Fully commutative elements of Coxeter groups

An element \( w \) in a Coxeter system \((W, S)\) is fully commutative (FC) if every two reduced words of \( w \) can be related by a sequence of only commutation relations, i.e., relations of the form \( st = ts \) where \( s, t \) are commuting generators in \( S \). See [Ste1996].

Authors:

- Chase Meadors, Tianyuan Xu (2020): Initial version

Acknowledgements

A draft of this code was written during an REU project at University of Colorado Boulder. We thank Rachel Castro, Joel Courtney, Thomas Magnuson and Natalie Schoenhals for their contribution to the project and the code.

```python
class sage.combinat.fully_commutative_elements.FullyCommutativeElement
    Bases: sage.structure.list_clone.NormalizedClonableList

A fully commutative (FC) element in a Coxeter system.

An element \( w \) in a Coxeter system \((W, S)\) is fully commutative (FC) if every two reduced word of \( w \) can be related by a sequence of only commutation relations, i.e., relations of the form \( st = ts \) where \( s, t \) are commuting generators in \( S \).

Every FC element has a canonical reduced word called its Cartier–Foata form. See [Gre2006]. We will normalize each FC element to this form.

```cartier_foata_form()`

Return the Cartier–Foata normal form of self.

`normalize()` is an alias of this method, and is called automatically when an element is created.

EXAMPLES:

The following reduced words express the same FC elements in \( B_5 \):

```python
sage: FC = FullyCommutativeElements(['B', 5])
sage: FC([1, 4, 3, 5, 2, 4, 3])  # indirect doctest
[1, 4, 3, 5, 2, 4, 3]
sage: FC([4, 1, 3, 5, 2, 4, 3])  # indirect doctest
[1, 4, 3, 5, 2, 4, 3]
sage: FC([4, 3, 1, 5, 4, 2, 3])  # indirect doctest
[1, 4, 3, 5, 2, 4, 3]
```

Note: The Cartier–Foata form of a reduced word of an FC element \( w \) can be found recursively by repeatedly moving left descents of elements to the left and ordering the left descents from small to large. In the above example, the left descents of the element
are 4 and 1, therefore the Cartier–Foata form of the element is the concatenation of [1,4] with the Cartier–Foata form of the remaining part of the word. See [Gre2006].

See also: descents()

cHECK()

Called automatically when an element is created. Alias of is_fully_commutative()

coset_decomposition($J$, side='left')

Return the coset decomposition of self with respect to the parabolic subgroup generated by $J$.

INPUT:

- $J$ – subset of the generating set $S$ of the Coxeter system.

OUTPUT:

The tuple of elements $(w_J, w^J)$ such that $w = w_J \cdot w^J$, $w_J$ is generated by the elements in $J$, and $w^J$ has no left descent from $J$. This tuple is unique and satisfies the equation $l(w) = l(w_J) + l(w^J)$, where $l$ denotes Coxeter length, by general theory; see Proposition 2.4.4 of [BB2005].

OPTIONAL ARGUMENTS:

- side – string (default: ‘left’); if the value is set to ‘right’, then the function returns the tuple $(w'^J, w'_J)$ from the coset decomposition $w = w'^J \cdot w'_J$ of $w$ with respect to $J$.

EXAMPLES:

```
sage: FC = FullyCommutativeElements(['B', 6])
sage: w = FC([1, 6, 2, 5, 4, 6, 5])
sage: w.coset_decomposition({{1}})
([1], [6, 2, 5, 4, 6, 5])
sage: w.coset_decomposition({{1}}, side = 'right')
([1, 6, 2, 5, 4, 6, 5], [1])
sage: w.coset_decomposition({{5, 6}})
([5, 6], [1, 2, 4, 5])
sage: w.coset_decomposition({{5, 6}}, side='right')
([1, 6, 2, 5, 4], [6, 5])
```

