monoide de compositions

Franco Saliola

8/3/2014

```
class MonoideDeCompositions(Parent):
    def __init__(self, ensemble):
        Parent.__init__(self, category = FiniteMonoids())
        self._ensemble = Set(ensemble)

    def _repr_(self):
        return "Monoide de compositions de %s" % self._ensemble

    def product(self, x, y):
        new_composition = []
        for B in x.value:
            for C in y.value:
                BC = B.intersection(C)
                if BC:
                    new_composition.append(BC)
        return self.create_element(new_composition)

    @cached_method
    def one(self):
        return self.create_element([self._ensemble])

    @cached_method
    def semigroup_generators(self):
        return [self.one()] + [self.create_element(X) for X in \
            OrderedSetPartitions(self._ensemble, 2)]

    def an_element(self):
        return self.semigroup_generators()[0]

    def create_element(self, x):
        return self(self.Element.wrapped_class(map(Set, x)))

    def regions(self):
        return [x for x in self if len(x.value) == len(self._ensemble)]

    class Element (ElementWrapper):
        wrapped_class = tuple

        def shape(self):
            return Composition([len(B) for B in self.value])
        def _repr_(self):
            return '[%s]' % '|'.join(''.join(map(str, block)) for block \
               in self.value)
```

```
M = MonoideDeCompositions([1,2,3])
M
```

```
Monoide de compositions de {1, 2, 3}
```

```
M.one()
```

```
[123]
```

```
M.list()
```

```
[[1|23], [1|3|2], [1|2|3], [13|2], [23|1], [123], [12|3], [3|12], [2|13], [3|2|1],
[2|3|1], [3|1|2], [2|1|3]]
```

```
M.cardinality()
```

