Documentation for impl

Release unknown

unknown

May 04, 2021

CONTENTS

impl.**growing_letters**(*self*)
Return the list of growing letters.

See `is_growing()` for more information.

EXAMPLES:

```
sage: WordMorphism('0->01,1->10').growing_letters()
['0', '1']
sage: WordMorphism('0->01,1->1').growing_letters()
['0']
sage: WordMorphism('0->01,1->0,2->1',codomain=Words('012')).growing_letters()
['0', '1', '2']
```

TESTS:

Make sure that trac ticket #31454 is fixed:

```
sage: WordMorphism('a->a').growing_letters()
[]
```

impl.**immortal_letters**(*self*)
Return the set of immortal letters.

A letter a is *immortal* for the morphism s if the length of the iterates of $|s^n(a)|$ is larger than zero as n goes to infinity.

Requires this morphism to be an endomorphism.

EXAMPLES:

```
sage: WordMorphism('a->abcd,b->cd,c->dd,d->').immortal_letters()
{'a'}
```

impl.**infinite_repetitions**(*self*, *w=None*)
Return the set of primitive infinite repetitions (up to conjugacy) from the language $\{m^n(w)|n \geq 0\}$, where m is this morphism and w is a word inputted as a parameter.

Requires this morphism to be an endomorphism.

A non-empty word v is an infinite repetition (also known as an infinite periodic factor) of a language if for each positive integer k the word v^k is a factor of some word from the language.

If v is an infinite repetition, then all its powers are also infinite repetitions, therefore this method returns only the primitive ones. It turns out that a language created by iterating a morphism has a finite number of primitive infinite repetitions.

Similarly, if v is an infinite repetition, then all its conjugates are also infinite repetitions, therefore this method returns only the lexicographically minimal one from each conjugacy class.

INPUT:

- `w` – finite iterable representing a word used to start the language, default is `self.domain().alphabet()`

EXAMPLES:

```
sage: m = WordMorphism('a->aba,b->aba,c->cd,d->e,e->d')
sage: inf_reps = m.infinite_repetitions('ac')
sage: sorted(inf_reps)
[word: aab, word: de]
```

Incomplete check that these words are indeed infinite repetitions:

```
sage: SL = m._language_naive(10, Word('ac'))
sage: all(x in SL for x in inf_reps)
True
sage: all(x^2 in SL for x in inf_reps)
True
sage: all(x^3 in SL for x in inf_reps)
True
```

Larger example:

```
sage: m = WordMorphism('a->1b5,b->fcg,c->dae,d->432,e->678,f->f,g->g,1->2,2->3,3->
↪4,4->1,5->6,6->7,7->8,8->5')
sage: sorted(m.infinite_repetitions('a'))
[word: 1432f2143f3214f4321f, word: 5678g8567g7856g6785g]
```

impl.**infinite_repetitions_bounded**(*self*, *w=None*)

Return the set of primitive infinite repetitions (up to conjugacy), which contain no growing letters, from the language $\{m^n(w)|n \geq 0\}$, where m is this morphism and w is a word inputted as a parameter.

Requires this morphism to be an endomorphism.

See *infinite_repetitions()* and *is_growing()*.

INPUT:

- `w` – finite iterable representing a word used to start the language, default is `self.domain().alphabet()`

ALGORITHM:

The algorithm used is described in detail in [KS2015].

EXAMPLES:

```
sage: m = WordMorphism('a->aba,b->aba,c->cd,d->e,e->d')
sage: sorted(m.infinite_repetitions_bounded())
[word: de]

sage: m = WordMorphism('c->d,d->c,e->fc,f->ed')
sage: sorted(m.infinite_repetitions_bounded())
[word: c, word: d]
```

TESTS:

```
sage: m = WordMorphism('a->Cab,b->1c1,c->E2bd5,d->BbaA,5->6,6->7,7->8,8->9,9->5,1-
↪>2,2->1,A->B,B->C,C->D,D->E,E->')
sage: sorted(m.infinite_repetitions_bounded())
[word: 1, word: 1519181716, word: 2, word: 2529282726]
```

impl.**infinite_repetitions_growing**(*self*, *w=None*)

Return the set of primitive infinite repetitions (up to conjugacy), which contain at least one growing letter, from the language $\{m^n(w)|n \geq 0\}$, where m is this morphism and w is a word inputted as a parameter.

