'phi', embedding=RR(1.6))
sage: phi = K.gen()
sage: h = 1/phi^2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (h,0)])
sage: r = Polyhedron([(h,1), (1,1), (1,h), (h,0)])
sage: s = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s}, base_ring=K)
sage: sorted(P.edges())
[((0, 0), (0, -phi + 2)),
 ((0, 0), (-phi + 2, 0)),
 ((0, -phi + 2), (0, 1)),
 ((0, -phi + 2), (-phi + 2, 1)),
 ((0, 1), (-phi + 2, 1)),
 ((-phi + 2, 0), (-phi + 2, 1)),
 ((-phi + 2, 0), (1, 0)),
 ((-phi + 2, 0), (1, -phi + 2)),
 ((-phi + 2, 1), (1, 1)),
 ((1, 0), (1, -phi + 2)),
 ((1, -phi + 2), (1, 1))]
```

is_pairwise_disjoint()

Return whether atoms of the partition are pairwise disjoint.

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: P.is_pairwise_disjoint()
True
```

keys_permutation(*other*)

Return a relabelling permutation of the keys for self to look like other.

Note: currently, the code works only if the coding of self and other is injective, i.e., no two polyhedron are coded by the same letter.

INPUT:

- `other` – a polyhedron partition (with injective coding)

OUTPUT:

dict, key -> key

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({4:p, 1:q, 2:r})
sage: Q = PolyhedronPartition({0:p, 5:q})
sage: d = P.keys_permutation(Q)
sage: d
{1: 5, 2: 1, 4: 0}
sage: P.rename_keys(d)
Polyhedron partition of 3 atoms with 3 letters
```

keys_permutation_lexicographic()

Return a permutation relabelling of the keys for self in increasing order for the lexicographic order of the centers of the polyhedrons.

OUTPUT:

dict, key -> key

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({4:p, 1:q, 2:r})
sage: d = P.keys_permutation_lexicographic()
sage: d
{1: 1, 2: 2, 4: 0}
sage: P.rename_keys(d)
Polyhedron partition of 3 atoms with 3 letters
```

```
sage: Q = PolyhedronPartition({0:p, 5:q})
sage: Q.keys_permutation_lexicographic()
{0: 0, 5: 1}
```

It works when the partition has two atoms coded by the same key:

```
sage: P = PolyhedronPartition([(0,p), (0,q), (3,r)])
sage: d = P.keys_permutation_lexicographic()
sage: d
{0: 0, 3: 1}
sage: P.rename_keys(d).alphabet()
{0, 1}
```

merge_atoms(*d*)

Return the polyhedron partition obtained by merging atoms having the same image under the dictionnary.

INPUT:

- d – dict

OUTPUT:

a polyhedron partition

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
```

(continues on next page)

(continued from previous page)

```
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r})
sage: P.merge_atoms({0:4, 1:4, 2:5})
Polyhedron partition of 2 atoms with 2 letters
sage: P.merge_atoms({0:4, 1:5, 2:4})
Polyhedron partition of 3 atoms with 2 letters
```

When pair of atoms are not convex, it needs to merge 3 or more atoms:

```
sage: h = 1/5
sage: p = Polyhedron([(0,0),(h,1-h),(0,1)])
sage: q = Polyhedron([(0,1), (h,1-h), (1,1)])
sage: r = Polyhedron([(0,0), (h,1-h), (1,1), (1,0)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r})
sage: P.merge_atoms({0:4, 1:4, 2:4})
Polyhedron partition of 1 atoms with 1 letters
```

plot()

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: P.plot()
Graphics object consisting of 21 graphics primitives
```

refine_by_hyperplane(*ieq*)

Refine the partition with the two half spaces of each side of an hyperplane.

INPUT:

- ieq – list, an inequality. An entry equal to "[-1,7,3,4]" represents the inequality 7x_1+3x_2+4x_3>= 1.

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/3
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (h,0)])
sage: r = Polyhedron([(h,1), (1,1), (1,h), (h,0)])
sage: s = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s})
sage: ieq = [-4, 5, 1]
sage: P.refine_by_hyperplane(ieq)
Polyhedron partition of 6 atoms with 6 letters
```

refinement(*other*, *key_fn=None*)

Return the polyhedron partition obtained by the intersection of the atoms of self with the atoms of other.

Only atoms of positive volume are kept.

INPUT:

- other – a polyhedron partition
- key_fn – function to apply on pairs of labels, or None

OUTPUT:

a polyhedron partition

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/3
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (h,0)])
sage: r = Polyhedron([(h,1), (1,1), (1,h), (h,0)])
sage: s = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s})
sage: g = 1/5
sage: t1 = Polyhedron([(g,g), (g,1-g), (1-g,g) ])
sage: t2 = Polyhedron([(g,1-g), (1-g,g), (1-g,1-g)])
sage: Q = PolyhedronPartition([t1,t2])
sage: P.refinement(Q)
Polyhedron partition of 8 atoms with 8 letters
```

rename_keys(*d*)

Return a polyhedron partition whose keys are the images under a map.

INPUT:

- d – dict, function old key -> new key

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: Q = P.rename_keys({0:'b', 1:'a', 2:'z'})
sage: Q
Polyhedron partition of 3 atoms with 3 letters
sage: sorted(key for key,p in Q)
['a', 'b', 'z']
```

It does not have to be injective:

```
sage: Q = P.rename_keys({0:'b', 1:'a', 2:'b'})
sage: sorted(key for key,p in Q)
['a', 'b', 'b']
```

tikz(*fontsize='\\normalsize', scale=1, label_format='{}', extra_code=''*)

INPUT:

- `fontsize` – string (default: `r'\normalsize'`
- `scale` – number (default: 1)
- `label_format` – string (default: `r'{}'`) to be called with `label_format.format(key)`
- `extra_code` – string (default: `''`)

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: _ = P.tikz().pdf(view=False)
```

Irrational partition:

```
sage: z = polygen(QQ, 'z') #z = QQ['z'].0 # same as
sage: K = NumberField(z**2-z-1, 'phi', embedding=RR(1.6))
sage: phi = K.gen()
sage: h = 1/phi^2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (h,0)])
sage: r = Polyhedron([(h,1), (1,1), (1,h), (h,0)])
sage: s = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition({0:p, 1:q, 2:r, 3:s}, base_ring=K)
sage: _ = P.tikz().pdf(view=False)
```

Testing the options:

```
sage: _ = P.tikz(fontsize=r'\scriptsize').pdf(view=False)
sage: _ = P.tikz(scale=2).pdf(view=False)
sage: _ = P.tikz(label_format=r'$a_{{{}}}$').pdf(view=False)
```

translation(*displacement*)

Return the translated partition of polyhedron.

INPUT:

- `displacement` – a displacement vector or a list/tuple of coordinates that determines a displacement vector.

OUTPUT:

The translated partition.

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: P.translation((1,1))
Polyhedron partition of 3 atoms with 3 letters
```

volume()

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: P.volume()
1
```

TESTS:

```
sage: PolyhedronPartition([], base_ring=ZZ).volume()
0
```

volume_dict(*normalize=False*)

INPUT

- `normalize` – boolean (default:`False`), whether to normalize the sum of the whole volume to 1

EXAMPLES:

```
sage: from slabbe import PolyhedronPartition
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: P = PolyhedronPartition([p,q,r])
sage: P.volume_dict()
{0: 1/8, 1: 3/4, 2: 1/8}
sage: (2*P).volume_dict()
{0: 1/2, 1: 3, 2: 1/2}
```

`slabbe.polyhedron_partition.`**`find_unused_key`**(*d*, *sequence*)

Return the first key in sequence which is not in d.

EXAMPLES:

```
sage: from slabbe.polyhedron_partition import find_unused_key
sage: d = {3:32, 0:21, 1:4, 5:5}
sage: find_unused_key(d, NN)
2
sage: d[2] = 1234
sage: find_unused_key(d, NN)
4
sage: d[4] = 1234
sage: find_unused_key(d, NN)
6
```

`slabbe.polyhedron_partition.`**`is_union_convex`**(*t*)

Return whether the union of the polyhedrons is convex.

INPUT:

- `t` – list of polyhedron

EXAMPLES:

```
sage: from slabbe.polyhedron_partition import is_union_convex
sage: h = 1/2
sage: p = Polyhedron([(0,h),(0,1),(h,1)])
sage: q = Polyhedron([(0,0), (0,h), (h,1), (1,1), (1,h), (h,0)])
sage: r = Polyhedron([(h,0), (1,0), (1,h)])
sage: is_union_convex((p,q,r))
True
sage: is_union_convex((p,q))
True
sage: is_union_convex((p,r))
False
```

Here we need to consider the three at the same time to get a convex union:

```
sage: h = 1/5
sage: p = Polyhedron([(0,0),(h,1-h),(0,1)])
sage: q = Polyhedron([(0,1), (h,1-h), (1,1)])
sage: r = Polyhedron([(0,0), (h,1-h), (1,1), (1,0)])
sage: is_union_convex((p,q))
False
sage: is_union_convex((p,r))
False
sage: is_union_convex((q,r))
False
sage: is_union_convex((p,q,r))
True
```

4.8 2d Substitutions

2d substitutions

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[0,1],[2,3]]
sage: B = [[4,5]]
sage: d = {0:A, 1:B}
sage: Substitution2d(d)
Substitution 2d: {0: [[0, 1], [2, 3]], 1: [[4, 5]]}
```

class slabbe.substitution_2d.**Substitution2d**(*d*)

Bases: `object`

INPUT:

- `d` – dict, key -> value, where each value is a table such that table[x][y] refers to the tile at position (x,y) in cartesian coordinates (*not* in the matrix-like coordinates)

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[0,1],[2,3]]
sage: B = [[4,5]]
sage: d = {0:A, 1:B}
sage: s = Substitution2d(d)
sage: s
Substitution 2d: {0: [[0, 1], [2, 3]], 1: [[4, 5]]}
```

apply_matrix_transformation(*M*)

INPUT:

- `M` – matrix in SL(2,Z)

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[0,1],[0,1]]
sage: B = [[1,0],[1,1]]
sage: d = {0:A, 1:B}
sage: s = Substitution2d(d)
sage: M = matrix(2, (1,1,0,1))
sage: s
Substitution 2d: {0: [[0, 1], [0, 1]], 1: [[1, 0], [1, 1]]}
sage: s.apply_matrix_transformation(M)
Substitution 2d: {0: [[0, None], [0, 1], [None, 1]], 1: [[1, None], [1, 0], [None, 1]]}
```

call_on_column(*column*, *heights=None*)

INPUT:

- `column` – list
- `heights` – None or list (default: `None`)

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[0,1],[2,3]]
sage: B = [[4],[5]]
sage: C = [[6,7,8]]
sage: d = {0:A, 1:B, 2:C}
sage: s = Substitution2d(d)
```

(continues on next page)

(continued from previous page)

```
sage: s.call_on_column([0])
[[0, 1], [2, 3]]
sage: s.call_on_column([0,1])
[[0, 1, 4], [2, 3, 5]]
sage: s.call_on_column([0,1,1,0,0])
[[0, 1, 4, 4, 0, 1, 0, 1], [2, 3, 5, 5, 2, 3, 2, 3]]
```

It can compute the image of columns with `None` as entries:

```
sage: s.call_on_column([0,None], heights=[2,3])
[[0, 1, None, None, None], [2, 3, None, None, None]]
sage: s.call_on_column([0,None], heights=[2,2])
[[0, 1, None, None], [2, 3, None, None]]
sage: s.call_on_column([None], heights=[3])
[[None, None, None]]
```

TESTS:

```
sage: s.call_on_column([])
[]
sage: s.call_on_column([0,2])
Traceback (most recent call last):
...
ValueError: the image of 2 in the column (=[0, 2]) has width 1
but the image of another has width 2
```

call_on_row(*row*)

INPUT:

- `row` – list

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[0,1],[2,3]]
sage: B = [[4,5]]
sage: C = [[6,7,8]]
sage: d = {0:A, 1:B, 2:C}
sage: s = Substitution2d(d)
sage: row = [0,1,1,0]
sage: s.call_on_row(row)
[[0, 1], [2, 3], [4, 5], [4, 5], [0, 1], [2, 3]]
sage: s.call_on_row([2])
[[6, 7, 8]]
```

TESTS:

```
sage: s.call_on_row([])
[]
sage: s.call_on_row([1,2])
Traceback (most recent call last):
...
ValueError: the image of the row contains columns of different height (=set([2, 3]))
```

codomain_alphabet()

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[5,6],[7,8]]
sage: B = [[6,5],[9,8]]
sage: d = {0:A, 1:B}
sage: s = Substitution2d(d)
```

(continues on next page)

(continued from previous page)

```
sage: s.codomain_alphabet()
{5, 6, 7, 8, 9}
```

Blank `None` are ignored:

```
sage: A = [[5,6],[7,8]]
sage: B = [[6,5],[9,None]]
sage: d = {0:A, 1:B}
sage: s = Substitution2d(d)
sage: s.codomain_alphabet()
{5, 6, 7, 8, 9}
```

desubstitute(*tiles*, *function=None*)

Return the Wang tile set obtained from the desubstitution of the given Wang tile set.

INPUT:

- `tiles` – list of Wang tiles, each tile being a 4-tuple of (east, north, west, south) colors
- `fn` – a function (default: `None`) to apply to the new colors which are tuple of previous colors

OUTPUT:

dict, key -> tile

domain_alphabet()

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[5,6],[7,8]]
sage: B = [[6,5],[9,8]]
sage: d = {0:A, 1:B}
sage: s = Substitution2d(d)
sage: s.domain_alphabet()
{0, 1}
```

classmethod from_1d_column_substitution(*s*)

INPUT:

- s – dict

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: fibo = {0:[0,1], 1:[0]}
sage: s = Substitution2d.from_1d_column_substitution(fibo)
sage: s
Substitution 2d: {0: [[0, 1]], 1: [[0]]}
```

classmethod from_1d_row_substitution(*s*)

INPUT:

- s – dict

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: fibo = {0:[0,1], 1:[0]}
sage: s = Substitution2d.from_1d_row_substitution(fibo)
sage: s
Substitution 2d: {0: [[0], [1]], 1: [[0]]}
```

classmethod from_permutation(*d*)

INPUT:

- d – dict

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: s = Substitution2d.from_permutation({4:0, 5:1})
sage: s
Substitution 2d: {4: [[0]], 5: [[1]]}
```

```
sage: A = [[5,6],[7,8]]
sage: B = [[6,5],[9,8]]
sage: t = Substitution2d({0:A, 1:B})
sage: t
Substitution 2d: {0: [[5, 6], [7, 8]], 1: [[6, 5], [9, 8]]}
sage: t*s
Substitution 2d: {4: [[5, 6], [7, 8]], 5: [[6, 5], [9, 8]]}
```

```
sage: u = Substitution2d.from_permutation({5:0, 6:1, 7:2, 8:3, 9:4})
sage: u
Substitution 2d: {8: [[3]], 9: [[4]], 5: [[0]], 6: [[1]], 7: [[2]]}
sage: u * t
Substitution 2d: {0: [[0, 1], [2, 3]], 1: [[1, 0], [4, 3]]}
```

incidence_matrix()

Return the incidence matrix of self.

Some default ordering (sorted) is used for the domain and codomain alphabet.

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[0,1],[2,3]]
sage: B = [[4,5]]
sage: C = [[6,7,8]]
sage: d = {0:A, 1:B, 2:C}
sage: s = Substitution2d(d)
sage: s.incidence_matrix()
[1 0 0]
[1 0 0]
[1 0 0]
[1 0 0]
[0 1 0]
[0 1 0]
[0 0 1]
[0 0 1]
[0 0 1]
```

inverse()

Return the inverse of self (when self is a permutation).

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: d = {0:7, 1:8}
sage: s = Substitution2d.from_permutation(d)
sage: s
Substitution 2d: {0: [[7]], 1: [[8]]}
sage: s.inverse()
Substitution 2d: {8: [[1]], 7: [[0]]}
```

TESTS:

```
sage: s = Substitution2d({8: [[1]], 7: [[0,1]]})
sage: s.inverse()
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
...
ValueError: self must be a permutation but image of 7 is [[0, 1]]
```

letter_to_letter_dict(*pos=(0, 0)*)

Return the inverse of self (when self is a permutation).

INPUT:

- `pos` – tuple (default:`(0,0)`), tuple of two integers

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[0,1],[2,3]]
sage: B = [[4,5]]
sage: s = Substitution2d({0:A, 1:B})
sage: s
Substitution 2d: {0: [[0, 1], [2, 3]], 1: [[4, 5]]}
sage: s.letter_to_letter_dict(pos=(0,0))
{0: 0, 1: 4}
```

lines_alphabet(*direction='horizontal'*)

Return the possible alphabets on lines, i.e., the possible alphabet of letters that we see on a given line.

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[0,1],[0,1]]
sage: B = [[1,0],[1,1]]
sage: d = {0:A, 1:B}
sage: s = Substitution2d(d)
sage: sorted(s.lines_alphabet())
[(0,), (0, 1), (1,)]
sage: sorted(s.lines_alphabet(direction='vertical'))
[(0, 1), (1,)]
```

list_2x2_factors(*F=None*)

Return the list of 2x2 factors in the associated substitutive shift. If a list of factors `F` is given, it restrict to the factors inside the image of `F`.

