```
sage: Bijectionist?
Docstring:
   A toolbox to list all possible bijections between
   two finite sets under various constraints.

sage: sage.combinat.bijectionist?
File: ~/sage/src/sage/combinat/bijectionist.py
Docstring:
A bijectionist's toolkit

AUTHORS:

* Alexander Grosz, Tobias Kietreiber, Stephan Pfannerer
  and Martin Rubey (2020-2022): Initial version
```

Part I: Find bijections

Given two finite sets A, B of the same cardinality,

and some further constraints,

find all possible bijections $S : A \to B$.

Part I: Find bijections

Given two finite sets A, B of the same cardinality,

and some further constraints,

find all possible bijections $S : A \to B$.

```
sage: N = 3
sage: A = [D for n in range(1, N+1) for D in DyckWords(n)]
sage: B = [M for n in range(1, N+1) for M in PerfectMatchings(2*n)
           if M.is_noncrossing()]
```

Part I: Find bijections

Given two finite sets A, B of the same cardinality,

and some further constraints,

find all possible bijections $S : A \to B$.

```
sage: N = 3
sage: A = [D for n in range(1, N+1) for D in DyckWords(n)]
sage: B = [M for n in range(1, N+1) for M in PerfectMatchings(2*n)
           if M.is_noncrossing()]
sage: b = Bijectionist(A, B)
sage: view(list(b.minimal_subdistributions_iterator()))
```

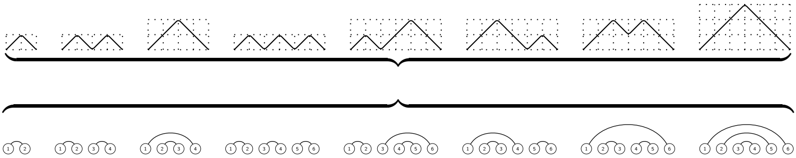
Part I: Find bijections

Given two finite sets A, B of the same cardinality,

and some further constraints,

find all possible bijections $S : A \to B$.

```
sage: N = 3
sage: A = [D for n in range(1, N+1) for D in DyckWords(n)]
sage: B = [M for n in range(1, N+1) for M in PerfectMatchings(2*n)
           if M.is_noncrossing()]
sage: b = Bijectionist(A, B)
sage: view(list(b.minimal_subdistributions_iterator()))
```



Statistics

Set constraints of the form $\alpha = \beta \circ S$.
($\alpha : A \to W$, $\beta : B \to W$ for any set W)

Statistics

Set constraints of the form $\alpha = \beta \circ S$.
($\alpha : A \to W$, $\beta : B \to W$ for any set W)

```
sage: a1 = lambda D: D.semilength()
sage: b1 = lambda M: len(M)

sage: a2 = lambda D: D.area()
sage: b2 = lambda M: M.number_of_nestings()

sage: b.set_statistics((a1, b1), (a2, b2))
sage: view(list(b.minimal_subdistributions_iterator()))
```

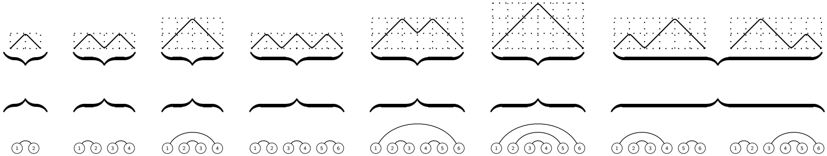
Statistics

Set constraints of the form $\alpha = \beta \circ S$.
($\alpha : A \to W$, $\beta : B \to W$ for any set W)

```
sage: a1 = lambda D: D.semilength()
sage: b1 = lambda M: len(M)

sage: a2 = lambda D: D.area()
sage: b2 = lambda M: M.number_of_nestings()

sage: b.set_statistics((a1, b1), (a2, b2))
sage: view(list(b.minimal_subdistributions_iterator()))
```



Intertwining relations

Add restrictions of the form $S\big(\pi(a_1, ..., a_k)\big) = \rho\big(S(a_1), ..., S(a_k)\big)$.
($\pi : A^k \to A$, $\rho : B^k \to B$ for any $k \in \mathbb{N}$)

Intertwining relations

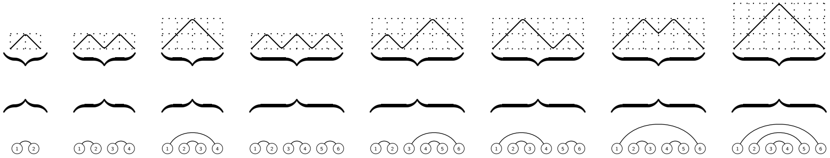
Add restrictions of the form $S\big(\pi(a_1, \ldots, a_k)\big) = \rho\big(S(a_1), \ldots, S(a_k)\big)$.
($\pi : A^k \to A$, $\rho : B^k \to B$ for any $k \in \mathbb{N}$)

```
sage: pi = lambda D1, D2: DyckWord(list(D1) + list(D2))
sage: rho = lambda M1, M2: PerfectMatching(list(M1)
                                            + [(o + M1.size(),
                                                c + M1.size())
                                               for o, c in M2])
sage: b.set_intertwining_relations((2, pi, rho))
sage: view(list(b.minimal_subdistributions_iterator()))
```