Requires this morphism to be an endomorphism.

See *infinite_repetitions()* and *is_growing()*.

INPUT:

- `w` – finite iterable representing a word used to start the language, default is `self.domain().alphabet()`

ALGORITHM:

The algorithm used is described in detail in [KS2015].

EXAMPLES:

```
sage: m = WordMorphism('a->aba,b->aba,c->cd,d->e,e->d')
sage: sorted(m.infinite_repetitions_growing())
[word: aab]

sage: m = WordMorphism('a->bcb,b->ada,c->d,d->c')
sage: sorted(m.infinite_repetitions_growing())
[word: ad, word: bc]

sage: m = WordMorphism('b->c,c->bcb')
sage: sorted(m.infinite_repetitions_growing())
[word: bc]

sage: m = WordMorphism('a->abc,b->dab,c->abc,d->dab')
sage: sorted(m.infinite_repetitions_growing())
[word: ababcd]
```

impl.**is_growing**(*self*, *letter=None*)

Return `True` if `letter` is a growing letter.

A letter a is *growing* for the morphism s if the length of the iterates of $|s^n(a)|$ tend to infinity as n goes to infinity.

INPUT:

- `letter` – `None` or a letter in the domain of `self`

Note: If letter is `None`, this returns `True` if `self` is everywhere growing, i.e., all letters are growing letters (see [CassNic10]), and that `self` **must** be an endomorphism.

EXAMPLES:

```
sage: WordMorphism('0->01,1->1').is_growing('0')
True
sage: WordMorphism('0->01,1->1').is_growing('1')
False
sage: WordMorphism('0->01,1->10').is_growing()
True
sage: WordMorphism('0->1,1->2,2->01').is_growing()
True
sage: WordMorphism('0->01,1->1').is_growing()
False
```

The domain needs to be equal to the codomain:

```
sage: WordMorphism('0->01,1->0,2->1',codomain=Words('012')).is_growing()
True
```

Test of erasing morphisms:

```
sage: WordMorphism('0->01,1->').is_growing('0')
False
sage: m = WordMorphism('a->bc,b->bcc,c->',codomain=Words('abc'))
sage: m.is_growing('a')
False
sage: m.is_growing('b')
False
sage: m.is_growing('c')
False
```

TESTS:

Make sure that trac ticket #31454 is fixed:

```
sage: WordMorphism('a->a').is_growing('a')
False
```

REFERENCES:

impl.**is_injective**(*self*)

Return whether this morphism is injective.

ALGORITHM:

Uses a version of Wikipedia article Sardinas–Patterson_algorithm. Time complexity is on average quadratic with regards to the size of the morphism.

EXAMPLES:

```
sage: WordMorphism('a->0,b->10,c->110,d->111').is_injective()
True
sage: WordMorphism('a->00,b->01,c->012,d->20001').is_injective()
False
```

impl.**is_pushy**(*self*, *w=None*)

Return whether the language $\{m^n(w)|n \geq 0\}$ is pushy, where m is this morphism and w is a word inputted as a parameter.

Requires this morphism to be an endomorphism.

A language created by iterating a morphism is pushy if its words contain an infinite number of factors containing no growing letters. It turns out that this is equivalent to having at least one infinite repetition containing no growing letters.

See *infinite_repetitions()* and *is_growing()*.

INPUT:

- `w` – finite iterable representing a word used to start the language, default is `self.domain().alphabet()`

EXAMPLES:

```
sage: WordMorphism('a->abca,b->bc,c->').is_pushy()
False
sage: WordMorphism('a->abc,b->,c->bcb').is_pushy()
True
```

impl.**is_repetitive**(*self*, *w=None*)

Return whether the language $\{m^n(w)|n \geq 0\}$ is repetitive, where m is this morphism and w is a word inputted as a parameter.