INPUT:

- `self` – expansive and primitive 2d substitution
- `F` – list of factors in the domain or `None`, if given the output is restricted to the factors in `F`

OUTPUT:

list of tables

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[0,1],[0,1]]
sage: B = [[1,0],[1,1]]
sage: d = {0:A, 1:B}
sage: s = Substitution2d(d)
sage: s.list_2x2_factors()
[[[0, 1], [0, 1]],
 [[1, 0], [1, 1]],
 [[1, 1], [1, 0]],
 [[1, 1], [1, 1]],
 [[1, 1], [0, 1]],
```

(continues on next page)

(continued from previous page)

```
 [[1, 1], [0, 0]],
 [[0, 1], [1, 1]],
 [[1, 0], [0, 1]],
 [[0, 0], [1, 0]],
 [[0, 1], [1, 0]],
 [[1, 0], [1, 0]],
 [[1, 0], [0, 0]]]
```

Restricting to the images of some factors:

```
sage: s.list_2x2_factors([A])
[[[1, 0], [1, 1]], [[1, 1], [1, 0]], [[1, 1], [1, 1]], [[0, 1], [0, 1]]]
sage: s.list_2x2_factors([B])
[[[1, 0], [1, 1]],
 [[0, 1], [1, 0]],
 [[1, 1], [1, 0]],
 [[0, 1], [0, 1]],
 [[0, 1], [1, 1]],
 [[1, 0], [0, 1]],
 [[0, 0], [1, 0]]]
sage: s.list_2x2_factors([A,B])
[[[1, 0], [1, 1]],
 [[1, 1], [1, 0]],
 [[0, 1], [1, 1]],
 [[1, 1], [1, 1]],
 [[0, 0], [1, 0]],
 [[1, 0], [0, 1]],
 [[0, 1], [1, 0]],
 [[0, 1], [0, 1]]]
sage: s.list_2x2_factors([])
[]
```

prolongable_origins()

relabel_domain(*other*)

Return a permutation p such that self*p == other, if it exists.

INPUT:

- `other` – substitution 2d

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[0,1],[0,1]]
sage: B = [[1,0],[1,1]]
sage: s = Substitution2d({0:A, 1:B})
sage: t = Substitution2d({7:A, 8:B})
sage: s.relabel_domain(t)
Substitution 2d: {8: [[1]], 7: [[0]]}
```

TESTS:

```
sage: s = Substitution2d({0:A, 1:B})
sage: s.relabel_domain(s)
Substitution 2d: {0: [[0]], 1: [[1]]}
```

```
sage: s = Substitution2d({0:A, 1:B})
sage: t = Substitution2d({7:A, 8:B, 9:[[4]]})
sage: t.relabel_domain(s)
Traceback (most recent call last):
...
ValueError: image of letter 9 is [[4]] and is not in other
```

(continues on next page)

(continued from previous page)

```
sage: s.relabel_domain(t)
Traceback (most recent call last):
...
AssertionError: problem: self * p == other not satisfied
```

reversal()

Return the reversal of self.

EXAMPLES:

```
sage: from slabbe import Substitution2d
sage: A = [[1,2],[3,4]]
sage: B = [[5,6],[7,8]]
sage: d = {0:A, 1:B}
sage: s = Substitution2d(d)
sage: s.reversal()
Substitution 2d: {0: [[4, 3], [2, 1]], 1: [[8, 7], [6, 5]]}
```

wang_tikz(*domain_tiles*, *codomain_tiles*, *domain_color=None*, *codomain_color=None*, *size=1*, *scale=1*, *font='\\normalsize'*, *rotate=None*, *label_shift=0.2*, *id=True*, *edges=True*, *ncolumns=4*, *direction='right'*, *extra_space=1*)

Return the tikz code showing what the substitution A->B* does on Wang tiles.

INPUT:

- `domain_tiles` – tiles of the domain
- `codomain_tiles` – tiles of the codomain
- `domain_color` – dict (default: `None`) from tile values -> tikz colors
- `codomain_color` – dict (default: `None`) from tile values -> tikz colors
- `size` – number (default: 1), size of the tile
- `scale` – number (default: 1), scale of tikzpicture
- `font` – string (default: `r'\normalsize'`
- `rotate` – list or `None` (default:`None`) list of four angles in degrees like `(0,0,0,0)`, the rotation angle to apply to each label of Wang tiles. If `None`, it performs a 90 degres rotation for left and right labels taking more than one character.
- `label_shift` – number (default: `.2`) translation distance of the label from the edge
- `id` – boolean (default: `True`), presence of the tile id
- `ncolumns` – integer (default: 4)
- `edges` – bool (default: `True`)
- `direction` – string (default: `'right'`) or `'down'`
- `extra_space` – number (default: 1), space between the tile and its image

OUTPUT:

dict, key -> tile

EXAMPLES:

```
sage: from slabbe import WangTileSet, Substitution2d
sage: A = [[0,1,2],[1,0,0]]
sage: B = [[0,1,2]]
sage: d = {4:A, 5:B}
```

(continues on next page)

(continued from previous page)

```
sage: s = Substitution2d(d)
sage: codomain_tiles = [(0,3,1,4), (1,4,0,3), (5,6,7,8)]
sage: W = WangTileSet(codomain_tiles)
sage: fn = lambda colors:''.join(map(str, colors))
sage: domain_tiles = W.desubstitute(s, fn)
sage: tikz = s.wang_tikz(domain_tiles, codomain_tiles, rotate=(90,0,90,0))
sage: _ = tikz.pdf(view=False)      # long time
```

Applying a transformation matrix:

```
sage: M = matrix(2, [1,1,0,1])
sage: sM = s.apply_matrix_transformation(M)
sage: tikz = sM.wang_tikz(domain_tiles, codomain_tiles)
sage: _ = tikz.pdf(view=False)      # long time
```

Down direction:

```
sage: tikz = s.wang_tikz(domain_tiles, codomain_tiles,
....:                         direction='down')
sage: _ = tikz.pdf(view=False)      # long time
```

wang_tiles_codomain_tikz(*codomain_tiles, color=None, size=1, scale=1, font='\\normalsize', rotate=None, id=True, label=True, label_shift=0.2, edges=True, ncolumns=4, direction='right'*)

Return the tikz code of the image of the letters as a table of tikz tilings.

INPUT:

- `domain_tiles` – tiles of the domain
- `codomain_tiles` – tiles of the codomain
- `domain_color` – dict (default: `None`) from tile values -> tikz colors
- `codomain_color` – dict (default: `None`) from tile values -> tikz colors
- `size` – number (default: `1`), size of the tile
- `scale` – number (default: `1`), scale of tikzpicture
- `font` – string (default: `r'\normalsize'`
- `rotate` – list or `None` (default:`None`) list of four angles in degrees like `(0,0,0,0)`, the rotation angle to apply to each label of Wang tiles. If `None`, it performs a 90 degres rotation for left and right labels taking more than one character.
- `id` – boolean (default: `True`), presence of the tile id
- `label` – boolean (default: `True`)
- `label_shift` – number (default: `.2`) translation distance of the label from the edge
- `edges` – bool (default: `True`)
- `ncolumns` – integer (default: `4`)

OUTPUT:

tikzpicture

EXAMPLES:

```
sage: from slabbe import WangTileSet, Substitution2d
sage: A = [[0,1,2],[1,0,0]]
sage: B = [[0,1,2]]
sage: d = {4:A, 5:B}
sage: s = Substitution2d(d)
sage: codomain_tiles = [(0,3,1,4), (1,4,0,3), (5,6,7,8)]
sage: W = WangTileSet(codomain_tiles)
sage: t = s.wang_tiles_codomain_tikz(W)
sage: _ = t.pdf(view=False)
```

`slabbe.substitution_2d.`**`set_of_factors`**(*table*, *shape*, *avoid_border=0*)

Return the set of factors of given shape in the table.

INPUT

- `table` – list of lists
- `shape` – list, list of coordinates
- **`avoid_border` – integer (default: 0), the size of the border** to avoid during the computation

OUTPUT:

set of tuple of integers

EXAMPLES:

```
sage: from slabbe.substitution_2d import set_of_factors
sage: table = [[0,1,2], [3,4,5], [6,7,8]]
sage: set_of_factors(table, shape=[(0,0), (1,0), (0,1), (1,1)])
{(0, 3, 1, 4), (1, 4, 2, 5), (3, 6, 4, 7), (4, 7, 5, 8)}
```

4.9 Wang tile, tilings and solver

Wang tile solver

This uses MILP solvers like Coin or Gurobi, Sat solvers like cryptominisat and dancing links solver which is already in Sage.

Coin can be installed with:

```
sage -i cbc sagelib
```

Cryptominisat can be installed with:

```
sage -i cryptominisat sagelib
```

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve()
sage: _ = tiling.tikz().pdf(view=False)
```

Using different kind of solvers:

```
sage: tiling = W.solve(solver='GLPK')
sage: tiling = W.solve(solver='dancing_links')
sage: tiling = W.solve(solver='Gurobi')         # optional Gurobi
sage: tiling = W.solve(solver='cryptominisat')  # optional cryptominisat
```

```
sage: tiles = [(1/2,1/2,1/2,1/2), (1,1,1,1), (2,2,2,2)]
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve()
sage: _ = tiling.tikz().pdf(view=False)
```

```
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve()
sage: tiling._table
[[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
```

Kari-Culik (Here 0' is replaced by 10):

```
sage: divide_by_2 = [(10,0,10,10), (10,1,10,2), (1/2,0,10,1), (1/2,10,10,1),
....:     (1/2,0,1/2,10), (1/2,1,1/2,2), (10,1,1/2,1)]
sage: times_3 = [(1,2,0,1), (2,1,0,1), (2,2,1,1), (0,1,1,0), (0,2,2,0),
....:     (1,1,2,0)]
sage: tiles = divide_by_2 + times_3
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve()
sage: _ = tiling.tikz().pdf(view=False)
```

Rao-Jeandel:

```
sage: tiles = [(2,4,2,1), (2,2,2,0), (1,1,3,1), (1,2,3,2), (3,1,3,3),
....: (0,1,3,1), (0,0,0,1), (3,1,0,2), (0,2,1,2), (1,2,1,4), (3,3,1,2)]
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve()
sage: _ = tiling.tikz().pdf(view=False)
```

class `slabbe.wang_tiles.`**`WangTileSet`**(*tiles*)

Bases: `object`

Construct a Wang tile set.

INPUT:

- `tiles` – list of tiles, a tile is a 4-tuple (right color, top color, left color, bottom color)

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: T = WangTileSet(tiles)
```

```
sage: tiles = [(0,0,0,2), (1,0,0,1), (2,1,0,0), (0,0,1,0),
....:          (1,2,1,1), (1,1,2,0), (2,0,2,1)]
sage: T = WangTileSet(tiles)
```

`admissible_horizontal_words`(*length*, *width*, *height*)

Return the horizontal word of given length appearing in every position inside a rectangle of given width and height.

INPUT:

- `length` – integer
- `width` – integer
- `height` – integer

OUTPUT:

set of tuples

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: T = WangTileSet(tiles)
sage: T.admissible_horizontal_words(2,2,2)
{(0, 0), (1, 1), (2, 2)}
```

The horizontal word 22 is impossible after looking at large enough boxes:

```
sage: tiles = [(0,0,0,2), (1,0,0,1), (2,1,0,0), (0,0,1,0),
....:          (1,2,1,1), (1,1,2,0), (2,0,2,1)]
sage: T = WangTileSet(tiles)
sage: T.admissible_horizontal_words(2,2,2)
{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (2, 0), (2, 2)}
sage: T.admissible_horizontal_words(2,3,3)
{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (2, 0)}
sage: T.admissible_horizontal_words(2,4,4)
{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (2, 0)}
sage: T.admissible_horizontal_words(2,5,5)
{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (2, 0)}
```

admissible_vertical_words(*length*, *width*, *height*)

Return the vertical word of given length appearing in every position inside a rectangle of given width and height.

INPUT:

- `length` – integer
- `width` – integer
- `height` – integer

OUTPUT:

set of tuples

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: T = WangTileSet(tiles)
sage: T.admissible_vertical_words(2,2,2)
{(0, 0), (1, 1), (2, 2)}
```

Every word of length 2 appear as a vertical word in every position of a 5×5 box:

```
sage: tiles = [(0,0,0,2), (1,0,0,1), (2,1,0,0), (0,0,1,0),
....:          (1,2,1,1), (1,1,2,0), (2,0,2,1)]
sage: T = WangTileSet(tiles)
sage: T.admissible_vertical_words(2,2,2)
{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)}
sage: T.admissible_vertical_words(2,5,5)
{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)}
```

clean_sources_and_sinks()

TODO: do it for the dual?

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,0), (1,1,1,1), (3,2,4,8), (0,5,0,7)]
sage: T = WangTileSet(tiles)
sage: T.clean_sources_and_sinks().tiles()
```

(continues on next page)

(continued from previous page)

```
[(0, 0, 0, 0), (0, 5, 0, 7), (1, 1, 1, 1)]
sage: T.dual().clean_sources_and_sinks().tiles()
[(0, 0, 0, 0), (1, 1, 1, 1)]
```

create_macro_file(*filename='macro.tex'*, *command_name='Tile'*, *color=None*, *size=1*, *scale=1*, *font='\\normalsize'*, *label_color='black'*, *rotate=None*, *label=True*, *label_shift=0.2*, *id=True*, *id_color=''*, *id_format='{}'*, *draw_H=None*, *draw_V=None*)

INPUT:

- `filename` – string (default: `r'macro.tex'`)
- `comand_name` – string (default: `r'Tile'`)
- `color` – dict (default: None)
- `size` – number (default: 1)
- `scale` – number (default: 1)
- `font` – string (default: `r'\normalsize'`)
- `rotate` – list or `None` (default:`None`) list of four angles in degrees like `(0,0,0,0)`, the rotation angle to apply to each label of Wang tiles. If `None`, it performs a 90 degres rotation for left and right labels taking more than one character.
- `label` – boolean (default: `True`), presence of the colors
- `label_shift` – number (default: `.2`) translation distance of the label from the edge
- `label_color` – string (default: `'black'`)
- `id` – boolean (default: `True`), presence of the tile id
- `id_color` – string (default: `''`)
- `id_format` – string (default: `r'{}'`) to be called with `id_format.format(key)`
- `draw_H` – dict (default: `None`) from tile values -> tikz draw commands. If `None` the values of the dict get replaced by straight lines, more precisely by `r'\draw {{}} -- ++ (1,0);'`. Dict values must be strings `s` such that `s.format((x,y))` works.
- `draw_V` – dict (default: `None`) from tile values -> tikz draw commands. If `None` the values of the dict get replaced by straight lines, more precisely by `r'\draw {{}} -- ++ (0,1);'`. Dict values must be strings `s` such that `s.format((x,y))` works.

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,2), (1,0,0,1), (2,1,0,0), (0,0,1,0),
....:          (1,2,1,1), (1,1,2,0), (2,0,2,1)]
sage: T = WangTileSet(tiles)
sage: T.create_macro_file() # not tested
creation of file macro.tex
```

```
sage: color = {0:'white',1:'red',2:'cyan',3:'green',4:'white'}
sage: T.create_macro_file(color=color) # not tested
creation of file macro.tex
```

create_tikz_pdf_files(*prefix='tile'*, *color=None*)

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,2), (1,0,0,1), (2,1,0,0), (0,0,1,0),
....:          (1,2,1,1), (1,1,2,0), (2,0,2,1)]
sage: T = WangTileSet(tiles)
sage: T.create_tikz_pdf_files() # not tested
```

This creates tile0.pdf, tile1.pdf, etc. in the repository.

desubstitute(*substitution*, *function=None*)

Return the Wang tile set obtained from the desubstitution of the given Wang tile set.

INPUT:

- `substitution` – substitution 2d
- `fn` – a function (default: `None`) to apply to the new colors which are tuple of previous colors

OUTPUT:

dict, key -> tile

EXAMPLES:

```
sage: from slabbe import Substitution2d, WangTileSet
sage: A = [[0,1,2],[1,0,0]]
sage: B = [[0,1,2]]
sage: d = {4:A, 5:B}
sage: s = Substitution2d(d)
sage: tiles = [(0,3,1,4), (1,4,0,3), (5,6,7,8)]
sage: W = WangTileSet(tiles)
sage: W.desubstitute(s)
{4: ((1, 0, 0), (6, 3), (1, 0, 7), (4, 3)),
 5: ((0, 1, 5), (6,), (1, 0, 7), (4,))}
```

Providing a function which gets back to integers:

```
sage: fn = lambda colors:int(''.join(map(str, colors)))
sage: W.desubstitute(s, fn)
{4: (100, 63, 107, 43), 5: (15, 6, 107, 4)}
```

Providing a function which concatenate label as strings:

```
sage: fn = lambda colors:''.join(map(str, colors))
sage: W.desubstitute(s, fn)
{4: ('100', '63', '107', '43'), 5: ('015', '6', '107', '4')}
```

dominoes_with_surrounding(*i=2*, *radius=1*, *solver=None*, *ncpus=1*, *verbose=False*)

INPUT:

- `i` - integer (default: `2`), 1 or 2
- `radius` - integer or 2-tuple (default: `1`), if 2-tuple is given, then it is interpreted as `(xradius, yradius)`
- `solver` - string or None (default: `None`)
- `ncpus` – integer (default: `1`), maximal number of subprocesses to use at the same time, used only if `solver` is `'dancing_links'`.
- `verbose` - bool

Note: The `solver='dancing_links'` is fast for this question.

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = ['ABCD', 'EFGH', 'AXCY', 'ABAB', 'EBEB']
sage: T = WangTileSet(tiles)
sage: sorted(T.dominoes_with_surrounding(i=1))
[(3, 3), (4, 4)]
sage: sorted(T.dominoes_with_surrounding(i=2))
[(3, 3), (3, 4), (4, 3), (4, 4)]
sage: sorted(T.dominoes_with_surrounding(i=2, radius=2))
[(3, 3), (3, 4), (4, 3), (4, 4)]
sage: sorted(T.dominoes_with_surrounding(i=2, radius=(1,2)))
[(3, 3), (3, 4), (4, 3), (4, 4)]
```