Intertwining relations

Add restrictions of the form $S\big(\pi(a_1, \ldots, a_k)\big) = \rho\big(S(a_1), \ldots, S(a_k)\big)$.
$(\pi : A^k \to A,\ \rho : B^k \to B$ for any $k \in \mathbb{N})$

```
sage: pi = lambda D1, D2: DyckWord(list(D1) + list(D2))
sage: rho = lambda M1, M2: PerfectMatching(list(M1)
                                           + [(o + M1.size(),
                                               c + M1.size())
                                              for o, c in M2])
sage: b.set_intertwining_relations((2, pi, rho))
sage: view(list(b.minimal_subdistributions_iterator()))
```



Interlude

Ask `FindStat` whether the map is known.

Interlude

Ask FindStat whether the map is known.

```
sage: findmap(list(b.minimal_subdistributions_iterator()))
0: Mp00146 (quality [100])
sage: _[0].info()
your input matches
    Mp00146: to tunnel matching: Dyck paths -> Perfect matchings

among the values you sent, 100 percent are actually in the database
```

Part II: Find statistics

Given two finite sets A, B of the same cardinality,

a statistic $\tau : B \to Z$, and some further constraints,

find all statistics $s : A \to Z$

such that there is a bijection $S : A \to B$ with $s = \tau \circ S$.

(finding a bijection is the special case $Z = B$, $\tau = \mathrm{id_B}$ and $s = S$)

Part II: Find statistics

Given two finite sets A, B of the same cardinality,
a statistic $\tau : B \to Z$, and some further constraints,
find all statistics $s : A \to Z$
such that there is a bijection $S : A \to B$ with $s = \tau \circ S$.

(finding a bijection is the special case $Z = B$, $\tau = \mathrm{id}_{\mathrm{B}}$ and $s = S$)

```
sage: A = B = [pi for n in range(4) for pi in Permutations(n)]
sage: tau = Permutation.longest_increasing_subsequence_length
sage: b = Bijectionist(A, B, tau)
sage: b.set_statistics((len, len))

sage: def rotate(pi):
....:     cycle = Permutation(tuple(range(1, len(pi) + 1)))
....:     return cycle*pi*cycle.inverse()
sage: b.set_constant_blocks(orbit_decomposition(A, rotate))
sage: list(b.minimal_subdistributions_iterator())
```

Part II: Find statistics

Given two finite sets A, B of the same cardinality,
a statistic $\tau : B \to Z$, and some further constraints,
find all statistics $s : A \to Z$
such that there is a bijection $S : A \to B$ with $s = \tau \circ S$.

(finding a bijection is the special case $Z = B$, $\tau = \mathrm{id}_B$ and $s = S$)

```
sage: A = B = [pi for n in range(4) for pi in Permutations(n)]
sage: tau = Permutation.longest_increasing_subsequence_length
sage: b = Bijectionist(A, B, tau)
sage: b.set_statistics((len, len))

sage: def rotate(pi):
....:     cycle = Permutation(tuple(range(1, len(pi) + 1)))
....:     return cycle*pi*cycle.inverse()
sage: b.set_constant_blocks(orbit_decomposition(A, rotate))
sage: list(b.minimal_subdistributions_iterator())

[([[]], [0]), ([[1]], [1]),
 ([[1, 3, 2]], [2]), ([[2, 1, 3]], [2]), ([[3, 2, 1]], [2]),
 ([[1, 2], [2, 1]], [1, 2]),
 ([[1, 2, 3], [2, 3, 1], [3, 1, 2]], [1, 2, 3])]
```

Find counterexamples

$s : A \to Z$ is homomesic with respect to a set partition Q of A if the average $\frac{1}{|p|} \sum_{a \in p} s(a)$ is the same for all blocks $p \in Q$.