Requires this morphism to be an endomorphism.

A language is repetitive if for each positive integer k there exists a word u such that u^k is a factor of some word of the language.

It turns that for languages created by iterating a morphism this is equivalent to having at least one infinite repetition (this property is also known as strong repetitiveness).

See *infinite_repetitions()*.

INPUT:

- `w` – finite iterable representing a word used to start the language, default is `self.domain().alphabet()`

EXAMPLES:

This method can be used to check whether a purely morphic word is NOT k-power free for all positive integers k. For example, the language containing just the Thue-Morse word and its prefixes is not repetitive, since the Thue-Morse word is cube-free:

```
sage: WordMorphism('a->ab,b->ba').is_repetitive('a')
False
```

Similarly, the Hanoi word is square-free:

```
sage: WordMorphism('a->aC,A->ac,b->cB,B->cb,c->bA,C->ba').is_repetitive('a')
False
```

However, this method solves a more general problem, as it can be called on any morphism m and with any word w:

```
sage: WordMorphism('a->c,b->cda,c->a,d->abc').is_repetitive('bd')
True
```

`impl.`**`is_unboundedly_repetitive`**(*self*, *w=None*)

Return whether the language $\{m^n(w)|n \geq 0\}$ is unboundedly repetitive, where m is this morphism and w is a word inputted as a parameter.

Requires this morphism to be an endomorphism.

A language created by iterating a morphism is unboundedly repetitive if it has at least one infinite repetition containing at least one growing letter.

See *infinite_repetitions()* and *is_growing()*.

INPUT:

- `w` – finite iterable representing a word used to start the language, default is `self.domain().alphabet()`

EXAMPLES:

```
sage: WordMorphism('a->abca,b->bc,c->').is_unboundedly_repetitive()
True
sage: WordMorphism('a->abc,b->,c->bcb').is_unboundedly_repetitive()
False
```

`impl.`**`language_naive`**(*self*, *n*, *u*)

Return all words of length less than `n` by naive substitution.

The language of the substitution is the DOL language which consist of factors of $s^n(u)$.

This method assumes this substitution is non-erasing.

INPUT:

- `n` – non-negative integer - length of the words in the language
- `u` – a word used as a seed

OUTPUT: a Python set

TESTS:

```
sage: s = WordMorphism({0: [0,1], 1:[0]})
sage: W = s.domain()
sage: sorted(s._language_naive(3, W([0])))
[word: 0, word: 00, word: 01, word: 1, word: 10]
sage: sorted(s._language_naive(3, W([1])))
[word: 0, word: 00, word: 01, word: 1, word: 10]

sage: s._language_naive(3, W())
set()
sage: W([1, 1]) in s._language_naive(3, W([1, 1]))
True
```

impl.**minimal_conjugate**(*self*)

Return the lexicographically minimal conjugate of this word (see Wikipedia article Lexicographically_minimal_string_rotation).

EXAMPLES:

```
sage: Word('213').minimal_conjugate()
word: 132
sage: Word('11').minimal_conjugate()
word: 11
sage: Word('12112').minimal_conjugate()
word: 11212
sage: Word('211').minimal_conjugate()
word: 112
sage: Word('211211211').minimal_conjugate()
word: 112112112
```

TESTS:

```
sage: Word().minimal_conjugate()
word:
```

impl.**reach**(*self*, *w*)

Return the set of letters which occur in words of $\{m^n(w)|n \geq 0\}$, where m is this morphism and w is a word (finite iterable is enough) inputted as a parameter.

Requires this morphism to be an endomorphism.

EXAMPLES:

```
sage: sorted(WordMorphism('a->ac,b->ce,c->bd,d->d,e->').reach('c'))
['b', 'c', 'd', 'e']
```

impl.**simplify**(*self*, *Z=None*)

If this morphism is simplifiable, return morphisms h and k such that this morphism is simplifiable with respect to h and k, otherwise raise `ValueError`.

This method is quite fast if this morphism is non-injective, but very slow if it is injective.