TESTS:

```
sage: tiles = [('02', '4', '02', '4'), ('32', '4', '02', '4')]
sage: T = WangTileSet(tiles)
sage: sorted(T.dominoes_with_surrounding(1))
[(0, 0)]
sage: sorted(T.dominoes_with_surrounding(2))
[(0, 0)]
```

dual()

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,2), (1,0,0,1), (2,1,0,0), (0,0,1,0),
....:          (1,2,1,1), (1,1,2,0), (2,0,2,1)]
sage: T = WangTileSet(tiles)
sage: dual = T.dual()
sage: dual
Wang tile set of cardinality 7
sage: dual.tiles()
[(0, 0, 2, 0),
 (0, 1, 1, 0),
 (1, 2, 0, 0),
 (0, 0, 0, 1),
 (2, 1, 1, 1),
 (1, 1, 0, 2),
 (0, 2, 1, 2)]
```

find_markers(*i=2*, *radius=1*, *solver=None*, *ncpus=1*, *verbose=False*)

Return a list of lists of marker tiles.

INPUT:

- `i` – integer (default:2), 1 or 2.
- `radius` - integer or 2-tuple (default: 1), if 2-tuple is given, then it is interpreted as `(xradius, yradius)`
- `solver` – string (default:`None`)
- `ncpus` – integer (default:1)
- `verbose` – boolean (default:`False`)

Note: The `solver='dancing_links'` is fast for this question.

OUTPUT:

list of lists

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(2,4,2,1), (2,2,2,0), (1,1,3,1), (1,2,3,2), (3,1,3,3),
....: (0,1,3,1), (0,0,0,1), (3,1,0,2), (0,2,1,2), (1,2,1,4), (3,3,1,2)]
sage: tiles = [map(str,t) for t in tiles]
sage: T = WangTileSet(tiles)
sage: T.find_markers(i=1)
[]
sage: T.find_markers(i=2)
[[0, 1]]
```

```
sage: tiles = ['ABCD', 'EFGH', 'AXCY', 'ABAB']
sage: T = WangTileSet(tiles)
sage: T.find_markers(i=1)
[[0], [1], [2]]
sage: T.find_markers(i=2)
[[0], [1], [2]]
```

find_markers_with_slope(*i=2*, *slope=None*, *radius=1*, *solver=None*, *ncpus=1*, *verbose=False*)

Return a list of lists of marker tiles.

INPUT:

- `i` – integer (default:2), 1 or 2.
- `slope` – -1, 0, 1 or Infinity (default:`None`)
- `radius` - integer or 2-tuple (default: 1), if 2-tuple is given, then it is interpreted as `(xradius, yradius)`
- `solver` – string (default:`None`)
- `ncpus` – integer (default:1)
- `verbose` – boolean (default:`False`)

OUTPUT:

list of lists

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = ['ABCD', 'EFGH', 'AXCY', 'ABAB']
sage: T = WangTileSet(tiles)
sage: T.find_markers_with_slope(i=1, slope=1)   # known bug
[{0, 3}, {1}, {2}]
sage: T.find_markers_with_slope(i=2, slope=1)   # known bug
[{0, 2, 3}, {1}]
```

find_substitution(*M=None*, *i=2*, *side='right'*, *radius=1*, *solver=None*, *ncpus=1*, *function=<slot wrapper '__add__' of 'str' objects>*, *initial=''*, *verbose=False*)

Return the derived Wang tile set obtained from desubstitution using a given set of marker tiles.

INPUT:

- `M` – markers, set of tile indices
- `i` – integer 1 or 2
- `side` – `'right'` or `'left'`
- `radius` - integer or 2-tuple (default: 1), if 2-tuple is given, then it is interpreted as `(xradius, yradius)`
- `solver` – string (default:`None`)

- `ncpus` – integer (default:1)
- **`function` – function (default:`str.__add__`), monoid** operation
- `initial` – object (default:`''`), monoid neutral
- `verbose` – boolean

OUTPUT:

a 3-tuple (Wang tile set, substitution2d, set of markers)

Note: The `solver='dancing_links'` is fast for this question.

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(2,4,2,1), (2,2,2,0), (1,1,3,1), (1,2,3,2), (3,1,3,3),
....: (0,1,3,1), (0,0,0,1), (3,1,0,2), (0,2,1,2), (1,2,1,4), (3,3,1,2)]
sage: tiles = [map(str,t) for t in tiles]
sage: T = WangTileSet(tiles)
sage: T.find_markers(i=2)
[[0, 1]]
sage: T.find_substitution(M=[0,1], i=2)
(Wang tile set of cardinality 12,
 Substitution 2d: {0: [[2]], 1: [[3]], 2: [[4]], 3:
    [[5]], 4: [[7]], 5: [[8]], 6: [[9]], 7: [[10]], 8:
    [[4, 0]], 9: [[5, 0]], 10: [[6, 1]], 11: [[7, 0]]})
```

fusion(*other*, *direction*, *function=<slot wrapper '__add__' of 'str' objects>*, *initial=''*, *clean_graph=True*)

Return the fusion of wang tile sets in the given direction.

TODO: check if and when to do the clean

INPUT:

- `other` – WangTileSet
- `direction` – integer (1 or 2)
- `function` – function (default:`str.__add__`), monoid operation
- `initial` – object (default:`''`), monoid neutral
- `clean_graph` – boolean (default: `False`), clean the graph by recursively removing sources and sinks transitions (or tiles).

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = ['ABCD', 'EFGH', 'AXCY', 'ABAB']
sage: tiles = map(tuple, tiles)
sage: T = WangTileSet(tiles)
sage: T1T = T.fusion(T, 1)
sage: T1T.tiles()
[('A', 'BB', 'A', 'BB')]
sage: T2T = T.fusion(T, 2)
sage: T2T.tiles()
[('AA', 'B', 'AA', 'B')]
```

To keep integers, one way is to wrap them into a tuple and do tuple operations:

```
sage: tiles = [(0,1,0,1)]
sage: tiles = [tuple((a,) for a in tile) for tile in tiles]
sage: T = WangTileSet(tiles)
sage: T2T = T.fusion(T, 2, function=tuple.__add__, initial=tuple())
sage: T2T2T = T2T.fusion(T, 2, function=tuple.__add__, initial=tuple())
sage: T2T2T.tiles()
[((0, 0, 0), (1,), (0, 0, 0), (1,))]
```

TESTS:

```
sage: tiles = [('02', '2', '02', '2'), ('32', '2', '02', '2')]
sage: T = WangTileSet(tiles)
sage: T.fusion(T, 1)
Wang tile set of cardinality 2
sage: T.fusion(T, 2)
Wang tile set of cardinality 1
```

horizontal_alphabet()

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,1,2,3), (4,5,6,7), (8,9,10,11)]
sage: T = WangTileSet(tiles)
sage: T.horizontal_alphabet()
{1, 3, 5, 7, 9, 11}
```

is_equivalent(*other*, *certificate=False*, *verbose=False*)

INPUT:

- `other` – wang tile set
- `certificate` – boolean (default:`False`)
- `verbose` – boolean (default:`False`)

Note: This code depends on the following bug to be fixed: https://trac.sagemath.org/ticket/24964

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(1,6,1,8), (2,6,1,7), (3,7,1,6), (1,6,2,6),
....:          (2,8,2,7), (2,7,3,6), (3,6,3,7)]
sage: T = WangTileSet(tiles)
sage: d = {1:'a', 2:'b', 3:'c', 6:'x', 7:'y', 8:'z'}
sage: L = [tuple(d[a] for a in t) for t in tiles]
sage: U = WangTileSet(L)
sage: T.is_equivalent(U)
True
sage: T.is_equivalent(U,certificate=True)
(True,
 {1: 'a', 2: 'b', 3: 'c'},
 {6: 'x', 7: 'y', 8: 'z'},
 Substitution 2d: {0: [[0]], 1: [[1]], 2: [[2]], 3: [[3]], 4: [[4]], 5: [[5]], 6: [[6]]})
```

Not equivalent example:

```
sage: _ = L.pop()
sage: U = WangTileSet(L)
sage: T.is_equivalent(U)
False
sage: T.is_equivalent(U,certificate=True)
(False, None, None, None)
```

When graphs admits non trivial automorphisms:

```
sage: T = WangTileSet([(1,3,0,2), (0,2,1,3)])
sage: U = WangTileSet([(7,'c',6,'z'), (6,'z',7,'c')])
sage: V = WangTileSet([(7,9,6,8), (6,8,7,9)])
sage: W = WangTileSet([(7,8,6,9), (6,9,7,8)])
sage: T.is_equivalent(T, certificate=True)
(True, {0: 0, 1: 1}, {2: 2, 3: 3}, Substitution 2d: {0: [[0]], 1: [[1]]})
sage: T.is_equivalent(U, certificate=True)
(True, {0: 6, 1: 7}, {2: 'z', 3: 'c'}, Substitution 2d: {0: [[0]], 1: [[1]]})
sage: T.is_equivalent(V, certificate=True)
(True, {0: 6, 1: 7}, {2: 8, 3: 9}, Substitution 2d: {0: [[0]], 1: [[1]]})
sage: T.is_equivalent(W, certificate=True)
(True, {0: 6, 1: 7}, {2: 9, 3: 8}, Substitution 2d: {0: [[0]], 1: [[1]]})
sage: T.is_equivalent(W, certificate=True, verbose=True)
True V_perm= {0: 6, 1: 7}
True H_perm= {2: 8, 3: 9}
Found automorphisms p=() and q=(2,3)
(True, {0: 6, 1: 7}, {2: 9, 3: 8}, Substitution 2d: {0: [[0]], 1: [[1]]})
```

is_forbidden_product(*A*, *B*, *i=2*, *radius=1*, *solver=None*, *ncpus=None*)

Return whether A odot^i B is forbidden using a given radius around the product and a given solver.

INPUT:

- `A` – list of tile indices
- `B` – list of tile indices
- `i` – integer, 1 or 2
- `radius` – integer (default:`1`)
- `solver` – string (default:`None`)
- `ncpus` – integer (default:`None`)

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = ['ABCD', 'EFGH', 'AXCY', 'ABAB']
sage: T = WangTileSet(tiles)
sage: T.is_forbidden_product([3],[3])
False
sage: T.is_forbidden_product([0],[0])
True
sage: T.is_forbidden_product([0,1],[0,1])
True
sage: T.is_forbidden_product([0,1],[0,1,2])
True
sage: T.is_forbidden_product([0,1],[0,1,2,3])
True
sage: T.is_forbidden_product([0,1,3],[0,1,2,3])
False
```

not_forbidden_dominoes()

Deprecated: Use *dominoes_with_surrounding()* instead. See trac ticket #123456 for details.

not_forbidden_tilings()

Deprecated: Use *tilings_with_surrounding()* instead. See trac ticket #123456 for details.

polyhedron_of_densities()

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,2), (1,0,0,1), (2,1,0,0), (0,0,1,0),
....:          (1,2,1,1), (1,1,2,0), (2,0,2,1)]
sage: T = WangTileSet(tiles)
sage: P = T.polyhedron_of_densities()
sage: P
A 2-dimensional polyhedron in QQ^7 defined as the convex hull of 3 vertices
sage: P.vertices()
(A vertex at (0, 2/7, 1/7, 3/7, 0, 1/7, 0),
 A vertex at (0, 0, 1/5, 1/5, 0, 1/5, 2/5),
 A vertex at (2/7, 0, 1/7, 1/7, 2/7, 1/7, 0))
sage: (0, 0, 1/5, 1/5, 0, 1/5, 2/5) in P
True
```

Jeandel-Rao tiles:

```
sage: tiles = [(2,4,2,1), (2,2,2,0), (1,1,3,1), (1,2,3,2), (3,1,3,3),
....: (0,1,3,1), (0,0,0,1), (3,1,0,2), (0,2,1,2), (1,2,1,4), (3,3,1,2)]
sage: T = WangTileSet(tiles)
sage: P = T.polyhedron_of_densities()
sage: P
A 4-dimensional polyhedron in QQ^11 defined as the convex hull of 10 vertices
sage: P.vertices()
(A vertex at (0, 1/5, 1/5, 0, 1/5, 0, 1/5, 0, 0, 0, 1/5),
 A vertex at (0, 1/5, 0, 1/5, 1/5, 0, 1/5, 0, 0, 0, 1/5),
 A vertex at (0, 1/5, 1/5, 0, 0, 0, 1/5, 1/5, 1/5, 0, 0),
 A vertex at (0, 1/5, 0, 1/5, 0, 0, 1/5, 1/5, 1/5, 0, 0),
 A vertex at (0, 1/4, 0, 0, 0, 1/4, 1/4, 1/4, 0, 0, 0),
 A vertex at (1/4, 0, 0, 0, 0, 1/4, 0, 1/4, 0, 1/4, 0),
 A vertex at (1/5, 0, 1/5, 0, 1/5, 0, 0, 0, 0, 1/5, 1/5),
 A vertex at (1/5, 0, 1/5, 0, 0, 0, 0, 1/5, 1/5, 1/5, 0),
 A vertex at (1/5, 0, 0, 1/5, 1/5, 0, 0, 0, 0, 1/5, 1/5),
 A vertex at (1/5, 0, 0, 1/5, 0, 0, 0, 1/5, 1/5, 1/5, 0))
```

shear(*radius=0*, *solver=None*, *ncpus=1*, *function=<slot wrapper '__add__' of 'str' objects>*, *verbose=False*)

Shears the Wang Tile set by the `matrix(2,(1,-1,0,1))`.

It is currently not implemented for other matrices.

INPUT:

- `radius` - integer or 2-tuple (default: `0`), if 2-tuple is given, then it is interpreted as `(xradius, yradius)`
- `solver` – string (default:`None`)
- `ncpus` – integer (default:1)
- `function` – function (default:`str.__add__`), monoid operation
- `verbose` – boolean (default:`False`)

OUTPUT:

- (WangTileSet, Substitution2d)

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [('aa','bb','cc','bb'), ('cc','dd','aa','dd')]
sage: T = WangTileSet(tiles)
sage: U,s = T.shear()
sage: s
Substitution 2d: {0: [[0]], 1: [[1]]}
sage: U.tiles()
```

(continues on next page)

(continued from previous page)

```
[('aadd', 'dd', 'ccbb', 'bb'), ('ccbb', 'bb', 'aadd', 'dd')]
sage: T.shear()[0].shear()[0].tiles()
[('aaddbb', 'bb', 'ccbbdd', 'bb'), ('ccbbdd', 'dd', 'aaddbb', 'dd')]
```

```
sage: tiles = [('aa','bb','cc','bb'), ('aa','dd','cc','bb'), ('cc','dd','aa','dd')]
sage: T = WangTileSet(tiles)
sage: U,s = T.shear()
sage: s
Substitution 2d: {0: [[0]], 1: [[1]], 2: [[2]], 3: [[2]]}
sage: U.tiles()
[('aadd', 'dd', 'ccbb', 'bb'),
 ('aadd', 'dd', 'ccdd', 'bb'),
 ('ccdd', 'dd', 'aadd', 'dd'),
 ('ccbb', 'bb', 'aadd', 'dd')]
sage: U,s = T.shear(radius=1)
sage: s
Substitution 2d: {0: [[0]], 1: [[2]]}
sage: U.tiles()
[('aadd', 'dd', 'ccbb', 'bb'), ('ccbb', 'bb', 'aadd', 'dd')]
```

`solver`(*width*, *height*, *preassigned_color=None*, *preassigned_tiles=None*, *color=None*)

Return the Wang tile solver of this Wang tile set inside a rectangle of given width and height.

INPUT:

- `width` – integer
- `height` – integer
- `preassigned_color` – None or list of 4 dict or the form `[{}, {}, {}, {}]` right, top, left, bottom colors preassigned to some positions (on the border or inside)
- `preassigned_tiles` – None or dict of tiles preassigned to some positions
- `color` – None or dict

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2), (0,1,2,0)]
sage: T = WangTileSet(tiles)
sage: W = T.solver(3,3)
sage: W.solve()
A wang tiling of a 3 x 3 rectangle
```

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: T = WangTileSet(tiles)
sage: W = T.solver(3,3, preassigned_tiles={(1,1):0})
sage: W.solve().table()
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
sage: W = T.solver(3,3, preassigned_tiles={(1,1):1})
sage: W.solve().table()
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]
sage: W = T.solver(3,3, preassigned_tiles={(1,1):2})
sage: W.solve().table()
[[2, 2, 2], [2, 2, 2], [2, 2, 2]]
```

When incompatible preassigned tiles:

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: T = WangTileSet(tiles)
sage: W = T.solver(3,3, preassigned_tiles={(0,0):0,(0,1):1})
sage: W.has_solution()
False
```

TESTS:

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2), (0,1,2,0)]
sage: T = WangTileSet(tiles)
sage: W = T.solver(3,3, preassigned_tiles={(1,1):3})
sage: W.solve().table()
Traceback (most recent call last):
...
MIPSolverException: ...
```

substitution_tikz(*substitution*, *function=None*, *color=None*, *size=1*, *scale=1*, *font='\\normalsize'*, *rotate=None*, *label_shift=0.2*, *ncolumns=4*, *tabular='tabular'*, *align='l'*)

Return the tikz code showing what the substitution A->B* does on Wang tiles.

