monoides de faces (sage)

Franco Saliola

3/3/2014

Contents

```
class FaceMonoid(Parent):
    def __init__(self, generators):
        Parent.__init__(self, category = FiniteMonoids())
        self._semigroup_generators = generators

    def _repr_(self):
        return "Un mono de de vecteurs signes engendr  par: %s" % (self\
           ._semigroup_generators)

    def one(self):
        length = len(self._semigroup_generators[0])
        return self((0,) * length)

    def product(self, x, y):
        xv = x.value
        yv = y.value
        z = tuple([xv[i] if xv[i] != 0 else yv[i] for i in range(len(xv))\
           ])
        return self(z)

    def semigroup_generators(self):
        return [self.one()] + [self(g) for g in self.\
           _semigroup_generators]

    class Element(ElementWrapper):
        wrapped_class = tuple

        def _repr_(self):
            d = {0:'0', 1:'+', -1:'-'}
            return "".join([d[i] for i in self.value])
```

```
F = FaceMonoid([ (0,1), (1,0), (0,-1), (-1,0) ])
```

```
x = F.one(); x
```

```
00
```

```
x * x
```

```
00
```

```
F.list()
```
```
[0+, -+, 00, -0, --, 0-, +0, +-, ++]
```

```
F.cardinality()
```
```
9
```

```
F.semigroup_generators()
```
```
[00, 0+, +0, 0-, -0]
```

```
F.multiplication_table()
```
```
*  a b c d e f g h i
 +------------------
a| a b a b b a i i i
b| b b b b b b b b b
c| a b c d e f g h i
d| b b d d e e d e b
e| e e e e e e e e e
f| f e f e e f h h h
g| i i g g h h g h i
h| h h h h h h h h h
i| i i i i i i i i i
```

```
F.multiplication_table(names='elements')
```
```
 *  0+ -+ 00 -0 -- 0- +0 +- ++
  +---------------------------
0+| 0+ -+ 0+ -+ -+ 0+ ++ ++ ++
-+| -+ -+ -+ -+ -+ -+ -+ -+ -+
00| 0+ -+ 00 -0 -- 0- +0 +- ++
-0| -+ -+ -0 -0 -- -- -0 -- -+
--| -- -- -- -- -- -- -- -- --
0-| 0- -- 0- -- -- 0- +- +- +-
+0| ++ ++ +0 +0 +- +- +0 +- ++
+-| +- +- +- +- +- +- +- +- +-
++| ++ ++ ++ ++ ++ ++ ++ ++ ++
```

