In [3]:
```
elements = range(10)
elements
```

Out[3]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [4]:
```
relation = lambda x, y : x <= y
```

In [5]:
```
E = Poset([elements, relation])
```

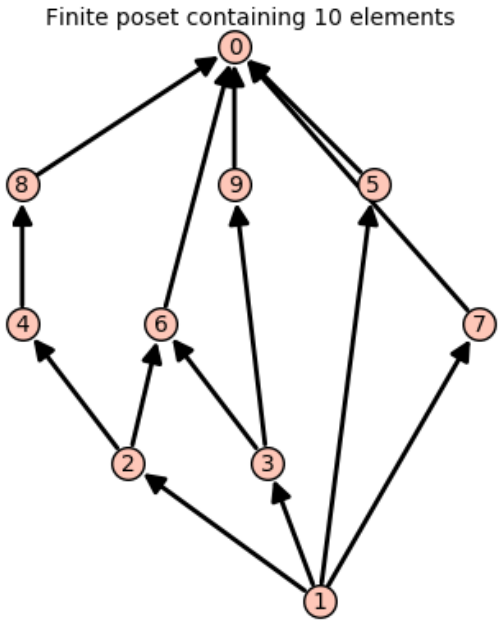
In [6]:
```
E
```

Out[6]:

Finite poset containing 10 elements

In [7]:
```
elements = range(10)
relation = lambda x, y : False if x == 0 else Integer(y)/Integer(x) in ZZ
E = Poset((elements, relation))
E
```

Out[7]:



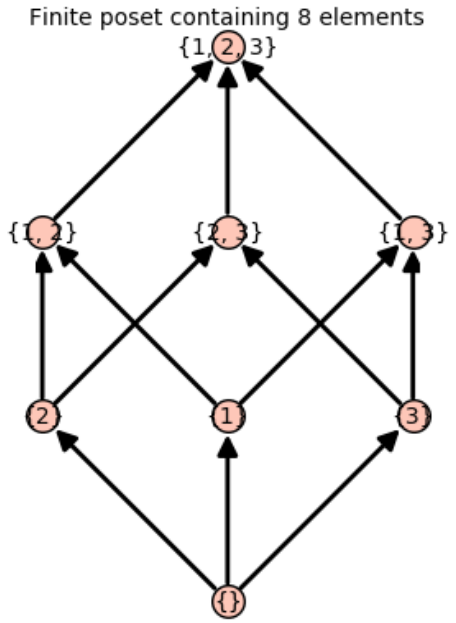
In [8]:
```
view(E)
```

In [9]:
```
E.is_lattice()
```

Out[9]: True

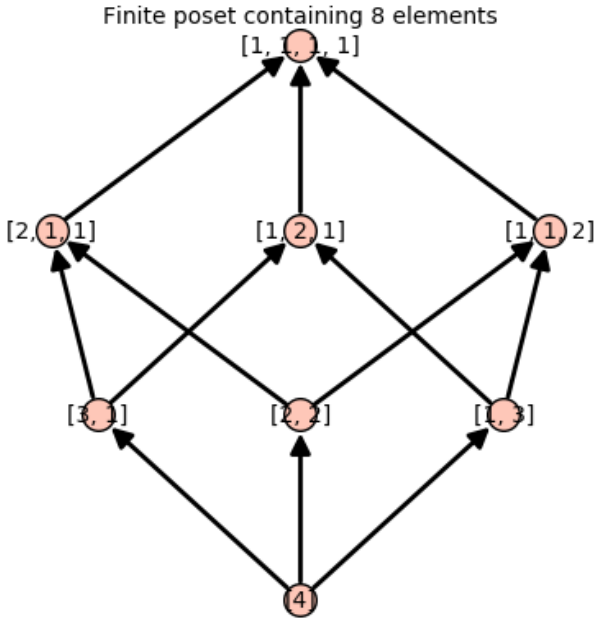
In [11]:
```
elements = Subsets([1,2,3])
relation = lambda U, V : U.issubset(V)
E = Poset((elements, relation))
E
```

Out[11]:



In [14]:
```
E.relabel(lambda U : Composition(from_subset=(U, 4)))
```

Out[14]:

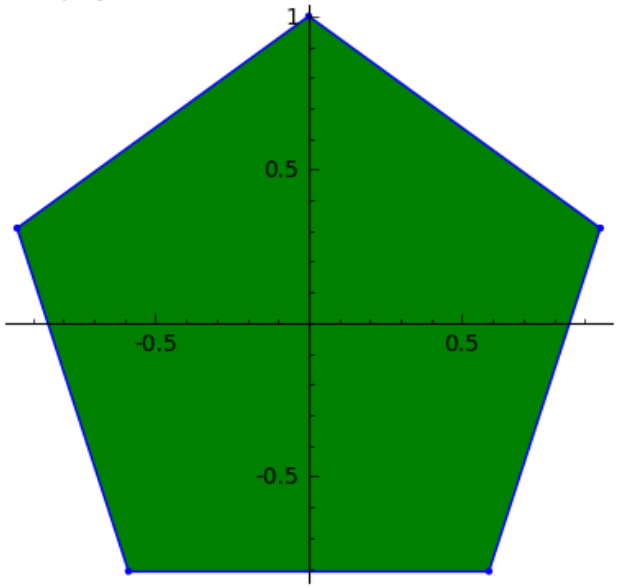


In [15]:
```
P = polytopes.regular_polygon(5)
```

In [16]:
```
P
```

Out[16]:

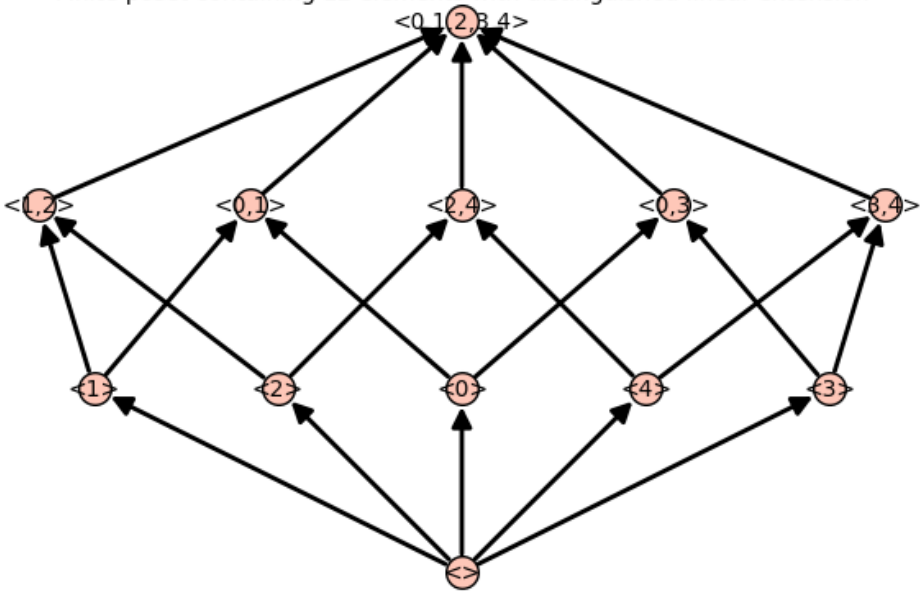
A 2-dimensional polyhedron in AA^2 defined as the convex hull of 5 vertices



In [17]:
```
P.face_lattice()
```

Out[17]:

Finite poset containing 12 elements with distinguished linear extension



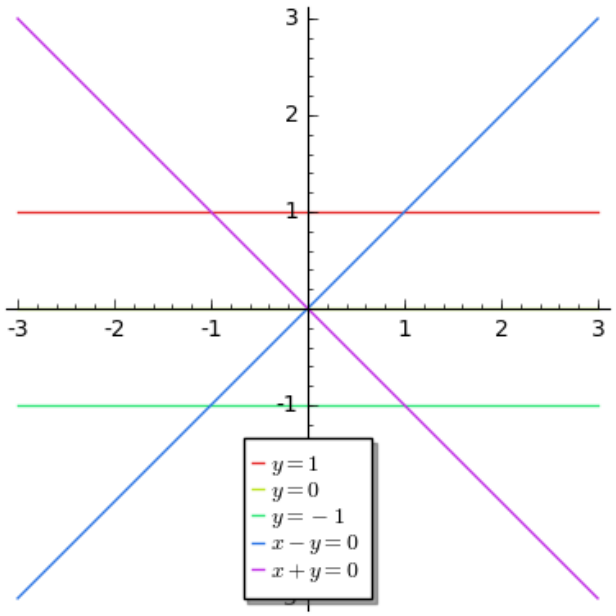
In [18]:
```
H.<x,y> = HyperplaneArrangements(QQ)
```

In [20]:
```
A = H(x + y, x - y, y - 1, y, y + 1)
A
```

Out[20]:
```
Arrangement of 5 hyperplanes of dimension 2 and rank 2
```

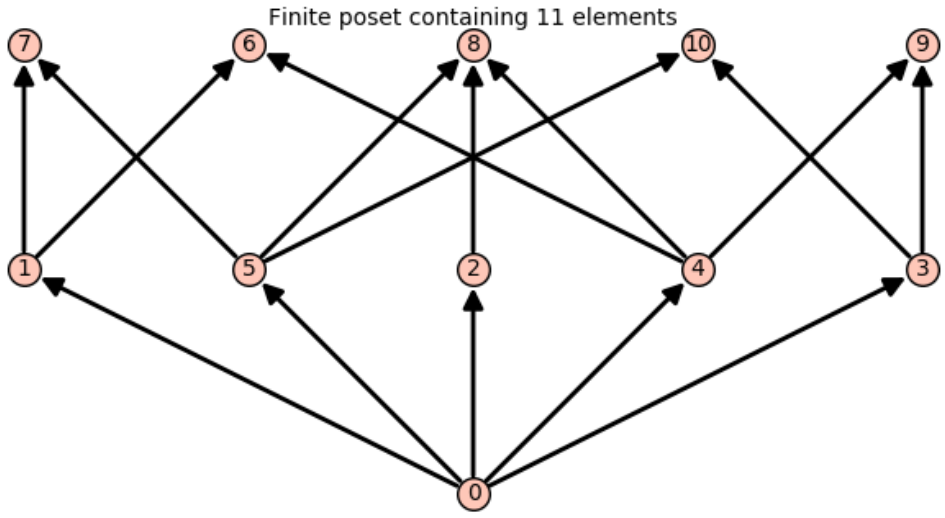
In [22]:
```
A.plot()
```

Out[22]:



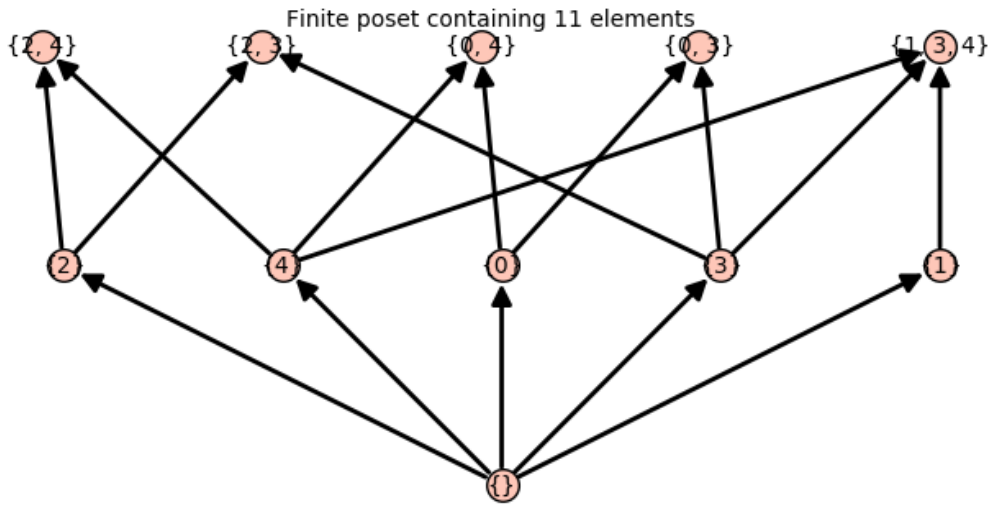
In [23]:
```
A.intersection_poset()
```

Out[23]:



In [24]:
```
T = A.intersection_poset(element_class = "subset")
T
```

Out[24]:



In [25]:
```
view(T)
```

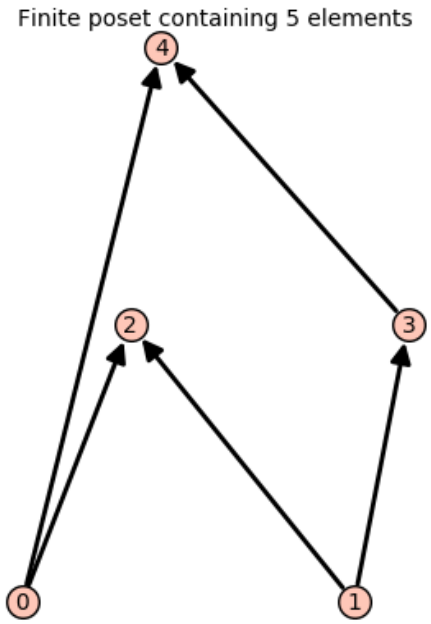
In [26]:
```
d = {U : LatexExpr(" \cap ".join(["H_{{{}}}".format(i) for i in sorted(U)])
) for U in T}
d[Set([])] = LatexExpr("V")
d
```

Out[26]:
```
{{0, 3}: H_{0} \cap H_{3},
 {2}: H_{2},
 {1}: H_{1},
 {2, 3}: H_{2} \cap H_{3},
 {4}: H_{4},
 {1, 3, 4}: H_{1} \cap H_{3} \cap H_{4},
 {}: V,
 {0, 4}: H_{0} \cap H_{4},
 {0}: H_{0},
 {3}: H_{3},
 {2, 4}: H_{2} \cap H_{4}}
```

In [27]:
```
S = T.relabel(d)
view(S)
```

In [29]:
```
# Devoir 1, Ex 1
E = Poset([[0,1,2,3,4], [(0,2),(0,4),(1,2),(1,3),(3,4),(1,4)]])
E
```

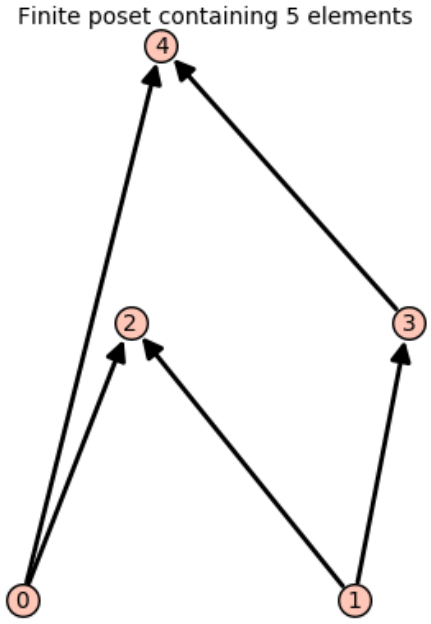
Out[29]:



In [30]:
```
view(E)
```

In [31]:
```
E = Poset([[2,4], [2,3], [], [4], []])
E
```

Out[31]:



In [34]:
```
m = E.moebius_function_matrix()
```

In [35]:
```
show(m)
```

$$\begin{pmatrix} 1 & -1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

In [36]:
```
latex(m)
```

Out[36]:
```
\left(\begin{array}{rrrrr}
1 & -1 & 0 & -1 & 0 \\
0 & 1 & 0 & 0 & -1 \\
0 & 0 & 1 & -1 & -1 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{array}\right)
```

In [37]:
```
E.list()
```

Out[37]:
```
[1, 3, 0, 2, 4]
```

In [38]:
```
mu = E.moebius_function
m = {}
for i in range(5):
    for j in range(5):
        m[i, j] = mu(i, j)
```

In [39]:
```
m
```

Out[39]:
```
{(0, 0): 1,
 (0, 1): 0,
 (0, 2): -1,
 (0, 3): 0,
 (0, 4): -1,
 (1, 0): 0,
 (1, 1): 1,
 (1, 2): -1,
 (1, 3): -1,
 (1, 4): 0,
 (2, 0): 0,
 (2, 1): 0,
 (2, 2): 1,
 (2, 3): 0,
 (2, 4): 0,
 (3, 0): 0,
 (3, 1): 0,
 (3, 2): 0,
 (3, 3): 1,
 (3, 4): -1,
 (4, 0): 0,
 (4, 1): 0,
 (4, 2): 0,
 (4, 3): 0,
 (4, 4): 1}
```

In [40]:
```
m = matrix(m)
```

In [41]:
```
m
```

Out[41]:
```
[ 1  0 -1  0 -1]
[ 0  1 -1 -1  0]
[ 0  0  1  0  0]
[ 0  0  0  1 -1]
[ 0  0  0  0  1]
```

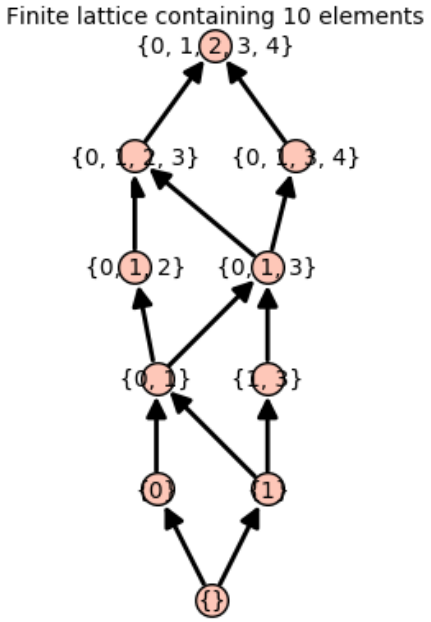
In [42]:
```
z = m.inverse()
```

In [43]:
```
show(m, z)
```

$$\begin{pmatrix} 1 & 0 & -1 & 0 & -1 \\ 0 & 1 & -1 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

In [46]:
```
JE = E.order_ideals_lattice()
JE
```

Out[46]:



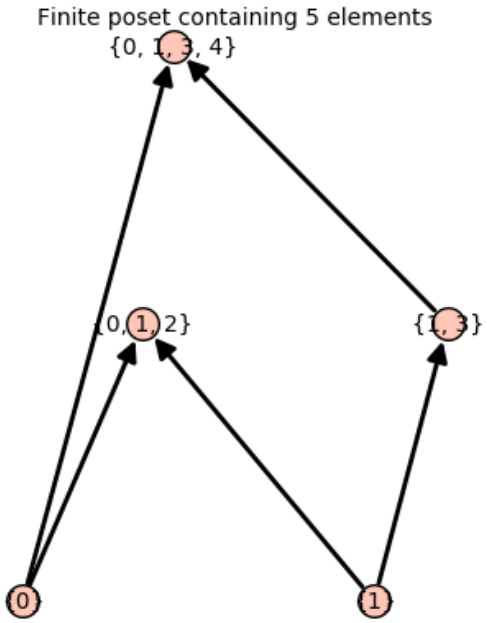
In [47]:
```
view(JE)
```

In [48]:
```
JE.join_irreducibles()
```

Out[48]:
```
[{0}, {1}, {1, 3}, {0, 1, 2}, {0, 1, 3, 4}]
```

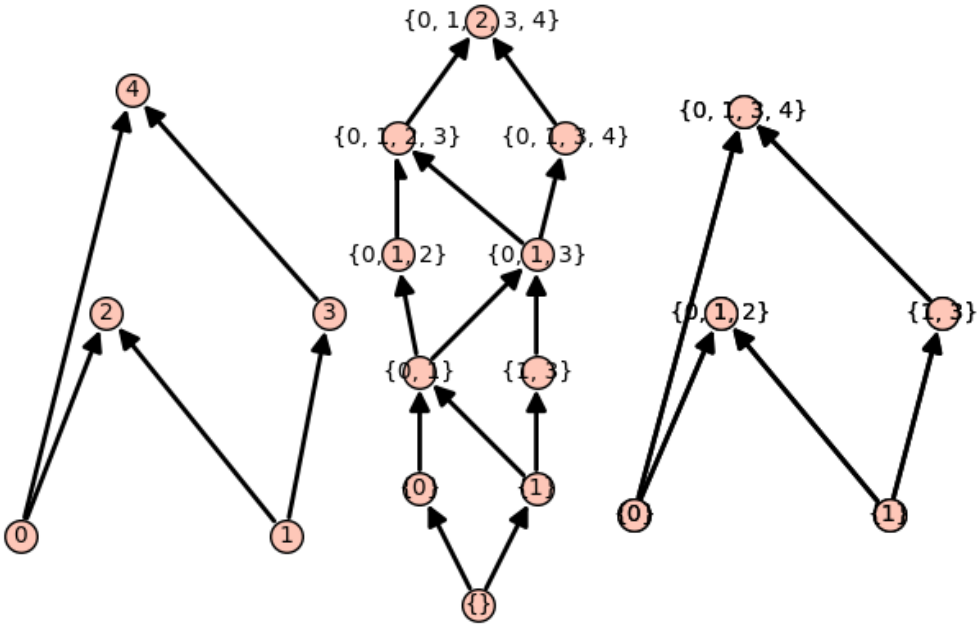
In [49]:
```
JE.join_irreducibles_poset()
```

Out[49]:



In [53]:
```
graphics_array( [ E.plot(), JE.plot(), JE.join_irreducibles_poset().plot()
] )
```

Out[53]:



In [62]:
```
J = lambda E : E.order_ideals_lattice()

J = attrcall('order_ideals_lattice')
```

In [56]:
```
E = Poset()
E
```

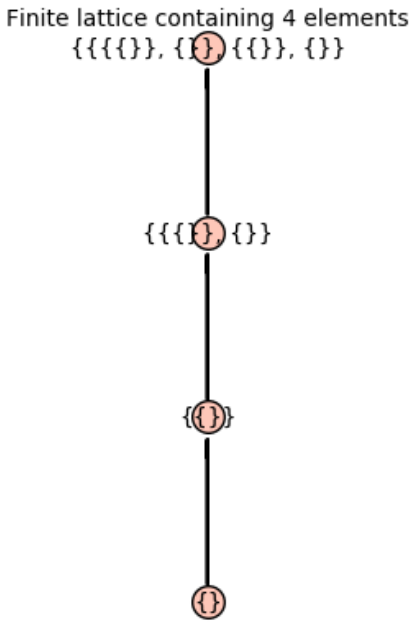
Out[56]: Finite poset containing 0 elements (use the .plot() method to plot)

In [57]:
```
E.plot()
```

Out[57]:

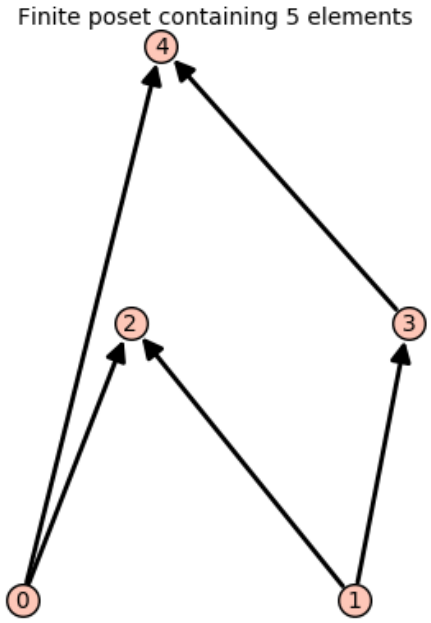
In [61]:
```
J(J(J(J(E))))
```

Out[61]:



In [64]:
```
E = Poset([[2,4], [2,3], [], [4], []])
E
```

Out[64]:

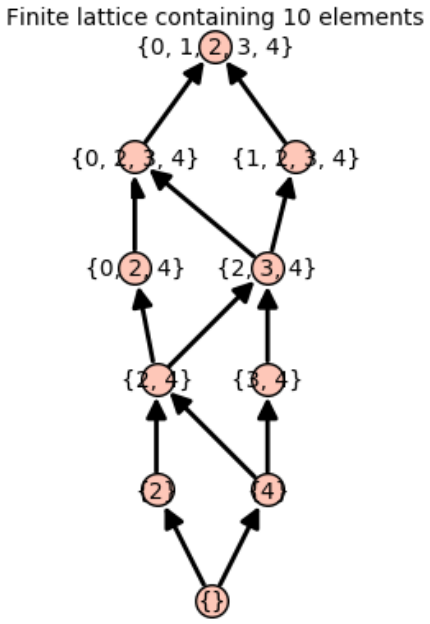


In [65]:
```
E.order_filters_lattice()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-65-d176407ef7dc> in <module>()
----> 1 E.order_filters_lattice()

/home/saliola/Applications/sage/src/sage/structure/category_object.pyx in s
age.structure.category_object.CategoryObject.__getattr__ (/home/saliola/App
lications/sage/src/build/cythonized/sage/structure/category_object.c:7662)(
)
    821             AttributeError: 'PrimeNumbers_with_category' object has
no attribute 'sadfasdf'
    822         """
--> 823         return self.getattr_from_category(name)
    824
    825     cdef getattr_from_category(self, name):

/home/saliola/Applications/sage/src/sage/structure/category_object.pyx in s
age.structure.category_object.CategoryObject.getattr_from_category (/home/s
aliola/Applications/sage/src/build/cythonized/sage/structure/category_objec
t.c:7826)()
    836                 cls = self._category.parent_class
    837
--> 838             attr = getattr_from_other_class(self, cls, name)
    839             self.__cached_methods[name] = attr
    840             return attr

/home/saliola/Applications/sage/src/sage/structure/misc.pyx in sage.structu
re.misc.getattr_from_other_class (/home/saliola/Applications/sage/src/build
/cythonized/sage/structure/misc.c:1866)()
    292         dummy_error_message.cls = type(self)
    293         dummy_error_message.name = name
--> 294         raise dummy_attribute_error
    295     cdef PyObject* attr = _PyType_Lookup(<type>cls, name)
    296     if attr is NULL:

AttributeError: 'FinitePoset_with_category' object has no attribute 'order_
filters_lattice'
```

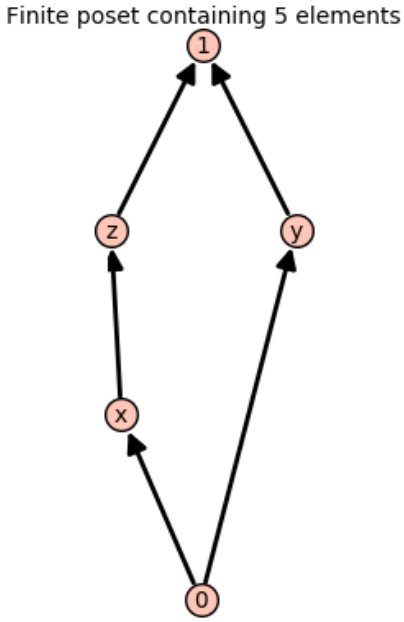
In [67]:
```
E.dual().order_ideals_lattice()
```

Out[67]:



In [68]:
```
elements = [0, 'x', 'y', 'z', 1]
relations = [(0, 'x'), (0, 'y'), ('x', 'z'), ('z', 1), ('y', 1)]
E = Poset((elements, relations))
E
```

Out[68]:



In [69]:
```
E.is_lattice()
```

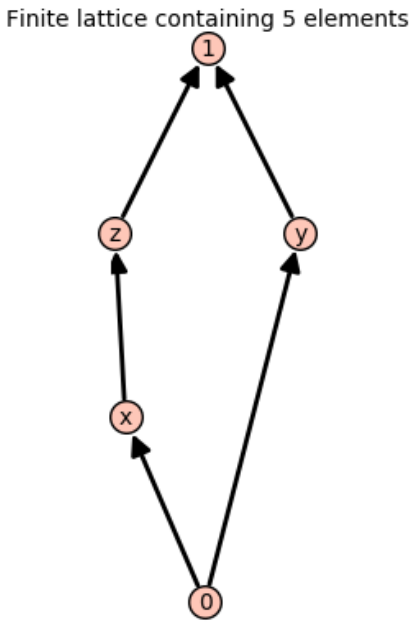
Out[69]: True

In [70]:
```
E is LatticePosets()
```

Out[70]: False

In [71]:
```
E = LatticePoset(E)
E
```

Out[71]:



In [72]:
```
E.is_lattice()
```

Out[72]: True

In [73]:
```
E in LatticePosets()
```

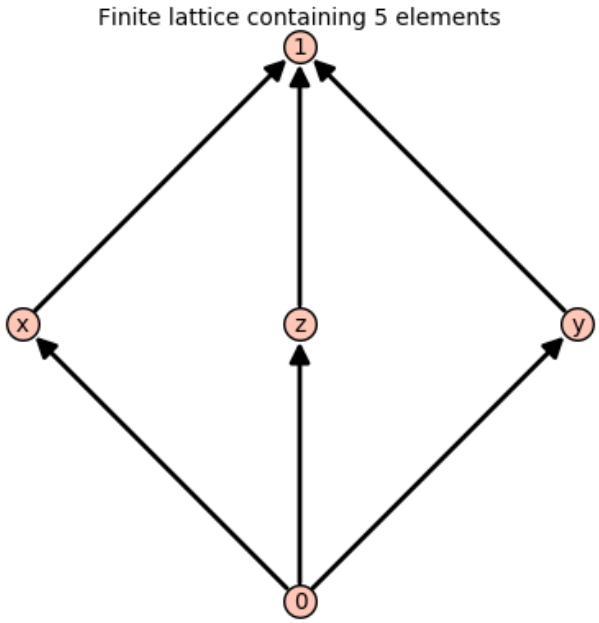
Out[73]: True

In [74]:
```
print E.is_distributive()
print E.is_modular()
print E.is_upper_semimodular()
```

```
False
False
False
```

In [75]:
```
elements = [0, 'x', 'y', 'z', 1]
relations = [(0, 'x'), (0, 'y'), (0, 'z'), ('x', 1), ('z', 1), ('y', 1)]
E = LatticePoset((elements, relations))
E
```

Out[75]:

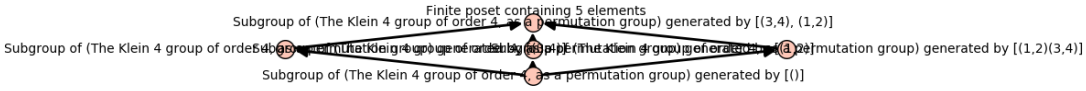


In [76]:
```
print E.is_distributive()
print E.is_modular()
print E.is_upper_semimodular()
```

```
False
True
True
```

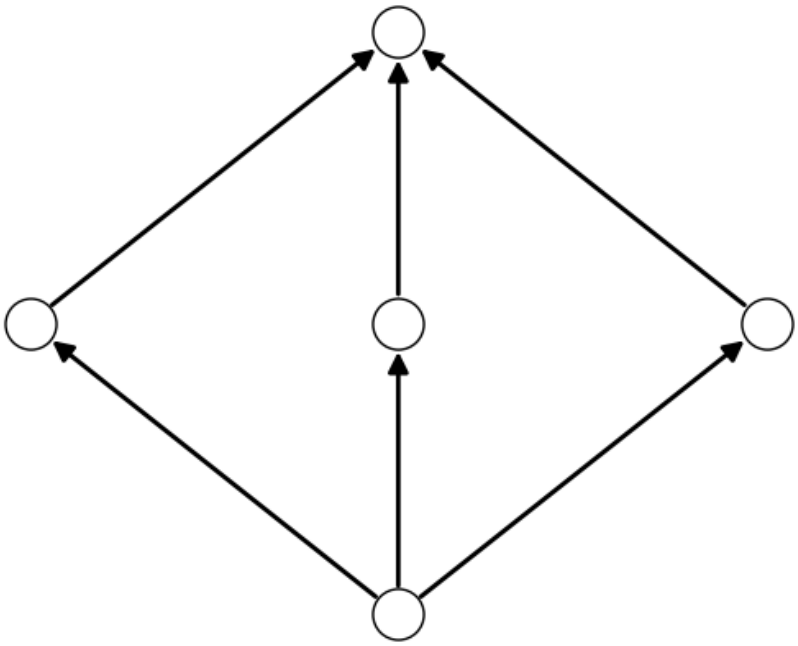
In [79]:
```
elements = KleinFourGroup().subgroups()
relation = attrcall('is_subgroup')
E = Poset((elements, relation))
E
```

Out[79]:



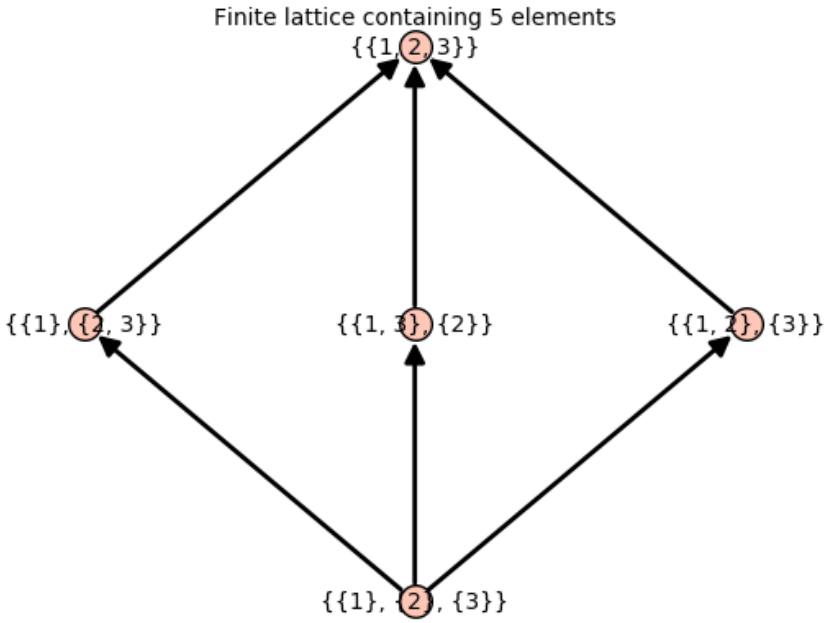
In [81]:
```
E.plot(label_elements=False, talk=True)
```

Out[81]:



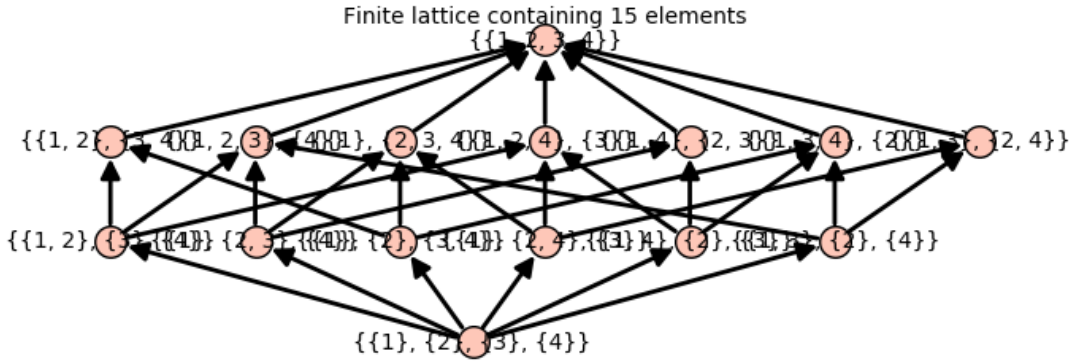
In [83]:
```
E = posets.SetPartitions(3)
E
```

Out[83]:



In [84]:
```
E = posets.SetPartitions(4)
E
```

Out[84]:

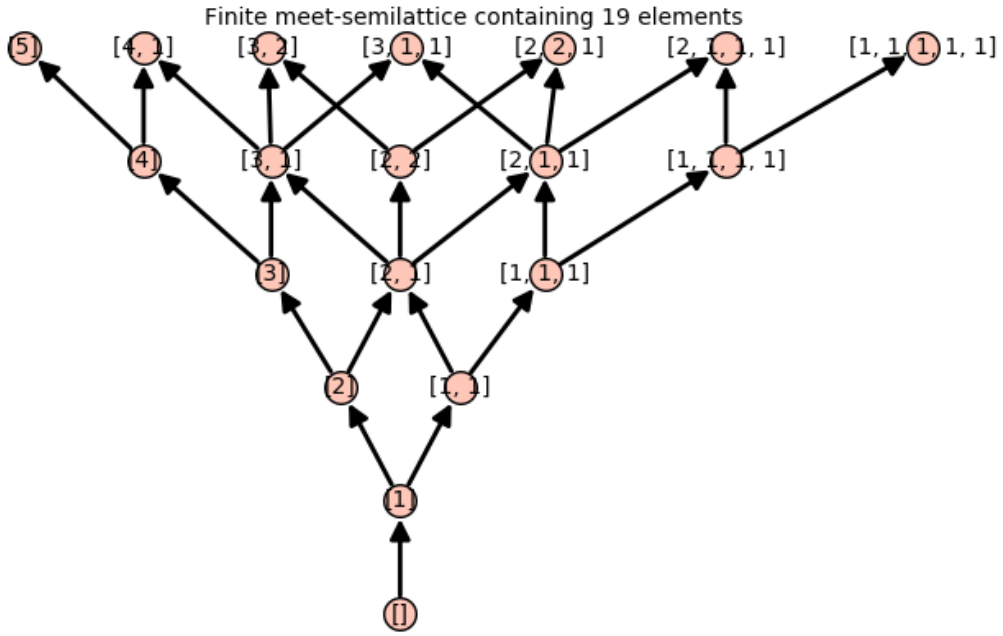


In [85]:
```
print E.is_modular()
```

```
False
```

In [86]:
```
E = posets.YoungsLattice(5)
E
```

Out[86]:



In [92]:
```
view(E)
```

In [89]:
```
Partitions.options()
```

```
Current options for Partitions
  - convention:        English
  - diagram_str:       *
  - display:           list
  - latex:             young_diagram
  - latex_diagram_str: \ast
```

In [90]:
```
Partitions.options(convention='French')
```

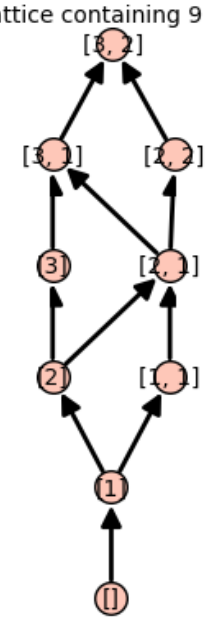
In [91]:
```
Partitions.options()
```

```
Current options for Partitions
  - convention:        French
  - diagram_str:       *
  - display:           list
  - latex:             young_diagram
  - latex_diagram_str: \ast
```

In [93]:
```
E = posets.YoungsLatticePrincipalOrderIdeal(Partition([3,2]))
E
```

Out[93]:

Finite lattice containing 9 elements

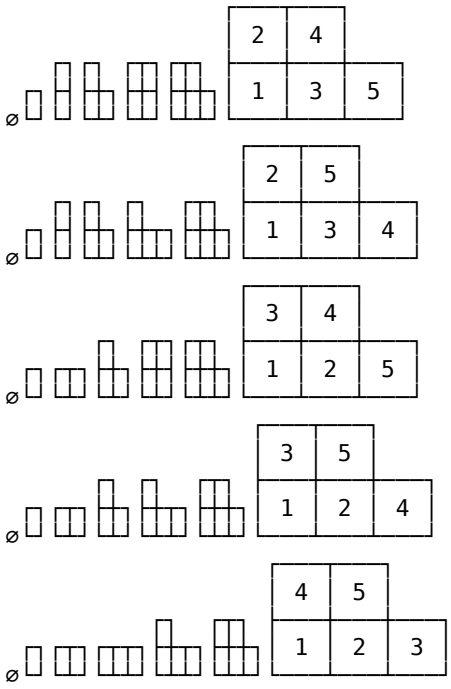


In [94]:
```
view(E)
```

In [96]:
```
E.maximal_chains()
```

Out[96]:
```
[[[], [1], [1, 1], [2, 1], [2, 2], [3, 2]],
 [[], [1], [1, 1], [2, 1], [3, 1], [3, 2]],
 [[], [1], [2], [2, 1], [2, 2], [3, 2]],
 [[], [1], [2], [2, 1], [3, 1], [3, 2]],
 [[], [1], [2], [3], [3, 1], [3, 2]]]
```

In [99]:
```
for chain in E.maximal_chains():
    print unicode_art(*chain) + \
        unicode_art(StandardTableau(SkewTableaux().from_chain(chain)))
```



In [100]:
```
StandardTableaux([3,2]).cardinality()
```

Out[100]: 5

In [103]:
```
L = E.linear_extensions()
L
```

Out[103]: The set of all linear extensions of Finite lattice containing 9 elements