Note: we assume that the tiles in self are the elements of B.

INPUT:

- `substitution` – substitution 2d
- `fn` – a function (default: `None`) to apply to the new colors which are tuple of previous colors
- `color` – dict (default: `None`) from tile values -> tikz colors
- `size` – number (default: 1), size of the tile
- `scale` – number (default: 1), scale of tikzpicture
- `font` – string (default: `r'\normalsize'`
- `rotate` – list or `None` (default:`None`) list of four angles in degrees like `(0,0,0,0)`, the rotation angle to apply to each label of Wang tiles. If `None`, it performs a 90 degres rotation for left and right labels taking more than one character.
- `label_shift` – number (default: `.2`) translation distance of the label from the edge
- `ncolumns` – integer (default: `4`)
- `tabular` – string (default: `'tabular'`) or `'longtable'`
- `align` – character (default:`'l'`), latex alignment symbol `'l'`, `'r'` or `'c'`.

OUTPUT:

dict, key -> tile

system_of_density_equations()

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,2), (1,0,0,1), (2,1,0,0), (0,0,1,0),
....:          (1,2,1,1), (1,1,2,0), (2,0,2,1)]
sage: T = WangTileSet(tiles)
sage: M = T.system_of_density_equations()
sage: M
[ 0  1  1 -1  0  0  0  0]
[ 0 -1  0  1  0 -1  0  0]
[ 0  0 -1  0  0  1  0  0]
[ 1  1 -1  0  0 -1  1  0]
[ 0 -1  1  0 -1  1 -1  0]
[-1  0  0  0  1  0  0  0]
[ 1  1  1  1  1  1  1  1]
sage: M.rank()
5
```

table()

Return a table representation of the tile set.

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,2), (1,0,0,1), (2,1,0,0), (0,0,1,0),
....:          (1,2,1,1), (1,1,2,0), (2,0,2,1)]
sage: T = WangTileSet(tiles)
sage: T.table()
  Id   Right   Top   Left   Bottom
+----+-------+-----+------+--------+
  0    0       0     0      2
  1    1       0     0      1
  2    2       1     0      0
  3    0       0     1      0
  4    1       2     1      1
  5    1       1     2      0
  6    2       0     2      1
```

tikz(*ncolumns=10*, *color=None*, *size=1*, *space=0.1*, *scale=1*, *font='\\normalsize'*, *rotate=None*, *label=True*, *id=True*, *id_format='{}'*, *id_color=''*, *label_shift=0.2*, *label_color='black'*, *right_edges=True*, *top_edges=True*, *left_edges=True*, *bottom_edges=True*, *draw_H=None*, *draw_V=None*)

INPUT:

- `ncolumns` – integer (default: `10`)
- `color` – dict (default: None)
- `size` – number (default: `1`)
- `space` – number (default: `.1`)
- `scale` – number (default: `1`)
- `font` – string (default: `r'\normalsize'`)
- `rotate` – list or `None` (default:`None`) list of four angles in degrees like `(0,0,0,0)`, the rotation angle to apply to each label of Wang tiles. If `None`, it performs a 90 degres rotation for left and right labels taking more than one character.
- `label` – boolean (default: `True`), presence of the colors
- `id` – boolean (default: `True`), presence of the tile id
- `id_color` – string (default: `''`)
- `id_format` – string (default: `r'{}'`) to be called with `id_format.format(key)`
- `label_shift` – number (default: `.2`) translation distance of the label from the edge
- `label_color` – string (default: `'black'`)
- `right_edges` – bool (default: `True`)
- `top_edges` – bool (default: `True`)
- `left_edges` – bool (default: `True`)
- `bottom_edges` – bool (default: `True`)
- `draw_H` – dict (default: `None`) from tile values -> tikz draw commands. If `None` the values of the dict get replaced by straight lines, more precisely by `r'\draw {{}} -- ++ (1,0);'`. Dict values must be strings `s` such that `s.format((x,y))` works.
- `draw_V` – dict (default: `None`) from tile values -> tikz draw commands. If `None` the values of the dict get replaced by straight lines, more precisely by `r'\draw {{}} -- ++ (0,1);'`. Dict values must be strings `s` such that `s.format((x,y))` works.

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,2), (1,0,0,1), (2,1,0,0), (0,0,1,0),
....:          (1,2,1,1), (1,1,2,0), (2,0,2,1)]
sage: T = WangTileSet(tiles)
sage: color = {0:'white',1:'red',2:'cyan',3:'green',4:'white'}
sage: _ = T.tikz(color=color).pdf(view=False)
```

tiles()

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: T = WangTileSet(tiles)
sage: T.tiles()
[(0, 0, 0, 0), (1, 1, 1, 1), (2, 2, 2, 2)]
```

tiles_allowing_surrounding(*radius*, *solver=None*, *ncpus=None*, *verbose=False*)

Return the subset of tiles allowing a surrounding of given radius.

INPUT:

- `radius` - integer
- `solver` - string or None
- `ncpus` - integer
- `verbose` - boolean

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2), (0,1,2,0)]
sage: T = WangTileSet(tiles)
sage: U = T.tiles_allowing_surrounding(1)
sage: U
Wang tile set of cardinality 3
sage: U.tiles()
[(0, 0, 0, 0), (1, 1, 1, 1), (2, 2, 2, 2)]
```

```
sage: T.tiles_allowing_surrounding(1, verbose=True)
Solution found for tile 0:
[[0, 0, 3], [0, 0, 0], [0, 0, 0]]
Solution found for tile 1:
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]
Solution found for tile 2:
[[2, 2, 2], [2, 2, 2], [2, 2, 2]]
Wang tile set of cardinality 3
```

tiling_with_surrounding()

Deprecated: Use *tilings_with_surrounding()* instead. See trac ticket #123456 for details.

tilings_with_surrounding(*width*, *height*, *radius=1*, *solver=None*, *verbose=False*)

Return the set of valid tiling of a rectangle of given width and height allowing a surrounding of itself of given radius.

INPUT:

- `width` - integer
- `height` - integer
- `radius` - integer

- `solver` - string or None
- `verbose` - boolean

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2), (0,1,2,0)]
sage: tiles = [map(str, tile) for tile in tiles]
sage: T = WangTileSet(tiles)
sage: S = T.tilings_with_surrounding(2,2)
sage: S
[A wang tiling of a 2 x 2 rectangle,
 A wang tiling of a 2 x 2 rectangle,
 A wang tiling of a 2 x 2 rectangle]
sage: [a.table() for a in S]
[[[0, 0], [0, 0]], [[1, 1], [1, 1]], [[2, 2], [2, 2]]]
```

```
sage: S = T.tilings_with_surrounding(3,3)
sage: S
[A wang tiling of a 3 x 3 rectangle,
 A wang tiling of a 3 x 3 rectangle,
 A wang tiling of a 3 x 3 rectangle]
sage: [a.table() for a in S]
[[[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[1, 1, 1], [1, 1, 1], [1, 1, 1]],
 [[2, 2, 2], [2, 2, 2], [2, 2, 2]]]
```

TESTS:

```
sage: tiles = ['ABCD', 'EFGH', 'AXCY', 'ABAB', 'EBEB']
sage: T = WangTileSet(tiles)
sage: solutions = T.tilings_with_surrounding(1,2)
sage: [t.table() for t in solutions]
[[[3, 3]], [[3, 4]], [[4, 3]], [[4, 4]]]
```

```
sage: tiles = [('02', '4', '02', '4'), ('32', '4', '02', '4')]
sage: T = WangTileSet(tiles)
sage: [t.table() for t in T.tilings_with_surrounding(1,2)]
[[[0, 0]]]
sage: [t.table() for t in T.tilings_with_surrounding(2,1)]
[[[0], [0]]]
```

to_transducer()

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,0,0,2), (1,0,0,1), (2,1,0,0), (0,0,1,0),
....:          (1,2,1,1), (1,1,2,0), (2,0,2,1)]
sage: T = WangTileSet(tiles)
sage: T.to_transducer()
Transducer with 3 states
```

to_transducer_graph(*label_function=<type 'tuple'>*, *merge_multiedges=True*)

Return the graph of the transducer.

Labels are cleaned. Label of multiedges are merged with commas.

INPUT:

- `label_function` – function (default:`tuple`), a function to apply to each list of labels when merging multiedges into one
- `merge_multiedges` – boolean (default:`True`)

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = ['ABCD', 'EFGH', 'AXCY']
sage: tiles = map(tuple, tiles)
sage: T = WangTileSet(tiles)
sage: G = T.to_transducer_graph()
sage: G
Digraph on 4 vertices
```

The edge labels are clean:

```
sage: G.edges()
[('C', 'A', ('D|B', 'Y|X')), ('G', 'E', ('H|F',))]
```

Using `label_function`:

```
sage: fn = lambda L: ','.join(map(str, L))
sage: G = T.to_transducer_graph(label_function=fn)
sage: G.edges()
[('C', 'A', 'D|B,Y|X'), ('G', 'E', 'H|F')]
```

Using `label_function` with latex expressions:

```
sage: fn = lambda L: LatexExpr(','.join(map(str, L)))
sage: G = T.to_transducer_graph(label_function=fn)
sage: G.edges()
[('C', 'A', D|B,Y|X), ('G', 'E', H|F)]
```

This is to compared to:

```
sage: T.to_transducer().graph().edges()
[('C', 'A', "'D'|'B'"), ('C', 'A', "'Y'|'X'"), ('G', 'E', "'H'|'F'")]
```

It works for integers entries:

```
sage: tiles = [(0,1,2,3), (0,5,2,3)]
sage: T = WangTileSet(tiles)
sage: G = T.to_transducer_graph()
sage: G
Digraph on 2 vertices
sage: G.edges()
[(2, 0, ('3|1', '3|5'))]
```

unsynchronized_graph(*i=1*, *size=2*, *verbose=False*)

INPUT:

- `i` – integer, 1 or 2
- `size` – integer, 2 or more
- `verbose` – boolean (default:`False`)

OUTPUT:

- graph of vertices (delays, blocks)

Signification of the nodes (d,b):

```
     +-----------+
     |           |
     |    b[1]   |
     |           |
+----+-----+-----+
```

(continues on next page)

(continued from previous page)

```
|          |     |
|   b[0]   |    d[1]
|          |
+----------+
           |
          d[0]
```

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [('aa','bb','cc','bb'), ('cc','dd','aa','dd')]
sage: T = WangTileSet(tiles)
sage: G = T.unsynchronized_graph()
sage: sorted(G.vertices()) # known bug
[(d=(0, 0), b=(0, 0)),
 (d=(0, 0), b=(1, 1)),
 (d=(2, 0), b=(0, 1)),
 (d=(2, 0), b=(1, 0))]
sage: G.edges()   # known bug
[((d=(0, 0), b=(1, 1)), (d=(2, 0), b=(0, 1)), (1, 0)),
 ((d=(0, 0), b=(0, 0)), (d=(2, 0), b=(1, 0)), (0, 1)),
 ((d=(2, 0), b=(1, 0)), (d=(0, 0), b=(1, 1)), (0, 1)),
 ((d=(2, 0), b=(0, 1)), (d=(0, 0), b=(0, 0)), (1, 0))]
sage: [node.lengths_x() for node in G]
[[2, 2], [2, 2], [2, 2], [2, 2]]
sage: [node.is_synchronized() for node in G]
[True, True, True, True]
sage: from slabbe import TikzPicture
sage: _ = TikzPicture.from_graph(G).pdf(view=False)
```

unsynchronized_graph_size2(*i=1*)

INPUT:

- i – integer, 1 or 2

Signification of the nodes (u,v,w,d):

```
d = 0      |w| = d > 0       -|w| = d < 0

  |              |           |
 v|             v|          v|
  |           w  |           |
  +         +-----+          +-----+
  |         |                  w   |
 u|        u|                     u|
  |         |                      |
```

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [('aa','bb','cc','bb'), ('cc','dd','aa','dd')]
sage: T = WangTileSet(tiles)
sage: G = T.unsynchronized_graph_size2()
sage: sorted(G.vertices())
[('aa', 'aa', '', 0), ('cc', 'cc', '', 0)]
```

vertical_alphabet()

EXAMPLES:

```
sage: from slabbe import WangTileSet
sage: tiles = [(0,1,2,3), (4,5,6,7), (8,9,10,11)]
sage: T = WangTileSet(tiles)
sage: T.vertical_alphabet()
{0, 2, 4, 6, 8, 10}
```

class slabbe.wang_tiles.**WangTileSolver**(*tiles*, *width*, *height*, *preassigned_color=None*, *preassigned_tiles=None*, *color=None*)

Bases: `object`

Wang tile solver inside a rectangle of given width and height.

INPUT:

- `tiles` – list of tiles, a tile is a 4-tuple (right color, top color, left color, bottom color)
- `width` – integer
- `height` – integer
- `preassigned_color` – None or list of 4 dict or the form `[{}, {}, {}, {}]` right, top, left, bottom colors preassigned to some positions (on the border or inside)
- `preassigned_tiles` – None or dict of tiles preassigned to some positions
- `color` – None or dict

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: W = WangTileSolver(tiles, 3, 3)
sage: tiling = W.solve()
sage: tiling._table
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

With color 2 preassigned to the right part of tile at position (1,1):

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: right = {(1,1):2}
sage: W = WangTileSolver(tiles,3,3,preassigned_color=[right,{},{},{}])
sage: tiling = W.solve()
sage: tiling._table
[[2, 2, 2], [2, 2, 2], [2, 2, 2]]
```

With tile 2 preassigned at position (0,1):

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: preassigned = {(0,1):1}
sage: W = WangTileSolver(tiles,3,3,preassigned_tiles=preassigned)
sage: tiling = W.solve()
sage: tiling._table
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]
```

When constraints are inconsistent:

```
sage: right = {(1,1):1, (2,2):0}
sage: W = WangTileSolver(tiles,3,3,preassigned_color=[right,{},{},{}])
sage: W.solve(solver='GLPK')
Traceback (most recent call last):
...
MIPSolverException: GLPK: Problem has no feasible solution
```

TESTS:

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2), (0,1,2,0)]
sage: preassigned = {(0,1):1}
sage: W = WangTileSolver(tiles,3,3,preassigned_tiles=preassigned)
sage: tiling = W.solve()
sage: tiling._table
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]
```

all_solutions(*ncpus=8*)

Return the list of all solutions.

Note: This uses the reduction to dancing links.

INPUT:

- ncpus – integer (default: 8), maximal number of subprocesses to use at the same time

OUTPUT:

list of wang tilings

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(2,4,2,1), (2,2,2,0), (1,1,3,1), (1,2,3,2), (3,1,3,3),
....: (0,1,3,1), (0,0,0,1), (3,1,0,2), (0,2,1,2), (1,2,1,4), (3,3,1,2)]
sage: W = WangTileSolver(tiles,3,4)
sage: W.number_of_solutions()
908
sage: L = W.all_solutions()
sage: len(L)
908
```

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: W = WangTileSolver(tiles,2,2)
sage: W.all_solutions()
[A wang tiling of a 2 x 2 rectangle,
 A wang tiling of a 2 x 2 rectangle,
 A wang tiling of a 2 x 2 rectangle]
```

With preassigned colors and tiles:

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2), (0,1,2,0)]
sage: t = {(0,1):0}
sage: c = [{},{},{(1,1):0},{}]
sage: W = WangTileSolver(tiles,3,3,preassigned_tiles=t,preassigned_color=c)
sage: S = W.all_solutions()
sage: sorted([s._table for s in S])
[[[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 3], [0, 0, 0], [0, 0, 0]]]
```

With preassigned colors and tiles:

```
sage: right = {(0, 1): 'A', (0, 0): 'A'}
sage: top = {(0, 1): 'B'}
sage: left = {(0, 1): 'A', (0, 0): 'A'}
sage: bottom = {(0, 0): 'B'}
sage: preassigned_color=[right,top,left,bottom]
sage: tiles = ['ABCD', 'EFGH', 'AXCY', 'ABAB', 'EBEB']
sage: W = WangTileSolver(tiles, 1, 2, preassigned_color=preassigned_color)
sage: [t.table() for t in W.all_solutions()]
[[[3, 3]]]
```

all_solutions_tikz(*ncpus=8*)

INPUT:

- ncpus – integer (default: 8), maximal number of subprocesses to use at the same time

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(2,4,2,1), (2,2,2,0), (1,1,3,1), (1,2,3,2), (3,1,3,3),
....: (0,1,3,1), (0,0,0,1), (3,1,0,2), (0,2,1,2), (1,2,1,4), (3,3,1,2)]
sage: W = WangTileSolver(tiles,2,2)
sage: t = W.all_solutions_tikz()
sage: view(t)    # long # not tested
```

dlx_solver()

Return the sage DLX solver of that Wang tiling problem.

OUTPUT:

DLX Solver

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: W = WangTileSolver(tiles,3,4)
sage: dlx = W.dlx_solver()
sage: dlx
Dancing links solver for 63 columns and 24 rows
sage: dlx.number_of_solutions()
2
```

TESTS:

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: W = WangTileSolver(tiles,2,2)
sage: dlx = W.dlx_solver()
sage: list(dlx.solutions_iterator())
[[1, 7, 4, 10], [6, 0, 9, 3], [8, 2, 5, 11]]
```

has_solution(*solver=None*, *solver_parameters=None*, *ncpus=1*)

Return whether there is a solution.

INPUT:

- `solver` – string or None (default: `None`), `'dancing_links'` or the name of a MILP solver in Sage like `'GLPK'`, `'Coin'` or `'Gurobi'`.
- `solver_parameters` – dict (default: `{}`), parameters given to the MILP solver using method `solver_parameter`. For a list of available parameters for example for the Gurobi backend, see dictionary `parameters_type` in the file `sage/numerical/backends/gurobi_backend.pyx`
- `ncpus` – integer (default: 1), maximal number of subprocesses to use at the same time, used only if `solver` is `'dancing_links'`.

OUTPUT:

a wang tiling object

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: W = WangTileSolver(tiles,3,4)
sage: W.has_solution()
True
```

Allowing more threads while using Gurobi:

```
sage: W = WangTileSolver(tiles,3,4)
sage: kwds = dict(Threads=4)
sage: tiling = W.has_solution(solver='Gurobi', kwds) # optional Gurobi
True
```

Using dancing links:

```
sage: W = WangTileSolver(tiles,3,4)
sage: W.has_solution(solver='dancing_links', ncpus=8)
True
```

Using cryptominisat:

```
sage: W = WangTileSolver(tiles,3,4)
sage: W.has_solution(solver='cryptominisat') # optional cryptominisat
True
```

horizontal_alphabet()

meet_of_all_solutions(*ncpus=8*)

Return the tiling of the rectangle with tiles that are imposed at each position (this is the meet of the partially ordered set of all partial solutions inside the rectangle).