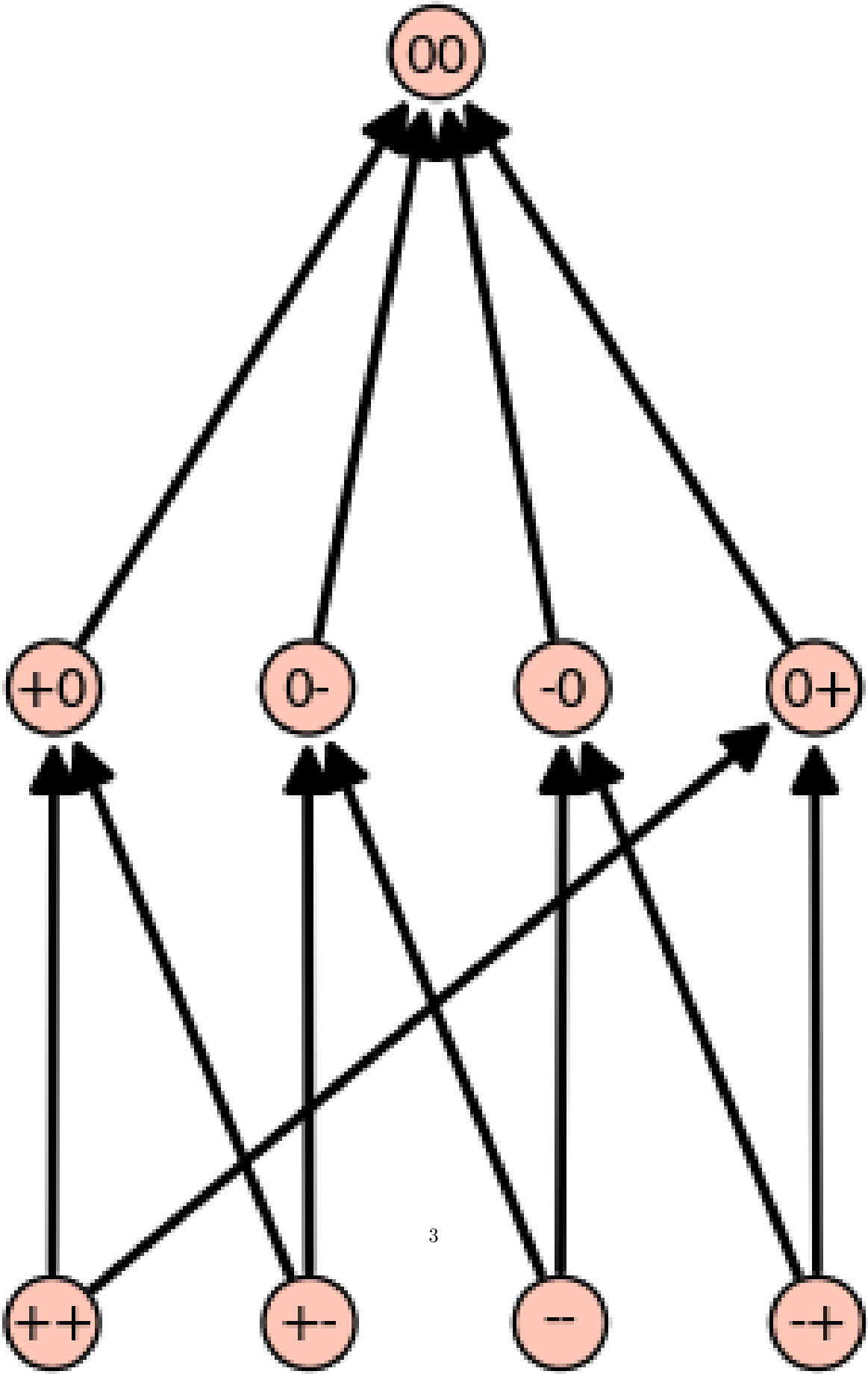
```
[F.random_element() for i in range(10)]
```
```
[-+, +-, +-, 0+, -+, -0, ++, 0+, 00, -0]
```

```
def relation_de_face(x, y):
    r"""
    x <= y ssi y * x == x
    """
    return y * x == x
```

```
relns = [(x,y) for x in F for y in F if relation_de_face(x,y)]
```

```
P = Poset([F, relns]); P
```
```
Finite poset containing 9 elements
```

```
P.plot()
```


00
+0
0-
-0
0+
++
+-
--
-+

```
F.j_classes()
```
```
[[0+, 0-], [-+, +-, ++, --], [00], [+0, -0]]
```

```
supports = map(Set, F.j_classes()); supports
```
```
[{0+, 0-}, {-+, +-, ++, --}, {00}, {+0, -0}]
```