**Note:** The factor $w_J$ of the coset decomposition $w = w_J \cdot w^J$ can be obtained by greedily "pulling left descents of $w$ that are in $J$ to the left"; see the proof of [BB2005]. This greedy algorithm works for all elements in Coxeter group, but it becomes especially simple for FC elements because descents are easier to find for FC elements.

descents($side='left'$)

Obtain the set of descents on the appropriate side of self.
A generator $s$ is called a left or right descent of an element $w$ if $l(sw)$ or $l(ws)$ is smaller than $l(w)$, respectively. If $w$ is FC, then $s$ is a left descent of $w$ if and only if $s$ appears to the left of the leftmost $s$ in the word and every generator to the left of the leftmost $s$ in the word commutes with $s$. A similar characterization exists for right descents of FC elements.

**OPTIONAL ARGUMENTS:**

- `side` — string (default: 'left'); if set to 'right', find the right descents.

**EXAMPLES:**

```python
sage: FC = FullyCommutativeElements(['B', 5])
sage: w = FC([1, 4, 3, 5, 2, 4, 3])
sage: sorted(w.descents())
[1, 4]
sage: w.descents(side='right')
{3}
sage: FC = FullyCommutativeElements(['A', 5])
sage: sorted(FC([1, 4, 3, 5, 2, 4, 3]).descents())
[1, 4]
```

**See also:** `find_descent()`

```python
find_descent(s, side='left')
```

Check if $s$ is a descent of `self` and find its position if so.

A generator $s$ is called a left or right descent of an element $w$ if $l(sw)$ or $l(ws)$ is smaller than $l(w)$, respectively. If $w$ is FC, then $s$ is a left descent of $w$ if and only if $s$ appears to the left of the leftmost $s$ in the word and every generator to the left of the leftmost $s$ in the word commutes with $s$. A similar characterization exists for right descents of FC elements.

**INPUT:**

- `s` — integer representing a generator of the Coxeter system.

**OUTPUT:**

Determine if the generator $s$ is a left descent of `self`; return the index of the leftmost occurrence of $s$ in `self` if so and return `None` if not.

**OPTIONAL ARGUMENTS:**

- `side` — string (default: 'left'); if the argument is set to 'right', the function checks if $s$ is a right descent of `self` and finds the index of the rightmost occurrence of $s$ if so.

**EXAMPLES:**

```python
sage: FC = FullyCommutativeElements(['B', 5])
sage: w = FC([1, 4, 3, 5, 2, 4, 3])
sage: w.find_descent(1)
0
sage: w.find_descent(1, side='right')
```
```sage
w.find_descent(4)
1
sage: w.find_descent(4, side='right')

sage: w.find_descent(3)
```

```python
has_descent(s, side='left')
```

Determine if `s` is a descent on the appropriate side of `self`.

**OUTPUT:** a boolean value

**OPTIONAL ARGUMENTS:**

- `side` – string (default: 'left'); if set to 'right', determine if `self` has `s` as a right descent.

**EXAMPLES:**

```sage
sage: FC = FullyCommutativeElements(['B', 5])
sage: w = FC([1, 4, 3, 5, 2, 4, 3])
sage: w.has_descent(1)
True
sage: w.has_descent(1, side='right')
False
sage: w.has_descent(4)
True
sage: w.has_descent(4, side='right')
False
```

**See also:** `find_descent()`

```python
heap(**kargs)
```

Create the heap poset of `self`.

The heap of an FC element `w` is a labeled poset that can be defined from any reduced word of `w`. Different reduced words yield isomorphic labeled posets, so the heap is well defined.

Heaps are very useful for visualizing and studying FC elements; see, for example, [Ste1996] and [GX2020].

**INPUT:**

- `self` – list, a reduced word `w = s_0 \ldots s_{k-1}` of an FC element.

**OUTPUT:** A labeled poset where the underlying set is \{0, 1, \ldots, k - 1\} and where each element `i` carries `s_i` as its label. The partial order `<` on the poset is defined by declaring `i` `<` `j` if `i` `<` `j` and `m(s_i, s_j) \neq 2`.