INPUT:

- `ncpus` – integer (default: 8), maximal number of subprocesses to use at the same time

OUTPUT:

A Wang tiling (with `None` at positions where more than one tile can occur)

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2), (0,1,2,0)]
sage: t = {(0,1):0}
sage: W = WangTileSolver(tiles,3,3,preassigned_tiles=t)
sage: tiling = W.meet_of_all_solutions()
sage: tiling
A wang tiling of a 3 x 3 rectangle
sage: tiling.table()
[[0, 0, None], [0, 0, 0], [0, 0, 0]]
```

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2), (0,1,2,0)]
sage: W = WangTileSolver(tiles,3,3)
sage: tiling = W.meet_of_all_solutions()
sage: tiling.table()
[[None, None, None], [None, None, None], [None, None, None]]
```

milp(*solver=None*)

Return the Mixed integer linear program.

INPUT:

- `solver` – string or None (default: `None`), other possible values are `'Coin'` or `'Gurobi'`

OUTPUT:

a tuple (p,x) where p is the MILP and x is the variable

Note: In some cases, calling this method takes much more time (few minutes) than calling the method `solve` which takes few seconds.

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: W = WangTileSolver(tiles,3,4)
sage: p,x = W.milp(solver='GLPK')
sage: p
Boolean Program (maximization, 36 variables, 29 constraints)
sage: x
MIPVariable of dimension 1
```

Then you can solve it and get the solutions:

```
sage: p.solve()
1.0
sage: soln = p.get_values(x)
sage: support = [key for key in soln if soln[key]]
sage: support
[(0, 1, 1), (0, 1, 3), (0, 2, 1), (0, 2, 0), (0, 2, 3), (0, 2, 2),
 (0, 1, 2), (0, 0, 3), (0, 0, 2), (0, 0, 1), (0, 0, 0), (0, 1, 0)]
```

Other solver can be used:

```
sage: p,x = W.milp(solver='Gurobi')   # optional gurobi
```

TESTS:

Colors do not have to be integers:

```
sage: tiles = [('a','a','a','a'), ('b','b','b','b')]
sage: W = WangTileSolver(tiles,3,4)
sage: p,x = W.milp()
sage: tiling = W.solve()
```

number_of_solutions(*ncpus=8*)

Return the number of solutions

INPUT:

- ncpus – integer (default: 8), maximal number of subprocesses to use at the same time

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(2,4,2,1), (2,2,2,0), (1,1,3,1), (1,2,3,2), (3,1,3,3),
....: (0,1,3,1), (0,0,0,1), (3,1,0,2), (0,2,1,2), (1,2,1,4), (3,3,1,2)]
sage: W = WangTileSolver(tiles,3,4)
sage: W.number_of_solutions()
908
```

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: W = WangTileSolver(tiles,2,2)
sage: W.number_of_solutions()
3
```

rows_and_information(*verbose=False*)

Return the rows to give to the dancing links solver.

INPUT:

- verbose – bool (default: False)

OUTPUT:

Two lists:

- the rows
- row information (j,k,i) meaning tile i is at position (j,k)

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: W = WangTileSolver(tiles, 4, 1)
sage: rows,row_info = W.rows_and_information()
sage: rows
[[1, 2, 9],
 [0, 2, 9],
 [2, 9],
 [0, 4, 5, 10],
 [1, 3, 5, 10],
 [0, 1, 5, 10],
 [3, 7, 8, 11],
 [4, 6, 8, 11],
 [3, 4, 8, 11],
 [6, 12],
 [7, 12],
 [6, 7, 12]]
sage: row_info
[(0, 0, 0),
 (0, 0, 1),
 (0, 0, 2),
 (1, 0, 0),
 (1, 0, 1),
 (1, 0, 2),
 (2, 0, 0),
 (2, 0, 1),
 (2, 0, 2),
 (3, 0, 0),
 (3, 0, 1),
 (3, 0, 2)]
sage: from sage.combinat.matrices.dancing_links import dlx_solver
sage: dlx = dlx_solver(rows)
sage: dlx
Dancing links solver for 13 columns and 12 rows
sage: dlx.search()
1
sage: dlx.get_solution()
[1, 4, 7, 10]
sage: row_info[1]
(0, 0, 1)
sage: row_info[4]
(1, 0, 1)
sage: row_info[7]
(2, 0, 1)
sage: row_info[10]
(3, 0, 1)
```

... which means tile 1 is at position (0,0), (1,0), (2,0) and (3,0)

TESTS:

```
sage: tiles = [(0,0,0,0), (1,1,1,1)]
sage: W = WangTileSolver(tiles, 4, 1)
sage: W.rows_and_information(verbose=True)
Vertical colors (coded using 3 bits):
color 0 represented by bits [0] when on left
color 0 represented by bits [1, 2] when on right
color 1 represented by bits [1] when on left
color 1 represented by bits [0, 2] when on right
Horizontal colors (coded using 3 bits):
color 0 represented by bits [0] when on bottom
```

(continues on next page)

(continued from previous page)

```
color 0 represented by bits [1, 2] when on top
color 1 represented by bits [1] when on bottom
color 1 represented by bits [0, 2] when on top
([[1, 2, 9],
  [0, 2, 9],
  [0, 4, 5, 10],
  [1, 3, 5, 10],
  [3, 7, 8, 11],
  [4, 6, 8, 11],
  [6, 12],
  [7, 12]],
 [(0, 0, 0),
  (0, 0, 1),
  (1, 0, 0),
  (1, 0, 1),
  (2, 0, 0),
  (2, 0, 1),
  (3, 0, 0),
  (3, 0, 1)])
```

```
sage: tiles = [(0,0,0,0)]
sage: W = WangTileSolver(tiles, 4, 1)
sage: W.rows_and_information(verbose=True)
Vertical colors (coded using 2 bits):
color 0 represented by bits [0] when on left
color 0 represented by bits [1] when on right
Horizontal colors (coded using 2 bits):
color 0 represented by bits [0] when on bottom
color 0 represented by bits [1] when on top
([[1, 6], [0, 3, 7], [2, 5, 8], [4, 9]],
 [(0, 0, 0), (1, 0, 0), (2, 0, 0), (3, 0, 0)])
```

With preassigned colors:

```
sage: right = {(0, 1): 'A', (0, 0): 'A'}
sage: top = {(0, 1): 'B'}
sage: left = {(0, 1): 'A', (0, 0): 'A'}
sage: bottom = {(0, 0): 'B'}
sage: preassigned_color=[right,top,left,bottom]
sage: tiles = ['ABCD', 'EFGH', 'AXCY', 'ABAB', 'EBEB']
sage: W = WangTileSolver(tiles, 1, 2, preassigned_color=preassigned_color)
sage: W.rows_and_information()
([[4], [4], [4], [1, 2, 3, 4], [4], [5], [5], [5], [0, 5], [5]],
 [(0, 0, 0),
  (0, 0, 1),
  (0, 0, 2),
  (0, 0, 3),
  (0, 0, 4),
  (0, 1, 0),
  (0, 1, 1),
  (0, 1, 2),
  (0, 1, 3),
  (0, 1, 4)])
```

sat_solver(*solver=None*)

Return the SAT solver.

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: W = WangTileSolver(tiles,3,4)
sage: s = W.sat_solver()
sage: s           # random
```

(continues on next page)

(continued from previous page)

```
an ILP-based SAT Solver
CryptoMiniSat solver: 24 variables, 58 clauses.
sage: L = s()
sage: list(L)
[None, False, True, False, True, True, False, True, False,
 False, True, False, True, True, False, True, False, False,
 True, False, True, True, False, True, False]
```

sat_variable_to_tile_position_bijection()

Return the dictionary giving the correspondence between variables and tiles indices i at position (j,k)

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: W = WangTileSolver(tiles,3,4)
sage: d1,d2 = W.sat_variable_to_tile_position_bijection()
sage: d1
{1: (0, 0, 0),
 2: (0, 0, 1),
 3: (0, 0, 2),
 4: (0, 0, 3),
 5: (0, 1, 0),
 6: (0, 1, 1),
 7: (0, 1, 2),
 8: (0, 1, 3),
 9: (0, 2, 0),
 10: (0, 2, 1),
 11: (0, 2, 2),
 12: (0, 2, 3),
 13: (1, 0, 0),
 14: (1, 0, 1),
 15: (1, 0, 2),
 16: (1, 0, 3),
 17: (1, 1, 0),
 18: (1, 1, 1),
 19: (1, 1, 2),
 20: (1, 1, 3),
 21: (1, 2, 0),
 22: (1, 2, 1),
 23: (1, 2, 2),
 24: (1, 2, 3)}
```

solutions_iterator()

Iterator over all solutions

Note: This uses the reduction to dancing links.

OUTPUT:

iterator of wang tilings

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(2,4,2,1), (2,2,2,0), (1,1,3,1), (1,2,3,2), (3,1,3,3),
....: (0,1,3,1), (0,0,0,1), (3,1,0,2), (0,2,1,2), (1,2,1,4), (3,3,1,2)]
sage: W = WangTileSolver(tiles,3,4)
sage: it = W.solutions_iterator()
sage: next(it)
A wang tiling of a 3 x 4 rectangle
```

(continues on next page)

(continued from previous page)

```
sage: next(it)
A wang tiling of a 3 x 4 rectangle
```

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: W = WangTileSolver(tiles,2,2)
sage: list(W.solutions_iterator())
[A wang tiling of a 2 x 2 rectangle,
 A wang tiling of a 2 x 2 rectangle,
 A wang tiling of a 2 x 2 rectangle]
```

With preassigned colors and tiles:

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2), (0,1,2,0)]
sage: t = {(0,1):0}
sage: c = [{},{},{(1,1):0},{}]
sage: W = WangTileSolver(tiles,3,3,preassigned_tiles=t,preassigned_color=c)
sage: S = list(W.solutions_iterator())
sage: [s._table for s in S]
[[[0, 0, 0], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 3], [0, 0, 0], [0, 0, 0]]]
```

With preassigned colors and tiles:

```
sage: right = {(0, 1): 'A', (0, 0): 'A'}
sage: top = {(0, 1): 'B'}
sage: left = {(0, 1): 'A', (0, 0): 'A'}
sage: bottom = {(0, 0): 'B'}
sage: preassigned_color=[right,top,left,bottom]
sage: tiles = ['ABCD', 'EFGH', 'AXCY', 'ABAB', 'EBEB']
sage: W = WangTileSolver(tiles, 1, 2, preassigned_color=preassigned_color)
sage: solutions = list(W.solutions_iterator())
sage: [t.table() for t in solutions]
[[[3, 3]]]
```

solve(*solver=None*, *solver_parameters=None*, *ncpus=1*)

Return a dictionary associating to each tile a list of positions where to find this tile.

INPUT:

- `solver` – string or None (default: `None`), `'dancing_links'` or the name of a MILP solver in Sage like `'GLPK'`, `'Coin'` or `'Gurobi'`.
- `solver_parameters` – dict (default: `{}`), parameters given to the MILP solver using method `solver_parameter`. For a list of available parameters for example for the Gurobi backend, see dictionary `parameters_type` in the file `sage/numerical/backends/gurobi_backend.pyx`
- `ncpus` – integer (default: 1), maximal number of subprocesses to use at the same time, used only if `solver` is `'dancing_links'`.

OUTPUT:

a wang tiling object

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve()
sage: table = tiling._table
sage: table
[[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
```

The tile at position (1,3) is:

```
sage: table[1][3]
0
```

Allowing more threads while using Gurobi:

```
sage: W = WangTileSolver(tiles,3,4)
sage: kwds = dict(Threads=4)
sage: tiling = W.solve(solver='Gurobi', kwds) # optional Gurobi
sage: tiling._table                           # optional Gurobi
[[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
```

Using dancing links:

```
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve(solver='dancing_links', ncpus=8)
sage: tiling
A wang tiling of a 3 x 4 rectangle
```

Using dancing links with tile 2 preassigned at position (0,1):

```
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: preassigned = {(0,1):1}
sage: W = WangTileSolver(tiles,3,3,preassigned_tiles=preassigned)
sage: tiling = W.solve(solver='dancing_links')
sage: tiling._table
[[1, 1, 1], [1, 1, 1], [1, 1, 1]]
```

Using dancing links when constraints are inconsistent:

```
sage: right = {(1,1):1, (2,2):0}
sage: W = WangTileSolver(tiles,3,3,preassigned_color=[right,{},{},{}])
sage: W.solve(solver='dancing_links')
Traceback (most recent call last):
...
ValueError: no solution found using dancing links, the return
value from dancing links solver is None
```

Using SatLP solver:

```
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve('LP')
sage: tiling._table
[[1, 0, 1, 0], [0, 1, 0, 1], [1, 0, 1, 0]]
```

Using SatLP solver with preassigned tiles:

```
sage: preassigned = {(0,0):0}
sage: W = WangTileSolver(tiles,3,4,preassigned_tiles=preassigned)
sage: tiling = W.solve(solver='LP')
sage: tiling._table
[[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
```

Using cryptominisat solver:

```
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve('cryptominisat')  # optional cryptominisat
sage: tiling._table                      # optional cryptominisat
[[1, 0, 1, 0], [0, 1, 0, 1], [1, 0, 1, 0]]
```

REFERENCES:

How do I set solver_parameter to make Gurobi use more than one processor?, https://ask.sagemath.org/question/37726/

vertical_alphabet()

class slabbe.wang_tiles.**WangTiling**(*table*, *tiles*, *color=None*)

Bases: `object`

INPUT:

- `table` – list of lists
- `tiles` – list of tiles, a tile is a 4-tuple (right color, top color, left color, bottom color)
- `color` – dict (default: None)

Note: `table[x][y]` refers to the tile at position (x, y) using the cartesian coordinates. Thus, it is **not** using the matrix-like coordinates.

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: tiling
A wang tiling of a 3 x 4 rectangle
```

Using some blank tiles:

```
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, None, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: tiling
A wang tiling of a 3 x 4 rectangle
```

apply_matrix_transformation(*M*)

INPUT:

- `M` – matrix in SL(2,Z)

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: M = matrix(2, (1,1,0,1))
sage: tiling_M = tiling.apply_matrix_transformation(M)
sage: tiling_M.table()
[[0, None, None, None],
 [1, 1, None, None],
 [0, 0, 0, None],
 [None, 1, 1, 1],
 [None, None, 0, 0],
 [None, None, None, 1]]
```

height()

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: tiles = [(0,3,1,4), (1,4,0,3)]
```

(continues on next page)

(continued from previous page)

```
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: tiling.height()
4
```

horizontal_words_dict(*length*)

Return a dict of horizontal words (left to right) of given length starting at each position (x,y).

INPUT:

- `length` – integer

OUTPUT:

dict position -> word

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: tiling.horizontal_words_dict(2)
{(0, 0): (4, 3),
 (0, 1): (3, 4),
 (0, 2): (4, 3),
 (0, 3): (3, 4),
 (0, 4): (4, 3),
 (1, 0): (3, 4),
 (1, 1): (4, 3),
 (1, 2): (3, 4),
 (1, 3): (4, 3),
 (1, 4): (3, 4)}
```

horizontal_words_list(*side=3*)

Return a list of horizontal words of colors appearing on a given side.

INPUT

- `side` – integer in [0,1,2,3], 3 is for bottom

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve()
sage: tiling.horizontal_words_list()
[[4, 3, 4], [3, 4, 3], [4, 3, 4], [3, 4, 3]]
sage: tiling.horizontal_words_list(0)
[[0, 1, 0], [1, 0, 1], [0, 1, 0], [1, 0, 1]]
```

number_of_occurences(*pattern*, *avoid_border=0*)

Return the number of occurences of the given pattern in the tiling.

INPUT

- `pattern` – dict
- `avoid_border` – integer (default: 0), the size of the border to avoid during the computation

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve()
sage: tiling.number_of_occurences({(0,0):0})
6
sage: tiling.number_of_occurences({(0,0):1})
6
sage: tiling.number_of_occurences({(0,0):1, (1,0):1})
0
sage: tiling.number_of_occurences({(0,0):1, (1,0):1, (0,1):1})
0
sage: tiling.number_of_occurences({(0,0):1, (1,0):0, (0,1):0})
3
```

The pattern is translation invariant:

```
sage: tiling.number_of_occurences({(0,-1):1})
6
sage: tiling.number_of_occurences({(-1,-1):1})
6
sage: tiling.number_of_occurences({(-100,-100):1})
6
```

The x coordinates of the pattern corresponds to the x coordinates when you plot it:

```
sage: tiles = [(0,3,0,4), (1,4,1,3)]
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve()
sage: tiling.number_of_occurences({(0,0):1})
6
sage: tiling.number_of_occurences({(0,0):1, (1,0):1})
4
sage: tiling.number_of_occurences({(0,0):1, (0,1):1})
0
sage: tiling.tikz().pdf(view=False)   # not tested
```

When avoiding the border:

```
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve()
sage: tiling.number_of_occurences({(0,0):0}, avoid_border=1)
1
```

pattern_occurrences(*shape*, *avoid_border=0*)

Return the number of occurences of every pattern having a given shape.