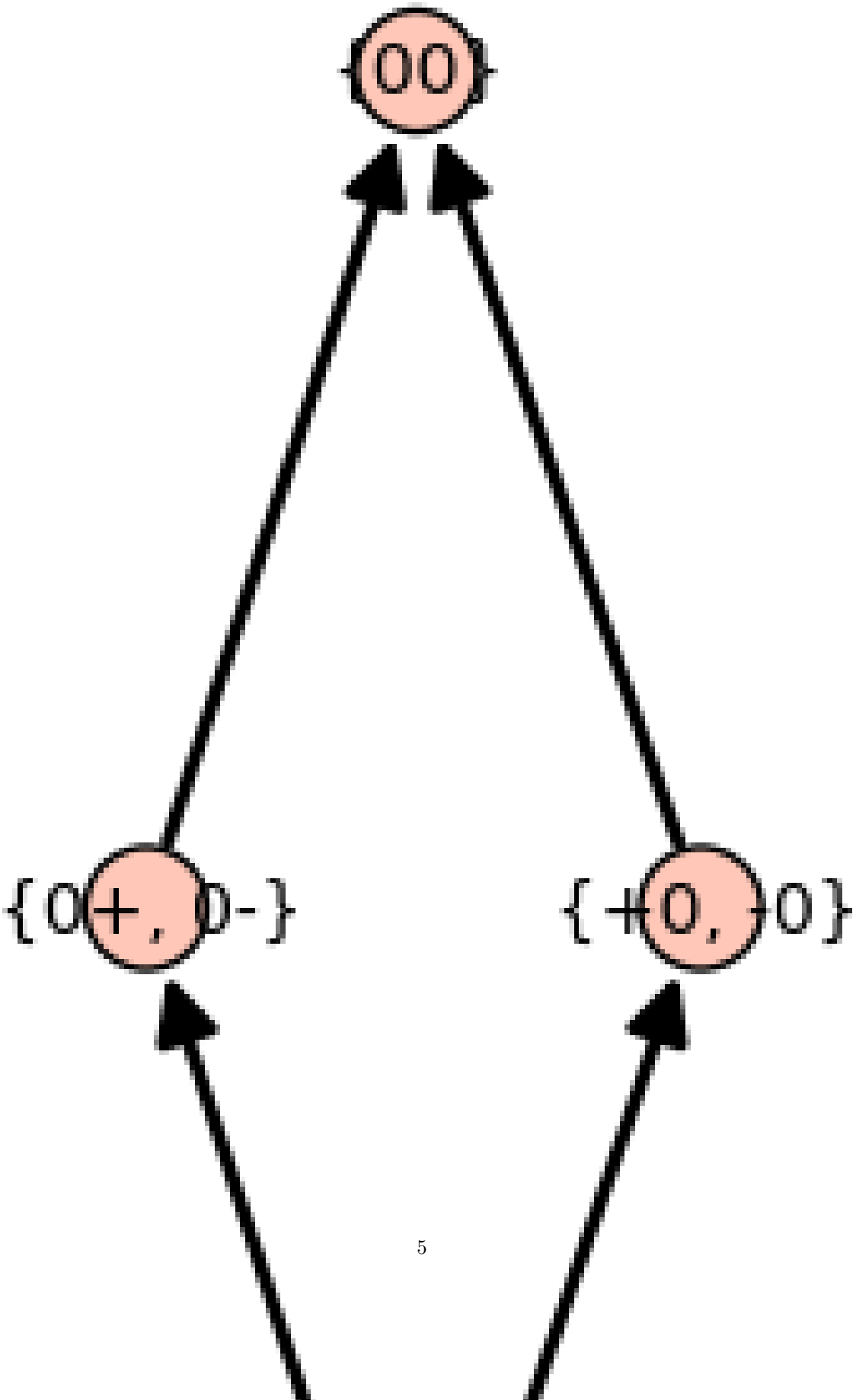
```
def ordre_sur_supports(X, Y):
    return X[0] * Y[0] == X[0]
```

```
relns = [(x,y) for x in supports for y in supports if ordre_sur_supports(\
   x,y)]
```

```
L = LatticePoset([supports, relns])
```

```
L
```
```
Finite lattice containing 4 elements
```

```
L.plot()
```


{0 0}
{0 +, 0-}
{+ 0, -0}

```
F = FaceMonoid([(1,-1,0), (-1,1,0), (1,0,1), (-1,0,-1), (0,1,1), \
   (0,-1,-1)])
```

```
F
```
Un monoïde de vecteurs signes engendré par: [(1, -1, 0), (-1, 1, 0), (1, 0, 1), (-1, 0, -1), (0, 1, 1), (0, -1, -1)]

```
F.cardinality()
```
13