**OPTIONAL ARGUMENTS:**

- `one_index` – boolean (default: False). Setting the value to True will change the underlying set of the poset to \{1, 2, \ldots, n\}. 
- **display_labeling** — boolean (default: False). Setting the value to True will display the label $s_i$ for each element $i$ of the poset.

**EXAMPLES:**

```python
sage: FC = FullyCommutativeElements(['A', 5])
sage: FC([1, 4, 3, 5, 2, 4]).heap().cover_relations()
[[1, 2], [1, 3], [2, 5], [2, 4], [3, 5], [0, 4]]
sage: FC([1, 4, 3, 5, 4, 2]).heap(one_index=True).cover_relations()
[[2, 3], [2, 4], [3, 6], [3, 5], [4, 6], [1, 5]]
```

**is_fully_commutative()**

Check if `self` is the reduced word of an FC element.

To check if `self` is FC, we use the well-known characterization that an element $w$ in a Coxeter system $(W, S)$ is FC if and only if for every pair of generators $s, t \in S$ for which $m(s, t) > 2$, no reduced word of $w$ contains the ‘braid’ word $sts\ldots$ of length $m(s, t)$ as a contiguous subword. See [Ste1996].

`check()` is an alias of this method, and is called automatically when an element is created.

**n_value()**

Calculate the n-value of `self`.

The n-value of a fully commutative element is the width (length of any longest antichain) of its heap. The n-value is important as it coincides with Lusztig’s a-value for FC elements in all Weyl and affine Weyl groups as well as so-called star-reducible groups; see [GX2020].

**EXAMPLES:**

```python
sage: FC = FullyCommutativeElements(['A', 5])
sage: FC([1, 3]).n_value()
2
sage: FC([1, 2, 3]).n_value()
1
sage: FC([1, 3, 2]).n_value()
2
sage: FC([1, 3, 2, 5]).n_value()
3
```

**normalize()**

Called automatically when an element is created. Alias of `cartier_foata_form()`

**plot_heap()**

Display the Hasse diagram of the heap of `self`.

The Hasse diagram is rendered in the lattice $S \times \mathbb{N}$, with every element $i$ in the poset drawn as a point labelled by its label $s_i$. Every point is placed in the column for its label at a certain level. The levels start at 0 and the level $k$ of an element $i$ is the maximal number $k$ such that the heap contains a chain $i_0 \prec i_1 \prec \ldots \prec i_k$ where $i_k = i$. See [Ste1996] and [GX2020].
star_operation(\(J, \text{direction}, \text{side}='\text{left}'\))

Apply a star operation on \(\text{self}\) relative to two noncommuting generators.

Star operations were first defined on elements of Coxeter groups by Kazhdan and Lusztig in [KL1979] with respect to pair of generators \(s, t\) such that \(m(s, t) = 3\). Later, Lusztig generalized the definition in [Lus1985], via coset decompositions, to allow star operations with respect to any pair of generators \(s, t\) such that \(m(s, t) \geq 3\). Given such a pair, we can potentially perform four types of star operations corresponding to all combinations of a 'direction' and a 'side': upper left, lower left, upper right and lower right; see [Gre2006].

Let \(w\) be an element in \(W\) and let \(J\) be any pair \(\{s, t\}\) of noncommuting generators in \(S\). Consider the coset decomposition \(w = w_J \cdot J w\) of \(w\) relative to \(J\). Then an upper left star operation is defined on \(w\) if and only if \(1 \leq l(w_J) \leq m(s, t) - 2\); when this is the case, the operation returns \(x \cdot w_J \cdot w_J^J\) where \(x\) is the letter \(J\) different from the leftmost letter of \(w_J\). A lower left star operation is defined on \(w\) if and only if...
2 \leq l(w_J) \leq m(s, t) - 1; when this is the case, the operation removes the leftmost letter of $w_J$ from $w$. Similar facts hold for right star operations. See [Gre2006].