INPUT

- `shape` – list, list of coordinates
- `avoid_border` – integer (default: 0), the size of the border to avoid during the computation

OUTPUT

a dict where each key is a tuple giving the tiles at each coordinate of the shape (in the same order) and values are integers

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: table = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
sage: tiles = [(0, 0, 0, 0), (1, 1, 1, 1), (2, 2, 2, 2)]
```

(continues on next page)

(continued from previous page)

```
sage: tiling = WangTiling(table, tiles)
sage: tiling.pattern_occurrences([(0,0)])
Counter({(0,): 12})
```

```
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: tiling.pattern_occurrences([(0,0)])
Counter({(0,): 6, (1,): 6})
sage: tiling.pattern_occurrences([(0,0), (1,0), (0,1)])
Counter({(1, 0, 0): 3, (0, 1, 1): 3})
```

When avoiding the border:

```
sage: tiling.pattern_occurrences([(0,0)], avoid_border=1)
Counter({(0,): 1, (1,): 1})
sage: tiling.pattern_occurrences([(0,0)], avoid_border=2)
Counter()
```

plot_points_on_torus(*M*, *pointsize=5*, *color_dict=None*)

Plot points modulo some values in x and y.

INPUT

- `M` – M is the matrix projection to $\mathbb{R}^2/\mathbb{Z}^2$
- `pointsize` – positive real number (default:5)
- `color_dict` – dict, tile index -> color or None (default:None)

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: z = polygen(QQ, 'z')
sage: K.<phi> = NumberField(z^2-z-1, 'phi', embedding=AA(golden_ratio))
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: T = WangTiling(table, tiles)
sage: M = matrix(2, [phi, 0, 0, 0.01])
sage: G = T.plot_points_on_torus(M)
```

slide(*shift*, *x0=None*, *y0=1*)

INPUT:

- `shift` – integer
- `x0` – integer or None, every tile at (x,y) such that x>=x0 will be shifted by (0,shift)
- `y0` – integer or None, every tile at (x,y) such that y>=y0 will be shifted by (shift,0)

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: tiling.slide(0).table()
[[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling.slide(3).table()
[[0, None, None, None],
 [1, None, None, None],
 [0, None, None, None],
 [None, 1, 0, 1],
 [None, 0, 1, 0],
```

(continues on next page)

(continued from previous page)

```
 [None, 1, 0, 1]]
sage: tiling.slide(2, x0=2).table()
[[0, 1, 0, 1, None, None], [1, 0, 1, 0, None, None], [None, None, 0, 1, 0, 1]]
sage: tiling.slide(-2, x0=2).table()
[[None, None, 0, 1, 0, 1], [None, None, 1, 0, 1, 0], [0, 1, 0, 1, None, None]]
```

table()

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: tiling.table()
[[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
```

tikz(*color=None*, *font='\\normalsize'*, *rotate=None*, *id=True*, *id_color=''*, *id_format='{}'*, *label=True*, *label_shift=0.2*, *label_color='black'*, *scale=1*, *size=1*, *edges=True*, *draw_H=None*, *draw_V=None*, *extra_before=''*, *extra_after=''*)

Return a tikzpicture showing one solution.

INPUT:

- `color` – None or dict from tile values -> tikz colors
- `font` – string (default: `r'\normalsize'`
- `rotate` – list or `None` (default:`None`) list of four angles in degrees like `(0,0,0,0)`, the rotation angle to apply to each label of Wang tiles. If `None`, it performs a 90 degres rotation for left and right labels taking more than one character.
- `id` – boolean (default: `True`), presence of the tile id
- `id_color` – string (default: `''`)
- `id_format` – string (default: `r'{}'`) to be called with `id_format.format(key)`
- `edges` – bool (default: `True`)
- `label` – boolean (default: `True`), presence of the color labels
- `label_shift` – number (default: `.2`) translation distance of the label from the edge
- `label_color` – string (default: `'black'`)
- `scale` – number (default: 1), tikzpicture scale
- `size` – number (default: 1) size of tiles
- `draw_H` – dict (default: `None`) from tile values -> tikz draw commands. If `None` the values of the dict get replaced by straight lines, more precisely by `r'\draw {} -- ++ (1,0);'`. Dict values must be strings `s` such that `s.format((x,y))` works.
- `draw_V` – dict (default: `None`) from tile values -> tikz draw commands. If `None` the values of the dict get replaced by straight lines, more precisely by `r'\draw {} -- ++ (0,1);'`. Dict values must be strings `s` such that `s.format((x,y))` works.
- `extra_before` – string (default: `''`) extra lines of tikz code to add at the start
- `extra_after` – string (default: `''`) extra lines of tikz code to add at the end

EXAMPLES:

```
sage: from slabbe import WangTileSolver
sage: tiles = [(0,0,0,0), (1,1,1,1), (2,2,2,2)]
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve()
sage: t = tiling.tikz()
sage: t
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\begin{document}
\begin{tikzpicture}[scale=1]
\tikzstyle{every node}=[font=\normalsize]
% tile at position (x,y)=(0, 0)
\node[] at (0.5, 0.5) {0};
\draw (0, 0) -- ++ (0,1);
...
... 96 lines not printed (3570 characters in total) ...
...
\node[rotate=0,black] at (2.8, 3.5) {0};
\node[rotate=0,black] at (2.5, 3.8) {0};
\node[rotate=0,black] at (2.2, 3.5) {0};
\node[rotate=0,black] at (2.5, 3.2) {0};
\end{tikzpicture}
\end{document}
```

With colors:

```
sage: tiles = [(0,2,1,3), (1,3,0,2)]
sage: color = {0:'white',1:'red',2:'blue',3:'green'}
sage: W = WangTileSolver(tiles,3,4,color=color)
sage: tiling = W.solve()
sage: t = tiling.tikz()
```

With colors, alternatively:

```
sage: tiles = [(0,2,1,3), (1,3,0,2)]
sage: W = WangTileSolver(tiles,3,4)
sage: tiling = W.solve('GLPK')
sage: color = {0:'white',1:'red',2:'blue',3:'green'}
sage: t = tiling.tikz(color=color)
```

Using some blank tiles:

```
sage: from slabbe import WangTiling
sage: tiles = [(0,3,1,2), (1,2,0,3)]
sage: table = [[0, 1, None, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: color = {0:'white',1:'red',2:'blue',3:'green'}
sage: tiling = WangTiling(table, tiles, color)
sage: t = tiling.tikz()
```

Testing the options:

```
sage: tiles = [(0,3,1,2), (1,2,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: color = {0:'white',1:'red',2:'blue',3:'green'}
sage: t = WangTiling(table, tiles, color).tikz(font=r'\Huge')
sage: t = WangTiling(table, tiles, color).tikz(rotate=(0,90,0,0))
sage: t = WangTiling(table, tiles, color).tikz(label_shift=.05)
sage: t = WangTiling(table, tiles, color).tikz(scale=4)
```

```
sage: m = matrix(2,[1,1,0,1])
sage: t = WangTiling(table, tiles, color).apply_matrix_transformation(m).tikz()
```

Using puzzle boundary instead of colors:

```
sage: tiles = [(0,3,1,2), (1,2,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: t = WangTiling(table, tiles)
sage: draw_H = {0:r'\draw {} -- ++ (1/2,.2) -- ++ (1/2,-.2);',
....:           1:r'\draw {} -- ++ (1/2,.2) -- ++ (1/2,-.2);',
....:           2:r'\draw {} -- ++ (1/2,.2) -- ++ (1/2,-.2);',
....:           3:r'\draw {} -- ++ (1/2,.2) -- ++ (1/2,-.2);'}
sage: v = r'\draw {} -- ++ (0,.4) -- ++ (.2,0) -- ++ (0,.2) -- ++ (-.2,0) -- ++ (0,.4);'
sage: draw_V = {0:v, 1:v, 2:v, 3:v}
sage: tikz = t.tikz(label=False, draw_H=draw_H, draw_V=draw_V)
```

tile_frequency(*avoid_border=1*)

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: tiling.tile_frequency()
{(0,): 1/2,
 (1,): 1/2}
```

tile_positions(*M*)

Return the list of positions where tile of M appear.

INPUT:

- `M` – subset of tile indices

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: tiling.tile_positions([0])
[(0, 0), (0, 2), (1, 1), (1, 3), (2, 0), (2, 2)]
```

TESTS:

```
sage: tiling.tile_positions([])
[]
```

transpose()

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: M = matrix(2, (1,1,0,1))
sage: tiling_T = tiling.transpose()
sage: tiling_T.table()
[[0, 1, 0], [1, 0, 1], [0, 1, 0], [1, 0, 1]]
```

vertical_words_dict(*length*)

Return a dict of vertical words (bottom to top) of given length starting at each position (x,y).

INPUT:

- `length` – integer

OUTPUT:

dict position -> word

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: tiling.vertical_words_dict(2)
{(0, 0): (1, 0),
 (0, 1): (0, 1),
 (0, 2): (1, 0),
 (1, 0): (0, 1),
 (1, 1): (1, 0),
 (1, 2): (0, 1),
 (2, 0): (1, 0),
 (2, 1): (0, 1),
 (2, 2): (1, 0),
 (3, 0): (0, 1),
 (3, 1): (1, 0),
 (3, 2): (0, 1)}
sage: tiling.vertical_words_dict(3)
{(0, 0): (1, 0, 1),
 (0, 1): (0, 1, 0),
 (1, 0): (0, 1, 0),
 (1, 1): (1, 0, 1),
 (2, 0): (1, 0, 1),
 (2, 1): (0, 1, 0),
 (3, 0): (0, 1, 0),
 (3, 1): (1, 0, 1)}
```

width()

EXAMPLES:

```
sage: from slabbe import WangTiling
sage: tiles = [(0,3,1,4), (1,4,0,3)]
sage: table = [[0, 1, 0, 1], [1, 0, 1, 0], [0, 1, 0, 1]]
sage: tiling = WangTiling(table, tiles)
sage: tiling.width()
3
```

slabbe.wang_tiles.**fusion**(*tile0*, *tile1*, *direction*, *function=<slot wrapper '__add__' of 'str' objects>*, *initial=''*)

Return the fusion of wang tile sets in the given direction.

We keep only the strongly connected components.

INPUT:

- `tile0` – 4-uple
- `tile1` – 4-uple
- `direction` – integer (1 or 2)
- **`function` – function (default:`str.__add__`), monoid** operation
- `initial` – object (default:`''`), monoid neutral

EXAMPLES:

```
sage: from slabbe.wang_tiles import fusion
sage: t0 = 'abcd'
sage: t1 = 'xyaz'
sage: fusion(t0,t1,1)
('x', 'by', 'c', 'dz')
```

```
sage: t0 = 'abcd'
sage: t1 = 'xyzb'
sage: fusion(t0,t1,2)
('ax', 'y', 'cz', 'd')
```

TESTS:

```
sage: t0 = 'abcd'
sage: t1 = 'efgh'
sage: fusion(t0,t1,1)
Traceback (most recent call last):
...
AssertionError: A must be equal to Y
```

`slabbe.wang_tiles.`**`tile_to_tikz`**(*tile*, *position*, *color=None*, *id=None*, *id_color=''*, *id_format='{}'*, *sizex=1*, *sizey=1*, *rotate=None*, *label=True*, *label_shift=0.2*, *label_color='black'*, *right_edges=True*, *top_edges=True*, *left_edges=True*, *bottom_edges=True*, *draw_H=None*, *draw_V=None*)

INPUT:

- `tile` – tuple of length 4
- `position` – tuple of two numbers
- `color` – dict (default: `None`) from tile values -> tikz colors
- `id` – id (default: `None`) of the tile to be printed in the center
- `id_color` – string (default: `''`)
- `id_format` – string (default: `r'{}'`) to be called with `id_format.format(key)`
- `sizex` – number (default: `1`), horizontal size of the tile
- `sizey` – number (default: `1`), vertical size of the tile
- `rotate` – list or `None` (default:`None`) list of four angles in degrees like `(0,0,0,0)`, the rotation angle to apply to each label of Wang tiles. If `None`, it performs a 90 degres rotation for left and right labels taking more than one character.
- `label` – boolean (default: `True`)
- `label_shift` – number (default: `.2`) translation distance of the label from the edge
- `label_color` – string (default: `'black'`)
- `right_edges` – bool (default: `True`)
- `top_edges` – bool (default: `True`)
- `left_edges` – bool (default: `True`)
- `bottom_edges` – bool (default: `True`)
- `draw_H` – dict (default: `None`) from tile values -> tikz draw commands. If `None` the values of the dict get replaced by straight lines, more precisely by `r'\draw {{}} -- ++ (1,0);'`. Dict values must be strings s such that `s.format((x,y))` works.
- `draw_V` – dict (default: `None`) from tile values -> tikz draw commands. If `None` the values of the dict get replaced by straight lines, more precisely by `r'\draw {{}} -- ++ (0,1);'`. Dict values must be strings s such that `s.format((x,y))` works.

OUTPUT:

- list of strings

EXAMPLES:

```
sage: from slabbe.wang_tiles import tile_to_tikz
sage: color = {0:'white',1:'red',2:'cyan',3:'green',4:'white'}
sage: tile_to_tikz((1,2,3,4), (10,100), color)
['% tile at position (x,y)=(10, 100)',
 '\\fill[red] (11, 100) -- (10.5, 100.5) -- (11, 101);',
 '\\fill[cyan] (10, 101) -- (10.5, 100.5) -- (11, 101);',
 '\\fill[green] (10, 100) -- (10.5, 100.5) -- (10, 101);',
 '\\fill[white] (10, 100) -- (10.5, 100.5) -- (11, 100);',
 '\\draw (11, 100) -- ++ (0,1);',
 '\\draw (10, 101) -- ++ (1,0);',
 '\\draw (10, 100) -- ++ (0,1);',
 '\\draw (10, 100) -- ++ (1,0);',
 '\\node[rotate=0,black] at (10.8, 100.5) {1};',
 '\\node[rotate=0,black] at (10.5, 100.8) {2};',
 '\\node[rotate=0,black] at (10.2, 100.5) {3};',
 '\\node[rotate=0,black] at (10.5, 100.2) {4};']
sage: tile_to_tikz((1,2,3,4), (10,100), color=None)
['% tile at position (x,y)=(10, 100)',
 '\\draw (11, 100) -- ++ (0,1);',
 '\\draw (10, 101) -- ++ (1,0);',
 '\\draw (10, 100) -- ++ (0,1);',
 '\\draw (10, 100) -- ++ (1,0);',
 '\\node[rotate=0,black] at (10.8, 100.5) {1};',
 '\\node[rotate=0,black] at (10.5, 100.8) {2};',
 '\\node[rotate=0,black] at (10.2, 100.5) {3};',
 '\\node[rotate=0,black] at (10.5, 100.2) {4};']
sage: tile_to_tikz((1,2,3,4), (10,100), color=None, rotate=(0,90,0,0))
['% tile at position (x,y)=(10, 100)',
 '\\draw (11, 100) -- ++ (0,1);',
 '\\draw (10, 101) -- ++ (1,0);',
 '\\draw (10, 100) -- ++ (0,1);',
 '\\draw (10, 100) -- ++ (1,0);',
 '\\node[rotate=0,black] at (10.8, 100.5) {1};',
 '\\node[rotate=90,black] at (10.5, 100.8) {2};',
 '\\node[rotate=0,black] at (10.2, 100.5) {3};',
 '\\node[rotate=0,black] at (10.5, 100.2) {4};']
sage: tile_to_tikz((1,2,3,4), (10,100), color=None, label_shift=.1)
['% tile at position (x,y)=(10, 100)',
 '\\draw (11, 100) -- ++ (0,1);',
 '\\draw (10, 101) -- ++ (1,0);',
 '\\draw (10, 100) -- ++ (0,1);',
 '\\draw (10, 100) -- ++ (1,0);',
 '\\node[rotate=0,black] at (10.9000000000000, 100.5) {1};',
 '\\node[rotate=0,black] at (10.5, 100.900000000000) {2};',
 '\\node[rotate=0,black] at (10.1000000000000, 100.5) {3};',
 '\\node[rotate=0,black] at (10.5, 100.100000000000) {4};']
```

```
sage: tile_to_tikz((10,20,30,40), (10,100), color=None)
['% tile at position (x,y)=(10, 100)',
 '\\draw (11, 100) -- ++ (0,1);',
 '\\draw (10, 101) -- ++ (1,0);',
 '\\draw (10, 100) -- ++ (0,1);',
 '\\draw (10, 100) -- ++ (1,0);',
 '\\node[rotate=90,black] at (10.8, 100.5) {10};',
 '\\node[rotate=0,black] at (10.5, 100.8) {20};',
 '\\node[rotate=90,black] at (10.2, 100.5) {30};',
 '\\node[rotate=0,black] at (10.5, 100.2) {40};']
```

CHAPTER

FIVE

MISCELLANEOUS

5.1 Analyze Sage Build

A time evolution picture of packages built by Sage

EXAMPLES:

```
sage: from slabbe.analyze_sage_build import draw_sage_build
sage: t = draw_sage_build()
sage: _ = t.pdf(view=False)
```

AUTHOR:

- Sébastien Labbé, December 9-11, 2016

slabbe.analyze_sage_build.**build_duration_logs**(*path_to_file*, *pattern=None*)

INPUT:

- `path_to_file` – string, file to parse
- `pattern` – string or `None`, string to parse. If `None`, it returns 0 seconds.