Find counterexamples

$s : A \to Z$ is homomesic with respect to a set partition Q of A if the average $\frac{1}{|p|}\sum_{a\in p} s(a)$ is the same for all blocks $p \in Q$.

```
sage: A = B = [pi for n in range(4) for pi in Permutations(n)]
sage: tau = Permutation.longest_increasing_subsequence_length
sage: b = Bijectionist(A, B, tau)
sage: b.set_statistics((len, len))

sage: def rotate(pi):
....:     cycle = Permutation(tuple(range(1, len(pi) + 1)))
....:     return cycle*pi*cycle.inverse()
sage: b.set_homomesic(orbit_decomposition(A, rotate))
sage: list(b.minimal_subdistributions_iterator())
```

Find counterexamples

$s : A \to Z$ is homomesic with respect to a set partition Q of A if the average $\frac{1}{|p|} \sum_{a \in p} s(a)$ is the same for all blocks $p \in Q$.

```
sage: A = B = [pi for n in range(4) for pi in Permutations(n)]
sage: tau = Permutation.longest_increasing_subsequence_length
sage: b = Bijectionist(A, B, tau)
sage: b.set_statistics((len, len))

sage: def rotate(pi):
....:     cycle = Permutation(tuple(range(1, len(pi) + 1)))
....:     return cycle*pi*cycle.inverse()
sage: b.set_homomesic(orbit_decomposition(A, rotate))
sage: list(b.minimal_subdistributions_iterator())

RuntimeError: generator raised StopIteration
```

All constraints

`set_statistics`	Declare statistics that are preserved by S.
`set_value_restrictions`	Restrict values of s on an element.
`set_distributions`	Restrict distribution of values of s.
`set_constant_blocks`	Declare that s is constant on some sets.
`set_intertwining_relations`	Declare that s intertwines with other maps.
`set_quadratic_relation`	Declare that s satisfies a quadratic relation.
`set_homomesic`	Declare that s is homomesic with respect to a given set partition.

Modelling the problem as a mixed integer linear program

Given A, B, $\tau : B \to Z$, P a set partition of A,
find all $s : P \to Z$ with $s = \tau \circ S$.

Variables

$$x_{p,z} = 1 \Leftrightarrow s(p) = z \quad \text{for} \quad p \in P, z \in \tau(A)$$

Modelling the problem as a mixed integer linear program

Given A, B, $\tau : B \to Z$, P a set partition of A,
find all $s : P \to Z$ with $s = \tau \circ S$.

Variables

$$x_{p,z} = 1 \Leftrightarrow s(p) = z \quad \text{for} \quad p \in P, z \in \tau(A)$$

s takes exactly one value on each block:

Constraint

$$\forall p \in P : \quad \sum_{z} x_{p,z} = 1$$

Modelling statistics $s = \tau \circ S, \ \alpha = \beta \circ S$

The total number of elements $a \in A$ with $(s(a), \alpha(a)) = (z, c)$ must equal the total number of $b \in B$ with $(\tau(b), \beta(b)) = (z, c)$.

Modelling statistics $s = \tau \circ S, \ \alpha = \beta \circ S$

The total number of elements $a \in A$ with $(s(a), \alpha(a)) = (z, c)$ must equal the total number of $b \in B$ with $(\tau(b), \beta(b)) = (z, c)$.
Let $m_c(p)$ be the *multiplicity* of an α-value c in a block $p \in P$

$$m_c(p) = \#\{a \in p : \alpha(a) = c\}$$

Let $n_c(z)$ be the *number* of elements in B with β-value c and τ-value z

$$n_c(z) = \#\{b \in B : \tau(b) = z, \beta(b) = c\}.$$

We require for all value pairs (z, c)

$$n_c(z) \stackrel{!}{=} \#\{a \in A : (s(a), \alpha(a)) = (z, c)\}.$$

Modelling statistics $s = \tau \circ S,\ \alpha = \beta \circ S$

$$\begin{aligned}
n_c(z) &\overset{!}{=} \#\{a \in A : (s(a), \alpha(a)) = (z, c)\} \\
&= \sum_{p \in P} \#\{a \in p : s(a) = s(p) = z, \alpha(a) = c\} \\
&= \sum_{\substack{p \in P \\ s(p) = z}} \#\{a \in p : \alpha(a) = c\} \\
&= \sum_{p \in P} x_{p,z} \cdot m_c(p)
\end{aligned}$$

Modelling statistics $s = \tau \circ S, \ \alpha = \beta \circ S$

$$
\begin{aligned}
n_c(z) &\overset{!}{=} \#\{a \in A : (s(a), \alpha(a)) = (z, c)\} \\
&= \sum_{p \in P} \#\{a \in p : s(a) = s(p) = z, \alpha(a) = c\} \\
&= \sum_{\substack{p \in P \\ s(p) = z}} \#\{a \in p : \alpha(a) = c\} \\
&= \sum_{p \in P} x_{p,z} \cdot m_c(p)
\end{aligned}
$$

Constraint

$$
\forall c \in \beta(B), \ z \in \tau(B) : \sum_{p \in P} x_{p,z} \cdot m_c(p) = n_c(z)
$$