The facts of the previous paragraph hold in general, even if $w$ is not FC. Note that if $f$ is a star operation of any kind, then for every element $w \in W$, the elements $w$ and $f(w)$ are either both FC or both not FC.

**INPUT:**

- $J$ – a set of two integers representing two noncommuting generators of the Coxeter system.
- `direction` – string, ‘upper’ or ‘lower’; the function performs an upper or lower star operation according to `direction`.
- `side` – string (default: ‘left’); if this is set to ‘right’, the function performs a right star operation.

**OUTPUT:**

The Cartier–Foata form of the result of the star operation if the operation is defined on self, None otherwise.

**EXAMPLES:**

We will compute all star operations on the following FC element in type $B_6$ relative to $J = \{5, 6\}$:

```sage
sage: FC = FullyCommutativeElements(['B', 6])
sage: w = FC([1, 6, 2, 5, 4, 6, 5])
```

Whether and how a left star operations can be applied depend on the coset decomposition $w = w_J \cdot w^J$:

```sage
sage: w.coset_decomposition({5, 6})
([6, 5, 6], [1, 2, 4, 5])
```

Only the lower star operation is defined on the left on $w$:

```sage
sage: w.star_operation({5, 6}, 'upper')
sage: w.star_operation({5, 6}, 'lower')
[1, 5, 2, 4, 6, 5]
```

Whether and how a right star operations can be applied depend on the coset decomposition $w = w^J \cdot w_J$:

```sage
sage: w.coset_decomposition({5, 6}, side='right')
([1, 6, 2, 5, 4], [6, 5])
```

Both types of right star operations on defined for this example:
sage: w.star_operation({5, 6}, 'upper', side='right')
[1, 6, 2, 5, 4, 6, 5, 6]
sage: w.star_operation({5, 6}, 'lower', side='right')
[1, 6, 2, 5, 4, 6]

still_reduced_fc_after_prepending(s)

Determine if self prepended with s is still a reduced word of an FC element in the Coxeter system.

INPUT:

- s – integer representing a generator of the Coxeter system.
- self – a reduced word of an FC element

EXAMPLES:

Consider the FC element $w = 12$ in the group $B_3$:

```sage
sage: FCB3 = FullyCommutativeElements(['B', 3])
sage: FCB3.coxeter_matrix()
[1 3 2]
[3 1 4]
[2 4 1]
sage: w = FCB3([1, 2])
```

When $s = 1$, $sw$ is 112, which is not reduced:

```sage
sage: w stil_reduced_fc_after_prepending(1)
False
```

When $s = 2$, $sw$ is 212, which is reduced but not FC:

```sage
sage: w still_reduced_fc_after_prepending(2)
False
```

When $s = 31$, is 312, which is reduced and FC:

```sage
sage: w still_reduced_fc_after_prepending(3)
True
```

More examples:

```sage
sage: u = FCB3([3, 1, 2])
sage: u still_reduced_fc_after_prepending(1)
False
sage: u still_reduced_fc_after_prepending(2)
True
sage: u still_reduced_fc_after_prepending(3)
False

sage: FCA5 = FullyCommutativeElements(['A', 5])
sage: w = FCA5([2, 4, 1, 3, 2, 5])
sage: w still_reduced_fc_after_prepending(5)
False
```
Note: If \( w \) is a reduced word of an element, then the concatenation \( sw \) is still a reduced word if and only if \( s \) is not a left descent of \( w \) by general Coxeter group theory. So now assume \( w \) is a reduced word of an FC element and \( s \) is not a left descent \( w \). In this case, Lemma 4.1 of [Ste1996] implies that \( sw \) is not a reduced word of an FC element if and only if some letter in \( w \) does not commute with \( s \) and the following conditions hold simultaneously for the leftmost such letter \( t \):

1. \( t \) is left descent of the word \( u_1 \) obtained by removing all letters to the left of the aforementioned \( t \) from \( w \); (this condition is automatically true by definition of \( u_1 \))

2. \( s \) is left descent of the word \( u_2 \) obtained by removing the leftmost \( t \) from \( u_1 \);

3. \( t \) is left descent of the word \( u_3 \) obtained by removing the leftmost \( s \) from \( u_2 \); ... (m-1) the appropriate element in \( \{s, t\} \) is a left descent of the word \( u_{m-1} \) obtained by removing the leftmost letter required to be a descent in Condition (m-2) from \( u_{m-2} \).