OUTPUT:

list of timedelta objects

EXAMPLES:

```
sage: L = [ ('/../ptestlong.log', 'Total time for all tests: '),
....:       ('/../dochtml.log', 'Elapsed time: '),
....:       ('/../start.log', None),
....:       ('/sagelib-7.5.beta6.log', 'real\t'),
....:       ('/sqlite.log', 'real\t')]
```

```
sage: from slabbe.analyze_sage_build import build_duration_logs
sage: from sage.env import SAGE_ROOT
sage: import os
sage: SAGE_LOGS_PKGS = os.path.join(SAGE_ROOT,'logs','pkgs')
sage: import os
sage: for (file, pattern) in L:                  # random
....:     print(file)
....:     if not os.path.exists(SAGE_LOGS_PKGS+file):
....:         continue
....:     build_duration_logs(SAGE_LOGS_PKGS+file, pattern)
/../ptestlong.log
/../dochtml.log
[datetime.timedelta(0, 471, 700000)]
/../start.log
[]
```

(continues on next page)

(continued from previous page)

```
/sagelib-7.5.beta6.log
/sqlite.log
[]
```

slabbe.analyze_sage_build.**draw_sage_build**(*start=None, stop=None, consider='last', verbose=False*)

Return a time evolution picture of packages built by Sage

INPUT:

- `start` – datetime object (default is January 1st, 2000)
- `stop` – datetime object (default is now)
- `consider` - string (default: `'last'`), `'last'` or `'all'`, in any log file, consider the last duration found or the sum of all of them
- `verbose` - boolean (default: `False`)

OUTPUT:

TikzPicture object

EXAMPLES:

```
sage: from slabbe.analyze_sage_build import draw_sage_build
sage: t = draw_sage_build()
```

During the previous 7 day:

```
sage: import datetime
sage: stop = datetime.datetime.now()
sage: start = stop - datetime.timedelta(7r)
sage: t = draw_sage_build(start, stop)        # not tested
```

```
sage: t = draw_sage_build(stop=datetime.datetime(2016,12,9)) # not tested
```

TESTS:

It may fail if nothing is found during the period:

```
sage: t = draw_sage_build()  # not tested
Traceback (most recent call last):
...
ValueError: no package found built between start date (=2016-12-08
16:58:35.189127) and stop date (=2016-12-09 16:58:35.189127). Oldest is
2016-02-16 11:38:00.673327. Newest is 2016-12-08 11:27:03.312340.
```

slabbe.analyze_sage_build.**last_modified_datetime**(*path_to_file*)

EXAMPLES:

```
sage: from slabbe.analyze_sage_build import last_modified_datetime
sage: from sage.env import SAGE_ROOT
sage: last_modified_datetime(SAGE_ROOT+'/README.md')
datetime.datetime(..., ..., ..., ..., ..., ...)
sage: last_modified_datetime(SAGE_ROOT+'/VERSION.txt')
datetime.datetime(..., ..., ..., ..., ..., ...)
```

slabbe.analyze_sage_build.**sage_logs_datetime_list**(*consider='last', verbose=False*)

Return a dictionnary of duration and last modified information from the sage log files.

INPUT:

- consider - string (default: 'last'), 'last' or 'all', in any log file, consider the last duration found or the sum of all of them
- verbose - boolean (default: False)

EXAMPLES:

```
sage: import datetime
sage: from slabbe.analyze_sage_build import sage_logs_datetime_list
sage: L = sage_logs_datetime_list()
sage: L = sage_logs_datetime_list()
sage: stop = datetime.datetime.now()
sage: start = stop - datetime.timedelta(14r)
sage: L_filtered = [(A,B,delta,file) for (A,B,delta,file) in L
....:                   if start <= A and B <= stop]
```

5.2 Tikz Picture

TikzPicture

A Python Module for tikz pictures. A TikzPicture object is created from a string starting with r'\begin{tikzpicture}' and ending with r'\end{tikzpicture}'.

The module allows easy creation of tikz pictures from Sage objects like graphs and posets. Conversion of tikz pictures to pdf and png format based on standalone LaTeX document class.

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: V = [[1,0,1],[1,0,0],[1,1,0],[0,0,-1],[0,1,0],[-1,0,0],[0,1,1],[0,0,1],[0,-1,0]]
sage: P = Polyhedron(vertices=V).polar()
sage: s = P.projection().tikz([674,108,-731],112)
sage: t = TikzPicture(s)
```

Creation of a pdf in a temporary directory. The returned value is a string giving the file path:

```
sage: path_to_file = t.pdf(view=False)   # long time (2s)
```

Setting view=True, which is the default, opens the pdf in a viewer.

```
sage: t
\documentclass[tikz]{standalone}
\usepackage{amsmath}
\begin{document}
\begin{tikzpicture}%
        [x={(0.249656cm, -0.577639cm)},
        y={(0.777700cm, -0.358578cm)},
        z={(-0.576936cm, -0.733318cm)},
        scale=2.000000,
...
... 80 lines not printed (4889 characters in total) ...
...
\node[vertex] at (1.00000, 1.00000, -1.00000)     {};
\node[vertex] at (1.00000, 1.00000, 1.00000)     {};
%%
%%
\end{tikzpicture}
\end{document}
```

Use print t to see the complete content of the file.

Adding a border avoids croping the vertices of a graph:

```
sage: g = graphs.PetersenGraph()
sage: s = latex(g)   # takes 3s but the result is cached
sage: t = TikzPicture(s, standalone_options=["border=4mm"], usepackage=['tkz-graph'])
sage: _ = t.pdf()    # not tested
```

If dot2tex Sage optional package and graphviz are installed, then the following one liner works:

```
sage: t = TikzPicture.from_graph(g)  # optional: dot2tex # long time (3s)
```

```
sage: s = latex(transducers.GrayCode())
sage: t = TikzPicture(s, usetikzlibrary=['automata'])
sage: _ = t.pdf(view=False)  # long time (2s)
```

AUTHORS:

- Sébastien Labbé, initial version in slabbe-0.2.spkg, nov 2015.

class `slabbe.tikz_picture.`**`StandaloneTex`**(*content, standalone_options=None, usepackage=['amsmath'], usetikzlibrary=None, macros=None, use_sage_preamble=False*)

Bases: `sage.structure.sage_object.SageObject`

See the class documentation for full information.

EXAMPLES:

```
sage: from slabbe import StandaloneTex
sage: content = "\\section{Intro}\n\nTest\n"
sage: t = StandaloneTex(content)
```

```
sage: from slabbe import TikzPicture
sage: s = "\\begin{tikzpicture}\n\\draw (0,0) -- (1,1);\n\\end{tikzpicture}"
sage: t = TikzPicture(s)
```

`content`()

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: s = "\\begin{tikzpicture}\n\\draw (0,0) -- (1,1);\n\\end{tikzpicture}"
sage: t = TikzPicture(s)
sage: print(t.tikz_picture_code())
\begin{tikzpicture}
\draw (0,0) -- (1,1);
\end{tikzpicture}
```

`pdf`(*filename=None, view=True*)

Compiles the latex code with pdflatex and create a pdf file.

INPUT:

- `filename` – string (default:`None`), the output filename. If `None`, it saves the file in a temporary directory.
- `view` – bool (default:`True`), whether to open the file in a pdf viewer. This option is ignored and automatically set to `False` if `filename` is not `None`.

OUTPUT:

string, path to pdf file

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: V = [[1,0,1],[1,0,0],[1,1,0],[0,0,-1],[0,1,0],[-1,0,0],[0,1,1],[0,0,1],[0,-1,0]]
sage: P = Polyhedron(vertices=V).polar()
sage: s = P.projection().tikz([674,108,-731],112)
sage: t = TikzPicture(s)
sage: _ = t.pdf()    # not tested
```

```
sage: from sage.misc.temporary_file import tmp_filename
sage: filename = tmp_filename('temp','.pdf')
sage: _ = t.pdf(filename)   # long time (2s)
```

ACKNOWLEDGEMENT:

The code was adapted and taken from the module `sage.misc.latex.py`.

png(*filename=None*, *density=150*, *view=True*)

Compiles the latex code with pdflatex and converts to a png file.

INPUT:

- `filename` – string (default:`None`), the output filename. If `None`, it saves the file in a temporary directory.
- `density` – integer, (default: `150`), horizontal and vertical density of the image
- `view` – bool (default:`True`), whether to open the file in a png viewer. This option is ignored and automatically set to `False` if `filename` is not `None`.

OUTPUT:

string, path to png file

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: V = [[1,0,1],[1,0,0],[1,1,0],[0,0,-1],[0,1,0],[-1,0,0],[0,1,1],[0,0,1],[0,-1,0]]
sage: P = Polyhedron(vertices=V).polar()
sage: s = P.projection().tikz([674,108,-731],112)
sage: t = TikzPicture(s)
sage: _ = t.png()    # not tested
```

```
sage: from sage.misc.temporary_file import tmp_filename
sage: filename = tmp_filename('temp','.png')
sage: _ = t.png(filename)      # long time (2s)
```

ACKNOWLEDGEMENT:

The code was adapted and taken from the module `sage.misc.latex.py`.

svg(*filename=None*, *view=True*)

Compiles the latex code with pdflatex and converts to a svg file.

INPUT:

- `filename` – string (default:`None`), the output filename. If `None`, it saves the file in a temporary directory.
- `view` – bool (default:`True`), whether to open the file in a browser. This option is ignored and automatically set to `False` if `filename` is not `None`.

OUTPUT:

string, path to svg file

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: V = [[1,0,1],[1,0,0],[1,1,0],[0,0,-1],[0,1,0],[-1,0,0],[0,1,1],[0,0,1],[0,-1,0]]
sage: P = Polyhedron(vertices=V).polar()
sage: s = P.projection().tikz([674,108,-731],112)
sage: t = TikzPicture(s)
sage: _ = t.svg()    # not tested
```

```
sage: from sage.misc.temporary_file import tmp_filename
sage: filename = tmp_filename('temp','.svg')
sage: _ = t.svg(filename)      # long time (2s)
```

ACKNOWLEDGEMENT:

The code was adapted and taken from the module `sage.misc.latex.py`.

tex(*filename=None*, *include_header=True*)
Writes the latex code to a file.

INPUT:

- `filename` – string (default:`None`), the output filename. If `None`, it saves the file in a temporary directory.
- `include_header` – bool (default:`True`) whether to include the header latex part. If `False`, it prints only the tikzpicture part to the file.

OUTPUT:

string, path to tex file

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: V = [[1,0,1],[1,0,0],[1,1,0],[0,0,-1],[0,1,0],[-1,0,0],[0,1,1],[0,0,1],[0,-1,0]]
sage: P = Polyhedron(vertices=V).polar()
sage: s = P.projection().tikz([674,108,-731],112)
sage: t = TikzPicture(s)
sage: _ = t.tex()
```

Write only the tikzpicture without header and begin/end document:

```
sage: _ = t.tex(include_header=False)
```

Write to a given filename:

```
sage: from sage.misc.temporary_file import tmp_filename
sage: filename = tmp_filename('temp','.tex')
sage: _ = t.tex(filename)
```

class `slabbe.tikz_picture.`**TikzPicture**(*content*, *standalone_options=None*, *usepackage=['amsmath']*, *usetikzlibrary=None*, *macros=None*, *use_sage_preamble=False*)
Bases: *`slabbe.tikz_picture.StandaloneTex`*

Creates a TikzPicture embedded in a LaTeX standalone document class.

INPUT:

- `code` – string, tikzpicture code starting with `r'\begin{tikzpicture}'` and ending with `r'\end{tikzpicture}'`
- `standalone_options` – list of strings (default: `[]`), latex document class standalone configuration options.

- `usepackage` – list of strings (default: `['amsmath']`), latex packages.
- `usetikzlibrary` – list of strings (default: `[]`), tikz libraries to use.
- `macros` – list of strings (default: `[]`), stuff you need for the picture.
- `use_sage_preamble` – bool (default: `False`), whether to include sage latex preamble and sage latex macros, that is, the content of `sage.misc.latex.extra_preamble()`, `sage.misc.latex.extra_macros()` and `sage.misc.latex_macros.sage_latex_macros()`.

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: g = graphs.PetersenGraph()
sage: s = latex(g)
sage: t = TikzPicture(s, standalone_options=["border=4mm"], usepackage=['tkz-graph'])
sage: _ = t.pdf(view=False)    # long time (2s)
```

Here are standalone configurations, packages, tikz libraries and macros you may want to set:

```
sage: options = ['preview', 'border=4mm', 'beamer', 'float']
sage: usepackage = ['nicefrac', 'amsmath', 'pifont', 'tikz-3dplot',
....:    'tkz-graph', 'tkz-berge', 'pgfplots']
sage: tikzlib = ['arrows', 'snakes', 'backgrounds', 'patterns',
....:      'matrix', 'shapes', 'fit', 'calc', 'shadows', 'plotmarks',
....:      'positioning', 'pgfplots.groupplots', 'mindmap']
sage: macros = [r'\newcommand{\ZZ}{\mathbb{Z}}']
sage: s = "\\begin{tikzpicture}\n\\draw (0,0) -- (1,1);\n\\end{tikzpicture}"
sage: t = TikzPicture(s, standalone_options=options, usepackage=usepackage,
....:        usetikzlibrary=tikzlib, macros=macros)
sage: _ = t.pdf(view=False)    # long time (2s)
```

classmethod from_graph(*graph*, *merge_multiedges=True*, *merge_label_function=<type 'tuple'>*, ***kwds*)

Convert a graph to a tikzpicture using graphviz and dot2tex.

Note: Prerequisite: dot2tex optional Sage package and graphviz must be installed.

INPUT:

- `graph` – graph
- `merge_multiedges` – bool (default: `True`), if the graph has multiple edges, whether to merge the multiedges into one single edge
- `merge_label_function` – function (default:`tuple`), a function to apply to each list of labels to be merged. It is ignored if `merge_multiedges` is not `True` or if the graph has no multiple edges.

Other inputs are used for latex drawing with dot2tex and graphviz:

- `prog` – string (default: `'dot'`) the program used for the layout corresponding to one of the software of the graphviz suite: 'dot', 'neato', 'twopi', 'circo' or 'fdp'.
- `edge_labels` – bool (default: `True`)
- `color_by_label` – bool (default: `False`)
- `rankdir` – string (default: `'down'`)
- `subgraph_clusters` – (default: []) a list of lists of vertices, if supported by the layout engine, nodes belonging to the same cluster subgraph are drawn together, with the entire drawing of the cluster contained within a bounding rectangle.

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: g = graphs.PetersenGraph()
sage: tikz = TikzPicture.from_graph(g) # optional dot2tex # long time (3s)
sage: _ = tikz.pdf()      # not tested
```

Using `prog`:

```
sage: tikz = TikzPicture.from_graph(g, prog='neato', color_by_label=True) # optional dot2tex # long␣
↪time (3s)
sage: _ = tikz.pdf()      # not tested
```

Using `rankdir`:

```
sage: tikz = TikzPicture.from_graph(g, rankdir='right') # optional dot2tex # long time (3s)
sage: _ = tikz.pdf()      # not tested
```

Using `merge_multiedges`:

```
sage: alpha = var('alpha')
sage: m = matrix(2,range(4)); m.set_immutable()
sage: G = DiGraph([(0,1,alpha), (0,1,0), (0,2,9), (0,2,m)], multiedges=True)
sage: tikz = TikzPicture.from_graph(G, merge_multiedges=True) # optional dot2tex
sage: _ = tikz.pdf()      # not tested
```

Using `merge_multiedges` with `merge_label_function`:

```
sage: fn = lambda L: LatexExpr(','.join(map(str, L)))
sage: G = DiGraph([(0,1,'a'), (0,1,'b'), (0,2,'c'), (0,2,'d')], multiedges=True)
sage: tikz = TikzPicture.from_graph(G, merge_multiedges=True,
....:               merge_label_function=fn) # optional dot2tex
sage: _ = tikz.pdf()      # not tested
```

Using subgraphs clusters (broken when using labels, see trac ticket #22070):

```
sage: S = FiniteSetMaps(5)
sage: I = S((0,1,2,3,4))
sage: a = S((0,1,3,0,0))
sage: b = S((0,2,4,1,0))
sage: roots = [I]
sage: succ = lambda v:[v*a,v*b,a*v,b*v]
sage: R = RecursivelyEnumeratedSet(roots, succ)
sage: G = R.to_digraph()
sage: G
Looped multi-digraph on 27 vertices
sage: C = G.strongly_connected_components()
sage: tikz = TikzPicture.from_graph(G, merge_multiedges=False,
....:                               subgraph_clusters=C)
sage: _ = tikz.pdf()      # not tested
```

classmethod from_graph_with_pos(*graph*, *scale=1*, *merge_multiedges=True*, *merge_label_function=<type 'tuple'>*)

Convert a graph with positions defined for vertices to a tikzpicture.