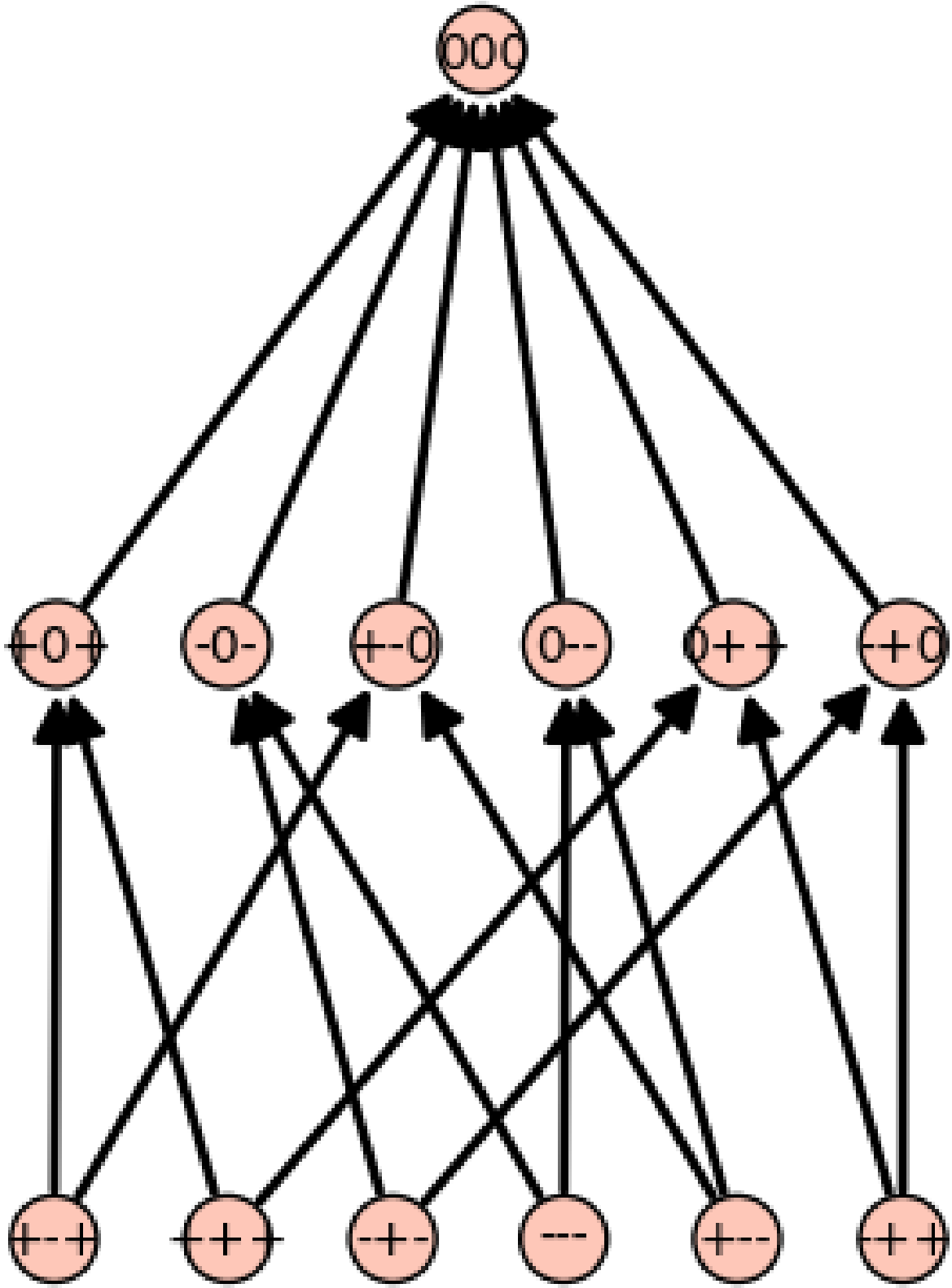
```
F.list()
```
[-+0, 0++, 000, 0--, -++, +--, -0-, +-0, +0+, ---, -+-, +++, +-+]

```
F.multiplication_table(names='elements')
```
```
  *  -+0 0++ 000 0-- -++ +-- -0- +-0 +0+ --- -+- +++ +-+
   +-------------------------------------------------
-+0| -+0 -++ -+0 -+- -++ -+- -+- -+0 -++ -+- -+- -++ -++
0++| -++ 0++ 0++ 0++ -++ +++ -++ +++ +++ -++ -++ +++ +++
000| -+0 0++ 000 0-- -++ +-- -0- +-0 +0+ --- -+- +++ +-+
0--| --- 0-- 0-- 0-- --- +-- --- +-- +-- --- --- +-- +--
-++| -++ -++ -++ -++ -++ -++ -++ -++ -++ -++ -++ -++ -++
+--| +-- +-- +-- +-- +-- +-- +-- +-- +-- +-- +-- +-- +--
-0-| -+- -+- -0- --- -+- --- -0- --- -0- --- -+- -+- ---
+-0| +-0 +-+ +-0 +-- +-+ +-- +-- +-0 +-+ +-- +-- +-+ +-+
+0+| +++ +++ +0+ +-+ +++ +-+ +0+ +-+ +0+ +-+ +++ +++ +-+
---| --- --- --- --- --- --- --- --- --- --- --- --- --- ---
-+-| -+- -+- -+- -+- -+- -+- -+- -+- -+- -+- -+- -+- -+- -+-
+++| +++ +++ +++ +++ +++ +++ +++ +++ +++ +++ +++ +++ +++ +++
+-+| +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+
```

```
def relation_de_face(x, y):
    r"""
    x <= y ssi y * x == x
    """
    return y * x == x

relns = [(x,y) for x in F for y in F if relation_de_face(x,y)]
P = Poset([F, relns]); P
```
Finite poset containing 13 elements

```
P.plot()
```