In the last example above, we have \( s = 5 \), \( t = 4 \), Condition (1) holds, but Condition (2) fails, therefore \( 5w \) is still a reduced word of an FC element.

Note that the conditions (1)–(m-1) are equivalent to the condition that the parabolic factor \( u_J \) from the coset decomposition \( u_1 = u_J \cdot u_J^J \) of \( u_1 \) with respect to \( J := \{s, t\} \) is the element \( tst... \) of length \( m(s, t) - 1 \).

REFERENCES:

See Lemma 4.1 of [Ste1996].

```python
class sage.combinat.fully_commutative_elements.FullyCommutativeElements(data)
```

Bases: `sage.structure.parent.Parent`

Class for the set of fully commutative (FC) elements of a Coxeter system.

Coxeter systems with finitely many FC elements, or FC-finite Coxeter systems, are classified by Stembridge in [Ste1996]. They fall into seven families, namely the groups of types \( A_n, B_n, D_n, E_n, F_n, H_n \) and \( I_2(m) \).

INPUT:

- data – CoxeterMatrix, CartanType, or the usual datum that can is taken in the constructors for these classes (see `sage.combinat.root_system.coxeter_group.CoxeterGroup()`)

OUTPUT:

The class of fully commutative elements in the Coxeter group constructed from data. This will belong to the category of enumerated sets. If the Coxeter data corresponds to a Cartan type, the category is further refined to either finite enumerated sets or infinite enumerated sets depending on i whether the Coxeter group is FC-finite; the refinement is not carried out if data is a Coxeter matrix not corresponding to a Cartan type.
Todo: It would be ideal to implement the aforementioned refinement to finite and infinite enumerated sets for all possible data, regardless of whether it corresponds to a Cartan type. Doing so requires determining if an arbitrary Coxeter matrix corresponds to a Cartan type. It may be best to address this issue in sage.combinat.root_system. On the other hand, the refinement in the general case may be unnecessary in light of the fact that Stembridge’s classification of FC-finite groups contains a very small number of easily-recognizable families.

EXAMPLES:

Create the enumerate set of fully commutative elements in $B_3$:

```sage
sage: FC = FullyCommutativeElements(['B', 3]); FC
Fully commutative elements in Coxeter system with Cartan type ['B', 3]
sage: FC.coxeter_matrix()
[1 3 2]
[3 1 4]
[2 4 1]
```

Construct elements:

```sage
sage: FC([])
[]
sage: FC([1,2])
[1, 2]
sage: FC([2,3,2])
[2, 3, 2]
sage: FC([3,2,3])
[3, 2, 3]
```

Elements are normalized to Cartier–Foata normal form upon construction:

```sage
sage: FC([3,1])
[1, 3]
sage: FC([2,3,1])
[2, 1, 3]
sage: FC([1,3]) == FC([3,1])
True
```

Attempting to create an element from an input that is not the reduced word of a fully commutative element throws a `ValueError`:

```sage
sage: FC([1,2,1])
Traceback (most recent call last):
 ... 
ValueError: the input is not a reduced word of a fully commutative element
```

Enumerate the FC elements in $A_3$:

```sage
sage: FCA3 = FullyCommutativeElements(['A', 3])
sage: FCA3.category()
```

Category of finite enumerated sets

sage: FCA3.list()
[[], [1], [2], [3], [2, 1], [1, 3], [1, 2], [3, 2], [2, 3], [3, 2, 1], [2, 1, 3], [1, 3, 2], [1, 2, 3], [2, 1, 3, 2]]