INPUT:

- `graph` – graph (with predefined positions)
- `scale` – number (default:1), tikzpicture scale
- `merge_multiedges` – bool (default: `True`), if the graph has multiple edges, whether to merge the multiedges into one single edge
- `merge_label_function` – function (default:`tuple`), a function to apply to each list of labels to be merged. It is ignored if `merge_multiedges` is not `True` or if the graph has no multiple edges.

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: g = graphs.PetersenGraph()
sage: tikz = TikzPicture.from_graph_with_pos(g)
```

```
sage: edges = [(0,0,'a'),(0,1,'b'),(0,1,'c')]
sage: kwds = dict(format='list_of_edges', loops=True, multiedges=True)
sage: G = DiGraph(edges, **kwds)
sage: G.set_pos({0:(0,0), 1:(1,0)})
sage: f = lambda label:','.join(label)
sage: TikzPicture.from_graph_with_pos(G, merge_label_function=f)
\documentclass[tikz]{standalone}
\standaloneconfig{border=4mm}
\usepackage{amsmath}
\begin{document}
\begin{tikzpicture}
[auto,scale=1]
% vertices
\node (node_0) at (0, 0) {0};
\node (node_1) at (1, 0) {1};
% edges
\draw[->] (node_0) -- node {b,c} (node_1);
% loops
\draw (node_0) edge [loop above] node {a} ();
\end{tikzpicture}
\end{document}
```

TESTS:

```
sage: edges = [(0,0,'a'),(0,1,'b'),(0,1,'c')]
sage: kwds = dict(format='list_of_edges', loops=True, multiedges=True)
sage: G = DiGraph(edges, **kwds)
sage: TikzPicture.from_graph_with_pos(G)
Traceback (most recent call last):
...
ValueError: vertex positions need to be set first
```

classmethod from_poset(*poset*, ***kwds*)

Convert a poset to a tikzpicture using graphviz and dot2tex.

Note: Prerequisite: dot2tex optional Sage package and graphviz must be installed.

INPUT:

- `poset` – poset
- `prog` – string (default: `'dot'`) the program used for the layout corresponding to one of the software of the graphviz suite: 'dot', 'neato', 'twopi', 'circo' or 'fdp'.
- `edge_labels` – bool (default: `True`)
- `color_by_label` – bool (default: `False`)
- `rankdir` – string (default: `'down'`)

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: P = posets.PentagonPoset()
sage: tikz = TikzPicture.from_poset(P) # optional dot2tex # long time (3s)
sage: tikz = TikzPicture.from_poset(P, prog='neato', color_by_label=True) # optional dot2tex # long␣
↪time (3s)
```

```
sage: P = posets.SymmetricGroupWeakOrderPoset(4)
sage: tikz = TikzPicture.from_poset(P) # optional dot2tex # long time (4s)
sage: tikz = TikzPicture.from_poset(P, prog='neato') # optional dot2tex # long time (4s)
```

tikz_picture_code()

EXAMPLES:

```
sage: from slabbe import TikzPicture
sage: s = "\\begin{tikzpicture}\n\\draw (0,0) -- (1,1);\n\\end{tikzpicture}"
sage: t = TikzPicture(s)
sage: print(t.tikz_picture_code())
\begin{tikzpicture}
\draw (0,0) -- (1,1);
\end{tikzpicture}
```

5.3 Ranking Scale for Ultimate Frisbee

Ranking Scale

EXEMPLES:

```
sage: from slabbe.ranking_scale import *
sage: R = RankingScale_CQU4_2011()
sage: R = RankingScale_USAU_2013()
sage: R = RankingScale_CQU4_2014()
```

AUTHOR :

- Sébastien Labbé, Fall 2011, first version

class slabbe.ranking_scale.**RankingScale**(*scale_names*, *scales*)

Bases: object

INPUT:

- scale_names – iterable of str
- scales – iterable of list

NOTE: the ordering of both input must match

length()

EXEMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2011
sage: R = RankingScale_CQU4_2011()
sage: R.length()
105
```

plot(*pointsize=20*)

pointage_csv(*filename='pointage.csv'*, *dialect='excel'*)

EXEMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2011
sage: R = RankingScale_CQU4_2011()
sage: R.pointage_csv()          # not tested
Creation of file pointage.csv
```

table()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2011
sage: R = RankingScale_CQU4_2011()
sage: R.table()
  Position    Grand Chelem   Mars Attaque   La Flotte   Petit Chelem
  1           1000           1000           800         400
  2           938            938            728         355
  3           884            884            668         317
  4           835            835            614         285
  5           791            791            566         256
  6           750            750            522         230
  7           711            711            482         206
  8           675            675            444         184
  9           641            641            409         164
  10          609            609            377         146
...
  95          0              11             0           0
  96          0              10             0           0
  97          0              9              0           0
  98          0              8              0           0
  99          0              7              0           0
  100         0              6              0           0
  101         0              4              0           0
  102         0              3              0           0
  103         0              2              0           0
  104         0              1              0           0
```

slabbe.ranking_scale.**RankingScale_CQU4_2011**()

EXEMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2011
sage: R = RankingScale_CQU4_2011()
sage: R.table()
  Position    Grand Chelem   Mars Attaque   La Flotte   Petit Chelem
  1           1000           1000           800         400
  2           938            938            728         355
  3           884            884            668         317
  4           835            835            614         285
  5           791            791            566         256
  6           750            750            522         230
  7           711            711            482         206
  8           675            675            444         184
  9           641            641            409         164
  10          609            609            377         146
...
  95          0              11             0           0
  96          0              10             0           0
  97          0              9              0           0
  98          0              8              0           0
  99          0              7              0           0
  100         0              6              0           0
  101         0              4              0           0
  102         0              3              0           0
  103         0              2              0           0
  104         0              1              0           0
```

slabbe.ranking_scale.**RankingScale_CQU4_2014**(*R=1*, *base=e*)

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2014
sage: R = RankingScale_CQU4_2014()
```

slabbe.ranking_scale.**RankingScale_CQU4_2015_v1**()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2015_v1
sage: R = RankingScale_CQU4_2015_v1()
```

slabbe.ranking_scale.**RankingScale_CQU4_2015_v2**()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2015_v2
sage: R = RankingScale_CQU4_2015_v2()
```

slabbe.ranking_scale.**RankingScale_CQU4_2015_v3**()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2015_v3
sage: R = RankingScale_CQU4_2015_v3()
```

slabbe.ranking_scale.**RankingScale_CQU4_2015_v4**()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2015_v4
sage: R = RankingScale_CQU4_2015_v4()
```

slabbe.ranking_scale.**RankingScale_CQU4_2015_v5**()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_CQU4_2015_v5
sage: R = RankingScale_CQU4_2015_v5()
```

slabbe.ranking_scale.**RankingScale_USAU_2013**()

EXAMPLES:

```
sage: from slabbe.ranking_scale import RankingScale_USAU_2013
sage: R = RankingScale_USAU_2013()
sage: R.table()
  Position   Serie 1500   Serie 1000   Serie 500   Serie 250
  1          1500         1000         500         250
  2          1366         911          455         228
  3          1252         835          417         209
  4          1152         768          384         192
  5          1061         708          354         177
  6          979          653          326         163
  7          903          602          301         151
  8          832          555          278         139
  9          767          511          256         128
  10         706          471          236         118
  11         648          432          217         109
  12         595          397          199         100
  13         544          363          182         91
  14         497          332          166         83
  15         452          302          151         76
  16         410          274          137         69
  17         371          248          124         62
  18         334          223          112         56
  19         299          199          100         50
  20         266          177          89          45
  21         235          157          79          40
  22         206          137          69          35
  23         178          119          60          30
  24         152          102          51          26
  25         128          86           43          22
  26         106          71           36          18
  27         85           57           29          15
  28         66           44           22          12
```

(continues on next page)

(continued from previous page)

```
29          47          32          16          9
30          31          21          11          6
31          15          10          6           3
32          1           1           1           1
```

slabbe.ranking_scale.**curve**(*nb_equipes*, *max_points=100*, *K=1*, *R=2*, *base=2*, *verbose=False*)

INPUT:

- `nb_equipes` – integer
- `max_points` – integer
- `K` – the value at `p = nb_equipes`
- `R` – real (default: `2`), curve parameter
- `base` – 2
- `verbose` - bool

EXEMPLES:

```
sage: from slabbe.ranking_scale import curve
sage: curve(20, 100)
-99*(p*(log(40) + 1) - p*log(p) - 20*log(40) + 20*log(20) -
20)/(19*log(40) - 20*log(20) + 19) + 1
sage: curve(64, 100)
-33*(p*(7*log(2) + 1) - p*log(p) - 64*log(2) - 64)/(19*log(2) + 21) + 1
```

```
sage: curve(64, 100)(p=64)
1
sage: curve(64, 100)(p=1)
100
sage: curve(64, 100)(p=2)
66*(26*log(2) + 31)/(19*log(2) + 21) + 1
sage: n(curve(64, 100)(p=2))     # abs tol 1e-10
95.6871477097753
```

```
sage: curve(64, 100, verbose=True)
fn = -(p*(7*log(2) + 1) - p*log(p) - 64*log(2) - 64)/log(2)
aire = 147.889787576005
fn normalise = -33*(p*(7*log(2) + 1) - p*log(p) - 64*log(2) - 64)/(19*log(2) + 21) + 1
-33*(p*(7*log(2) + 1) - p*log(p) - 64*log(2) - 64)/(19*log(2) + 21) + 1
```

The base argument seems to be useless (why?):

```
sage: curve(100,100,base=3)
-99*(p*(log(200) + 1) - p*log(p) - 100*log(200) + 200*log(10) -
100)/(99*log(200) - 200*log(10) + 99) + 1
sage: curve(100,100,base=2)
-99*(p*(log(200) + 1) - p*log(p) - 100*log(200) + 200*log(10) -
100)/(99*log(200) - 200*log(10) + 99) + 1
```

slabbe.ranking_scale.**discrete_curve**(*nb_equipes*, *max_points=100*, *K=1*, *R=2*, *base=2*, *verbose=False*)

INPUT:

- `nb_equipes` – integer
- `max_points` – integer
- `K` – the value at `p = nb_equipes`
- `R` – real (default: `2`), curve parameter

- `base` – 2
- `verbose` - bool

EXEMPLES:

```
sage: from slabbe.ranking_scale import discrete_curve
sage: A = discrete_curve(64, 100,verbose=True)
First difference sequence is
[1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1,
2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 3, 3, 2, 3, 3, 3, 4, 4, 4]
```

```
sage: A = discrete_curve(64, 100)
sage: B = discrete_curve(32, 50)
sage: C = discrete_curve(16, 25)
```

```
sage: A
[100, 96, 92, 88, 85, 82, 79, 77, 74, 71, 69, 67, 64, 62, 60, 58,
56, 54, 52, 50, 49, 47, 45, 44, 42, 41, 39, 38, 36, 35, 33, 32, 31,
29, 28, 27, 26, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12,
11, 10, 9, 8, 8, 7, 6, 5, 5, 4, 3, 2, 2, 1]
sage: B
[50, 46, 43, 40, 37, 35, 33, 31, 29, 27, 25, 23, 21, 20, 18, 17,
16, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 2, 1]
sage: C
[25, 22, 19, 17, 15, 13, 11, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
sage: A = discrete_curve(64+2, 100) #Movember, Bye Bye, Cdf, MA
sage: B = discrete_curve(32+2, 70)  # la flotte
sage: C = discrete_curve(16+2, 40) # october fest, funenuf, la viree
```

`slabbe.ranking_scale.`**`discrete_curve_2`**(*nb_equipes*, *max_points=100*)

`slabbe.ranking_scale.`**`table_to_csv`**(*self*, *filename*, *dialect='excel'*)

5.4 Fruit

Some fruits

This file is an example of Sage conventions for syntax and documentation. It also serves as an example to explain class inheritance: methods defined for fruits are automatically defined for bananas and strawberries.

AUTHORS:

- Sébastien Labbé, 2010-2013

EXAMPLES:

```
sage: from slabbe import Fruit
sage: f = Fruit(5)
sage: f
A fruit of 5 kilos.
sage: f.weight()
5
sage: f.is_a_fruit()
True
```

Because of class inheritance which says that a banana is a fruit and a strawberry is a fruit, the methods written for fruits are defined for bananas and strawberries automatically:

```
sage: from slabbe import Strawberry
sage: s = Strawberry(32)
sage: s
A strawberry of 32 kilos.
sage: s.weight()
32
sage: s.is_a_fruit()
True
```

```
sage: from slabbe import Banana
sage: b = Banana(13)
sage: b
A banana of 13 kilos.
sage: b.weight()
13
sage: b.is_a_fruit()
True
```

```
sage: s = Strawberry(32)
sage: t = Strawberry(7)
sage: s
A strawberry of 32 kilos.
sage: t
A strawberry of 7 kilos.
sage: s + t
A strawberry of 1073 kilos.
```

class `slabbe.fruit.`**`Banana`**(*weight=1*)

Bases: *`slabbe.fruit.Fruit`*

Creates a banana.

INPUT:

- `weight` - number, in kilos

OUTPUT:

Banana

EXAMPLES:

```
sage: from slabbe import Banana
sage: b = Banana(9)
sage: b
A banana of 9 kilos.
```

TESTS:

Testing that pickle works:

```
sage: loads(dumps(b))
A banana of 9 kilos.
sage: b == loads(dumps(b))
True
```

Running the test suite:

```
sage: TestSuite(b).run()
```

class `slabbe.fruit.`**`Fruit`**(*weight=1*)

Bases: `sage.structure.sage_object.SageObject`

Creates a fruit.

INPUT:

- `weight` - number, in kilos

OUTPUT:

Fruit

EXAMPLES:

```
sage: from slabbe import Fruit
sage: f = Fruit(5)
sage: f
A fruit of 5 kilos.
```

is_a_fruit()

Returns True if it is a fruit.

OUTPUT:

Boolean

EXAMPLES:

```
sage: from slabbe import Fruit
sage: f = Fruit(3)
sage: f.is_a_fruit()
True
```

weight()

Return the weight.

OUTPUT:

Number

EXAMPLES:

```
sage: from slabbe import Fruit
sage: f = Fruit(3)
sage: f.weight()
3
```

class slabbe.fruit.**Strawberry**(*weight=1*)

Bases: *slabbe.fruit.Fruit*

Creates a strawberry.

INPUT:

- `weight` - number, in kilos

OUTPUT:

Strawberry

EXAMPLES:

```
sage: from slabbe import Strawberry
sage: s = Strawberry(34)
sage: s
A strawberry of 34 kilos.
```

TESTS:

Testing that pickle works:

```
sage: loads(dumps(s))
A strawberry of 34 kilos.
sage: s == loads(dumps(s))
True
```

Running the test suite:

```
sage: TestSuite(s).run()
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[LR2014] Labbé, Sébastien, and Christophe Reutenauer. A d-dimensional Extension of Christoffel Words. arXiv:1404.4021 (April 15, 2014).

[WVL1984] Wijshoff, H. A. G, et J. Van Leeuwen. Arbitrary versus periodic storage schemes and tessellations of the plane using one type of polyomino. INFORM. AND CONTROL 62 (1984): 1-25.

[BN1991] Beauquier, D., and M. Nivat. On translating one polyomino to tile the plane. Discrete & Computational Geometry 6 (1991): 575-592. doi:10.1007/BF02574705

[BFP2009] S. Brlek, J.-M Fédou, X. Provençal, On the Tiling by Translation Problem, Discrete Applied Mathematics 157 Issue 3 (2009) 464-475. doi:10.1016/j.dam.2008.05.026

[BBL2012] A. Blondin Massé, S. Brlek, S. Labbé, A parallelogram tile fills the plane by translation in at most two distinct ways, Discrete Applied Mathematics 160 (2012) 1011-1018. doi:10.1016/j.dam.2011.12.023

[BBGL2011] A. Blondin Massé, S. Brlek, A. Garon, S. Labbé, Two infinite families of polyominoes that tile the plane by translation in two distinct ways, Theoret. Comput. Sci. 412 (2011) 4778-4786. doi:10.1016/j.tcs.2010.12.034

[BGL2012] A. Blondin Massé, A. Garon, S. Labbé, Combinatorial properties of double square tiles, Theoretical Computer Science, Available online 2 November 2012. doi:10.1016/j.tcs.2012.10.040

[K65] William Kolakoski, proposal 5304, American Mathematical Monthly 72 (1965), 674; for a partial solution, see "Self Generating Runs," by Necdet Üçoluk, Amer. Math. Mon. 73 (1966), 681-2.

[O39] R. Oldenburger, Exponent trajectories in dynamical systems, Trans. Amer. Math. Soc. 46 (1939), 453–466.

[BL2014] V. Berthé, S. Labbé, Factor Complexity of S-adic sequences generated by the Arnoux-Rauzy-Poincaré Algorithm. arXiv:1404.4189 (April, 2014).

[T1980] R., Tijdeman. The chairman assignment problem. Discrete Mathematics 32, no 3 (1980): 323-30. doi:10.1016/0012-365X(80)90269-1.

[Ber2001] Valérie Berthé. Autour du système de numération d'Ostrowski. Bull. Belg. Math. Soc. Simon Stevin, 8(2):209–239, 2001. Journées Montoises d'Informatique Théorique (Marne-la-Vallée, 2000).

[Bou2015] Bourla, Avraham. « Irrational Base Counting ». arXiv:1511.02179 [math], 6 novembre 2015. http://arxiv.org/abs/1511.02179.

[POG] Geoffrey Grimmett, Probability on Graphs, http://www.statslab.cam.ac.uk/~grg/books/pgs.html

PYTHON MODULE INDEX

INDEX