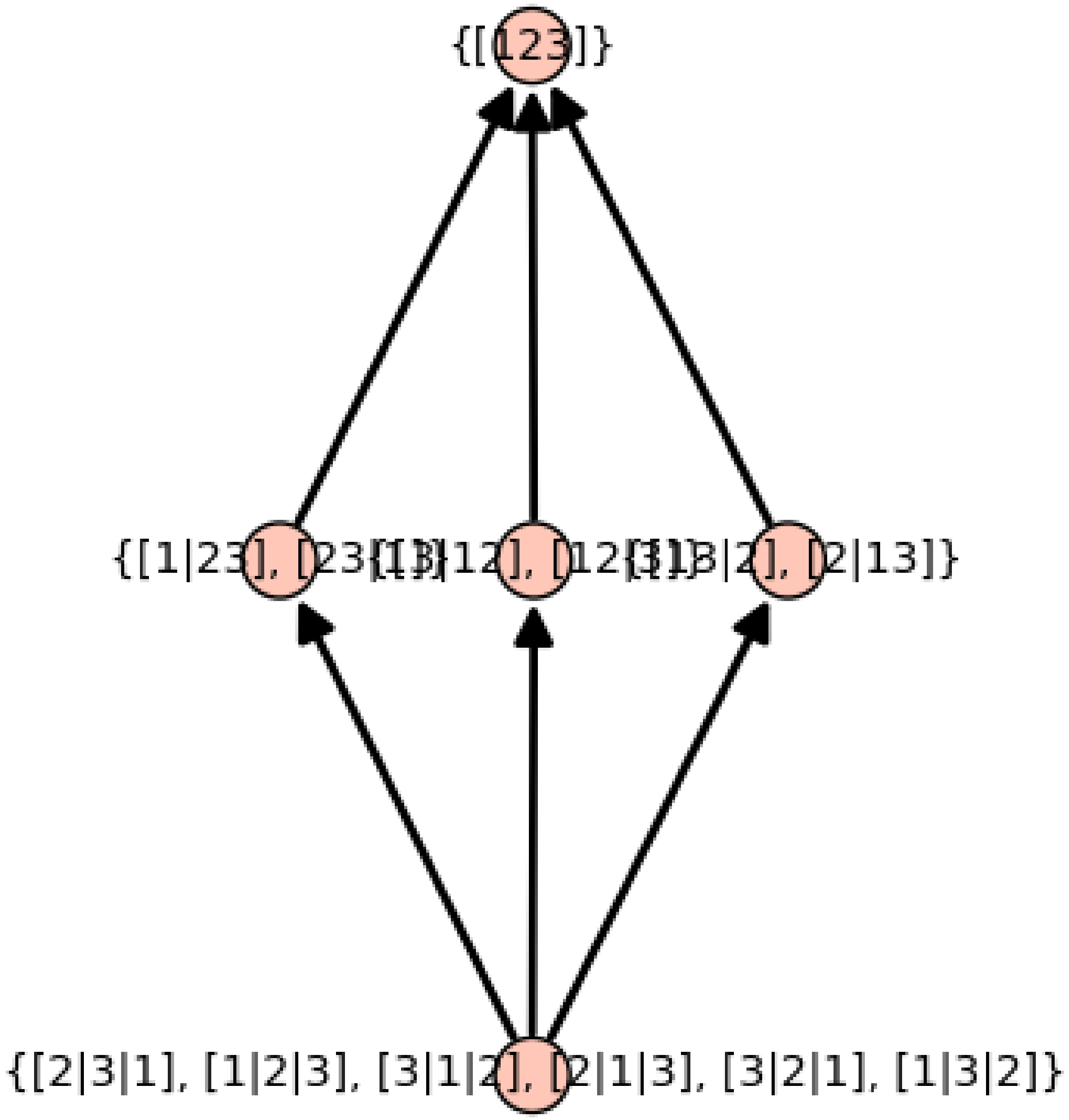
```
13
```

```
M.regions()
```

```
[[1|3|2], [1|2|3], [3|2|1], [2|3|1], [3|1|2], [2|1|3]]
```

```
def treillis_de_support(M):
    supports = map(Set, M.j_classes())
    def ordre_sur_supports(X, Y):
        return X[0] * Y[0] == X[0]
    relns = [(x,y) for x in supports for y in supports if \
       ordre_sur_supports(x,y)]
    L = Poset([supports, relns])
    return L
```

```
treillis_de_support(M).plot()
```



```
prob = {}
for x in M:
    if len(x.value) == 2 and len(x.value[0]) == 1:
        prob[x] = random_prime(10)
```

```
prob
{[1|23]: 5, [2|13]: 2, [3|12]: 3}
```

```
for x in prob:
    prob[x] = prob[x]/sum(prob.values())
prob
{[1|23]: 1/2, [2|13]: 4/11, [3|12]: 66/85}
```

```
def matrice_de_transition(M, prob):
    C = M.regions()
    T = matrix(QQ, len(C), len(C))
    for i in range(len(C)):
        for j in range(len(C)):
            T[i, j] = sum(prob[x] for x in prob if x * C[i] == C[j])
    return T
```

```
T = matrice_de_transition(M, prob)
T
[  1/2     0     0     0 66/85  4/11]
[    0   1/2     0     0 66/85  4/11]
[  1/2     0 66/85  4/11     0     0]
[    0   1/2 66/85  4/11     0     0]
[  1/2     0     0  4/11 66/85     0]
[    0   1/2 66/85     0     0  4/11]
```

```
v = vector((1,0,0,0,0,0))
v
(1, 0, 0, 0, 0, 0)
```

```
v * T
(1/2, 0, 0, 0, 66/85, 4/11)
```

```
for m in range(10):
    print m, (v * T**m).N(digits=4)
0 (1.000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000)
1 (0.5000, 0.0000, 0.0000, 0.0000, 0.7765, 0.3636)
2 (0.6382, 0.1818, 0.2824, 0.2824, 0.9911, 0.3140)
3 (0.9559, 0.3891, 0.6823, 0.5658, 1.406, 0.4124)
4 (1.522, 0.6836, 1.289, 0.9652, 2.136, 0.6390)
5 (2.474, 1.144, 2.247, 1.597, 3.372, 1.035)
6 (4.046, 1.888, 3.788, 2.624, 5.427, 1.692)
7 (6.630, 3.102, 6.292, 4.305, 8.821, 2.773)
8 (10.87, 5.090, 10.38, 7.061, 14.41, 4.547)
9 (17.83, 8.349, 17.07, 11.58, 23.58, 7.458)
```

```
T.eigenvalues()
[3067/1870, 66/85, 1/2, 4/11, 0, 0]
```

```
T.is_diagonalizable()
True
```