Let $f : X^* \to Y^*$ be a morphism. Then f is simplifiable with respect to morphisms $h : X^* \to Z^*$ and $k : Z^* \to Y^*$, if $f = k \circ h$ and $|Z| < |X|$. If also $Y \subseteq X$, then the morphism $g : Z^* \to Z^* = h \circ k$ is a simplification of f (with respect to h and k).

Loosely speaking a morphism is simplifiable if it contains "more letters than is needed". Non-injectivity implies simplifiability. Simplification preserves some properties of the original morphism (e.g. repetitiveness).

For more information see Section 3 in [KO2000].

INPUT:

- `Z` – iterable, whose elements are used as an alphabet for the simplification, default is `self.domain().alphabet()`

EXAMPLES:

Example of a simplifiable (non-injective) morphism:

```
sage: f = WordMorphism('a->aca,b->badc,c->acab,d->adc')
sage: h, k = f.simplify('xyz'); h, k
(WordMorphism: a->x, b->zy, c->xz, d->y, WordMorphism: x->aca, y->adc, z->b)
sage: k * h == f
True
sage: g = h * k; g
WordMorphism: x->xxzx, y->xyxz, z->zy
```

Example of a simplifiable (injective) morphism:

```
sage: f = WordMorphism('a->abcc,b->abcd,c->abdc,d->abdd')
sage: h, k = f.simplify('xyz'); h, k
(WordMorphism: a->xyy, b->xyz, c->xzy, d->xzz, WordMorphism: x->ab, y->c, z->d)
sage: k * h == f
True
sage: g = h * k; g
WordMorphism: x->xyyxyz, y->xzy, z->xzz
```

Example of a non-simplifiable morphism:

```
sage: WordMorphism('a->aa').simplify()
Traceback (most recent call last):
...
ValueError: self (a->aa) is not simplifiable
```

Example of an erasing morphism:

```
sage: f = WordMorphism('a->abc,b->cc,c->')
sage: h, k = f.simplify(); h, k
(WordMorphism: a->a, b->b, c->, WordMorphism: a->abc, b->cc)
sage: k * h == f
True
sage: g = h * k; g
WordMorphism: a->ab, b->
```

Example of a morphism, that is not an endomorphism:

```
sage: f = WordMorphism('a->xx,b->xy,c->yx,d->yy')
sage: h, k = f.simplify(NN); h, k
(WordMorphism: a->00, b->01, c->10, d->11, WordMorphism: 0->x, 1->y)
```

(continues on next page)

(continued from previous page)

```
sage: k * h == f
True
sage: len(k.domain().alphabet()) < len(f.domain().alphabet())
True
```

impl.**simplify_injective**(*self*)

Return a quadruplet (g, h, k, i), where g is an injective simplification of this morphism with respect to h, k and i.

Requires this morphism to be an endomorphism.

This methods basically calls *simplify()* until the returned simplification is injective. If this morphism is already injective, a quadruplet (g, h, k, i) is still returned, where g is this morphism, h and k are the identity morphisms and i is 0.

Let $f : X^* \to Y^*$ be a morphism and $Y \subseteq X$. Then $g : Z^* \to Z^*$ is an injective simplification of f with respect to morphisms $h : X^* \to Z^*$ and $k : Z^* \to Y^*$ and a positive integer i, if g is injective, $|Z| < |X|$, $g^i = h \circ k$ and $f^i = k \circ h$.

For more information see Section 4 in [KO2000].

EXAMPLES:

```
sage: f = WordMorphism('a->abc,b->a,c->bc')
sage: g, h, k, i = f.simplify_injective(); g, h, k, i
(WordMorphism: a->aa, WordMorphism: a->aa, b->a, c->a, WordMorphism: a->abc, 2)
sage: g.is_injective()
True
sage: g ** i == h * k
True
sage: f ** i == k * h
True
```

BIBLIOGRAPHY

[CassNic10] Cassaigne J., Nicolas F. Factor complexity. Combinatorics, automata and number theory, 163–247, Encyclopedia Math. Appl., 135, Cambridge Univ. Press, Cambridge, 2010.

INDEX