000
+0+
-0-
+-0
0--
0++
-+0
+--+
+++
+-+-

+--
-++

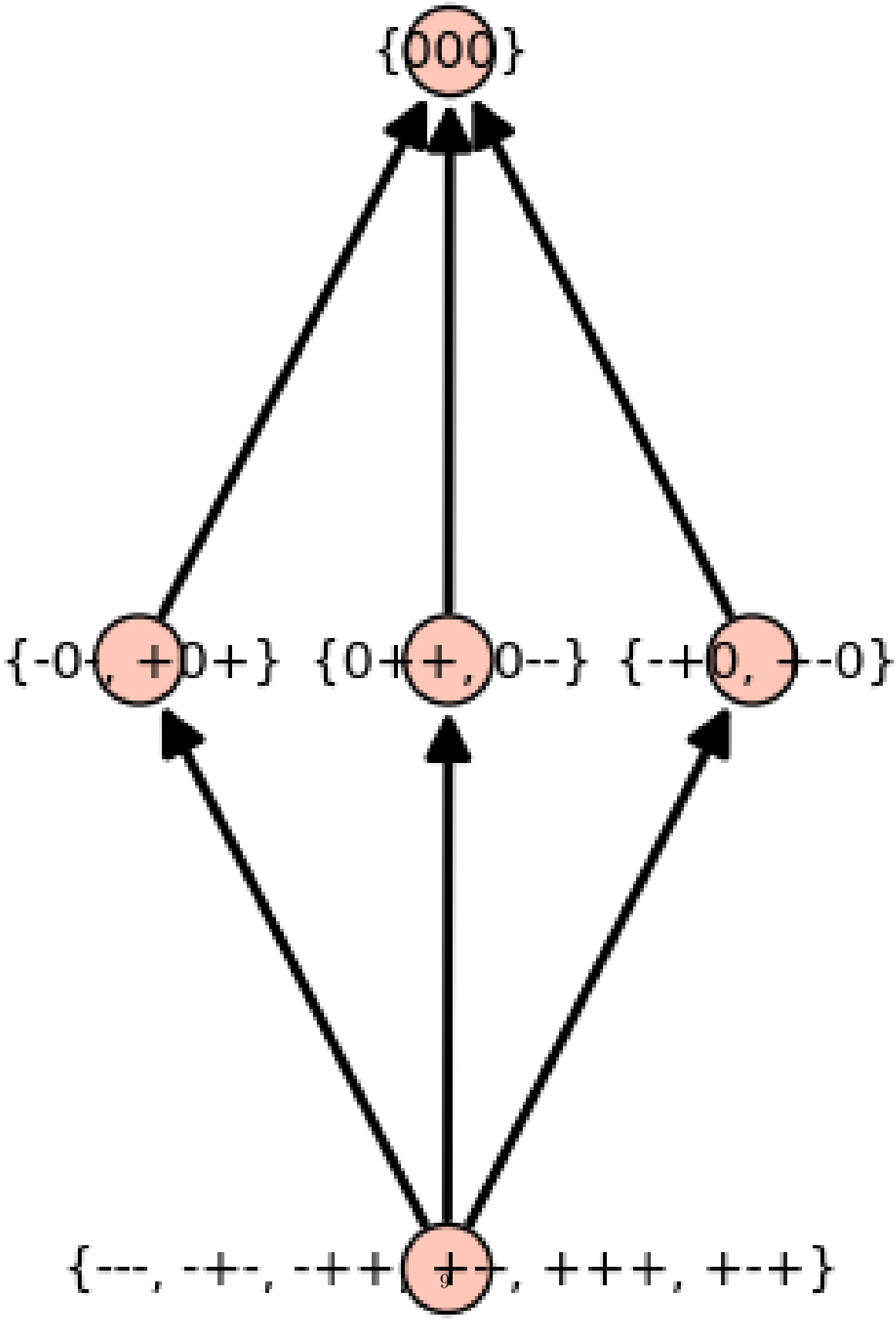
```
supports = map(Set, F.j_classes())

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 = LatticePoset([supports, relns])
L
```

```
Finite lattice containing 5 elements
```

```
L.plot()
```


{000}
{-0+, +0+}
{0++, 0--}
{-+0, +-0}
{--, -+-, -++, +--, +++, +-+}

```
A = F.algebra(QQ)
```

```
A
```
```
Free module generated by Un monoïde de vecteurs signes engendré par: [(1, -1, 0), (-1, 1, 0), (1, 0, 1), (-1, 0, -1), (0, 1, 1), (0, -1, -1)] over Rational Field
```

```
x = A.an_element()
```

```
x
```
```
2*B[-+0] + 2*B[0++] + 3*B[000]
```

```
x * x
```
```
16*B[-+0] + 16*B[0++] + 9*B[000] + 8*B[-++]
```

```
A.dimension()
```
```
13
```

```
A.category()
```
```
Category of monoid algebras over Rational Field
```

```
p = A.sum_of_monomials([F((1,0,1)), F((0,-1,-1)), F((-1, 1,0))])
p
```
```
B[-+0] + B[0--] + B[+0+]
```