Count the FC elements in $B_8$:

sage: FCB8 = FullyCommutativeElements(['B', 8])
sage: len(FCB8)  # long time (7 seconds)
14299

Iterate through the FC elements of length up to 2 in the non-FC-finite group affine $A_2$:

sage: FCAffineA2 = FullyCommutativeElements(['A', 2, 1])
sage: FCAffineA2.category()
Category of infinite enumerated sets
sage: list(FCAffineA2.iterate_to_length(2))
[[], [0], [1], [2], [1, 0], [2, 0], [0, 1], [2, 1], [0, 2], [1, 2]]

Element

alias of FullyCommutativeElement

coxeter_matrix()

Obtain the Coxeter matrix of the associated Coxeter system.

OUTPUT: CoxeterMatrix

EXAMPLES:

sage: FCA3 = FullyCommutativeElements(['A', 3])
sage: FCA3.coxeter_matrix()
[1 3 2]
[3 1 3]
[2 3 1]
sage: FCB5 = FullyCommutativeElements(['B', 5])
sage: FCB5.coxeter_matrix()
[1 3 2 2 2]
[3 1 3 2 2]
[2 3 1 3 2]
[2 2 3 1 4]
[2 2 2 4 1]

index_set()

Obtain the set of the generators/simple reflections of the associated Coxeter system.
OUTPUT: iterable of integers

EXAMPLES:

```python
sage: FCA3 = FullyCommutativeElements(['A', 3])
sage: FCA3.index_set()
[1, 2, 3]
sage: FCB5 = FullyCommutativeElements(['B', 5])
sage: FCB5.index_set()
[1, 2, 3, 4, 5]
```

**iterate_to_length**(length)

Iterate through the elements of this class up to a maximum length.

**INPUT:**

- length – integer; maximum length of element to generate.

**OUTPUT:** generator for elements of self of length up to length

**EXAMPLES:**

The following example produces all FC elements of length up to 2 in the group $A_3$:

```python
sage: FCA3 = FullyCommutativeElements(['A', 3])
sage: list(FCA3.iterate_to_length(2))
[[], [1], [2], [3], [2, 1], [1, 3], [1, 2], [3, 2], [2, 3], [3, 2, 1], [2, 1, 3], [1, 3, 2], [1, 2, 3], [2, 1, 3, 2]]
```

The lists for length 4 and 5 are the same since 4 is the maximum length of an FC element in $A_3$:

```python
sage: list(FCA3.iterate_to_length(4))
[[], [1], [2], [3], [2, 1], [1, 3], [1, 2], [3, 2], [2, 3], [3, 2, 1], [2, 1, 3], [1, 3, 2], [1, 2, 3], [2, 1, 3, 2]]
sage: list(FCA3.iterate_to_length(5))
[[], [1], [2], [3], [2, 1], [1, 3], [1, 2], [3, 2], [2, 3], [3, 2, 1], [2, 1, 3], [1, 3, 2], [1, 2, 3], [2, 1, 3, 2]]
sage: list(FCA3.iterate_to_length(4)) == list(FCA3)  
True
```

The following example produces all FC elements of length up to 4 in the affine Weyl group $\tilde{A}_2$:

```python
sage: FCAffineA2 = FullyCommutativeElements(['A', 2, 1])
sage: FCAffineA2.category()
Category of infinite enumerated sets
sage: list(FCAffineA2.iterate_to_length(4))
[[], [0], [1], [2], [1, 0], [2, 0], [0, 1], [2, 1], [0, 2], [1, 2], [2, 1, 0], [1, 2, 0], [0, 1, 2], [1, 0, 2], [0, 1], [0, 2, 1], [0, 1, 2], [0, 2, 1, 0], [0, 1, 2, 0], [1, 2, 0, 1], [1, 0, 2, 1], [2, 1, 0, 2], [2, 0, 1, 2]]
```