In [104]:
```
L.cardinality()
```

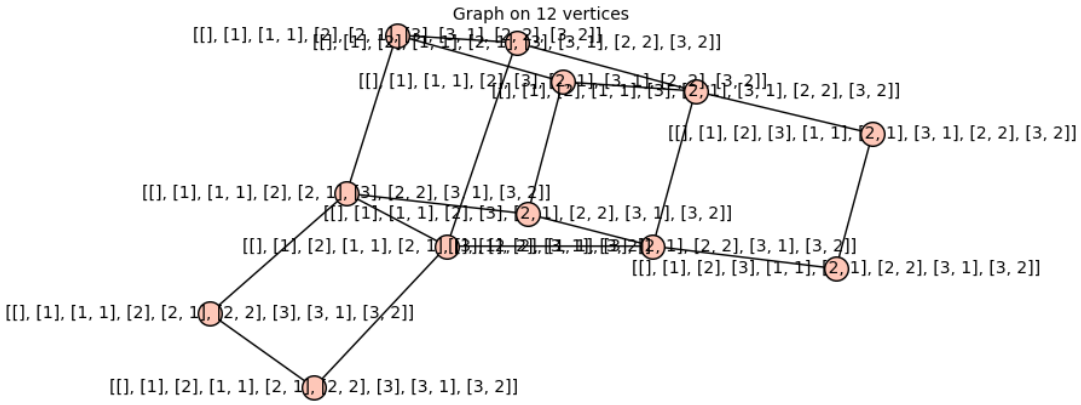
Out[104]: 12

In [105]:
```
L.list()
```

Out[105]:
```
[[[], [1], [1, 1], [2], [2, 1], [2, 2], [3], [3, 1], [3, 2]],
 [[], [1], [1, 1], [2], [2, 1], [3], [2, 2], [3, 1], [3, 2]],
 [[], [1], [1, 1], [2], [2, 1], [3], [3, 1], [2, 2], [3, 2]],
 [[], [1], [1, 1], [2], [3], [2, 1], [2, 2], [3, 1], [3, 2]],
 [[], [1], [1, 1], [2], [3], [2, 1], [3, 1], [2, 2], [3, 2]],
 [[], [1], [2], [1, 1], [2, 1], [2, 2], [3], [3, 1], [3, 2]],
 [[], [1], [2], [1, 1], [2, 1], [3], [2, 2], [3, 1], [3, 2]],
 [[], [1], [2], [1, 1], [2, 1], [3], [3, 1], [2, 2], [3, 2]],
 [[], [1], [2], [1, 1], [3], [2, 1], [2, 2], [3, 1], [3, 2]],
 [[], [1], [2], [1, 1], [3], [2, 1], [3, 1], [2, 2], [3, 2]],
 [[], [1], [2], [3], [1, 1], [2, 1], [2, 2], [3, 1], [3, 2]],
 [[], [1], [2], [3], [1, 1], [2, 1], [3, 1], [2, 2], [3, 2]]]
```

In [106]:
```
E.linear_extensions_graph()
```

Out[106]:

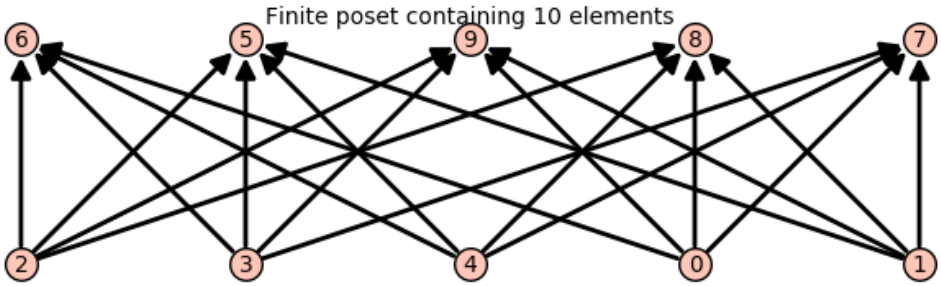


In [ ]:
```
```

In [110]:
```
p = Partition([3,2])
```

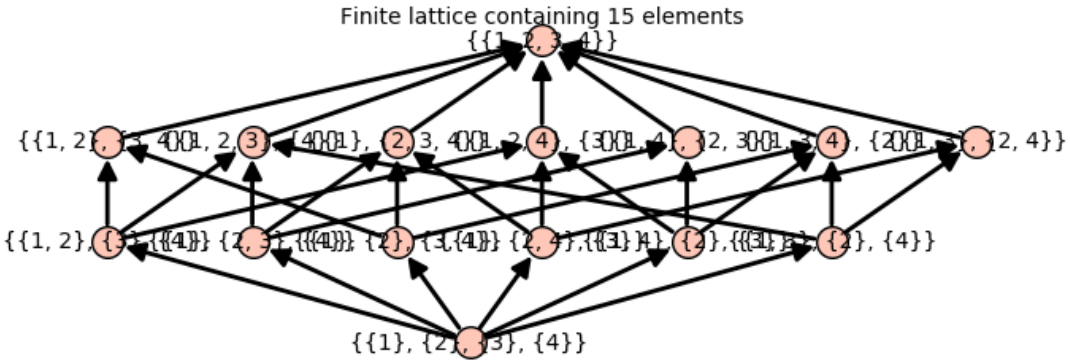
In [113]:
```
posets.StandardExample(5)
```

Out[113]:



In [114]:
```
E = posets.SetPartitions(4)
E
```

Out[114]:



In [115]:
```
E.is_distributive()
```

Out[115]:
```
False
```

In [116]:
```
E.is_modular()
```

Out[116]: False

In [117]:
```
non_distributive_triples = set([])
for x in E:
    for y in E:
        for z in E:
            gauche = E.meet(x, E.join(y, z))
            droite = E.join(E.meet(x, y), E.meet(x, z))
            if gauche != droite:
                non_distributive_triples.add(Set([x,y,z]))
```

In [118]:
```
len(non_distributive_triples)
```

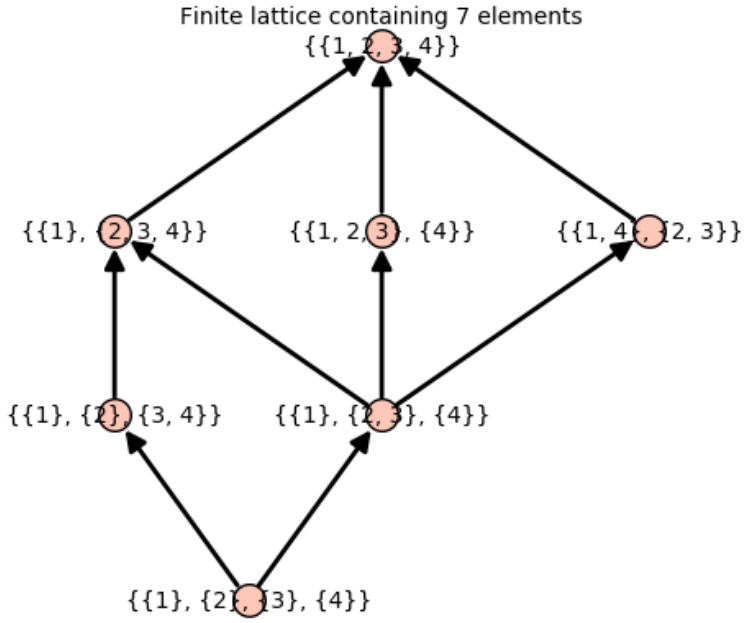
Out[118]: 101

In [120]:
```
xyz = list(non_distributive_triples)[91]
xyz
```

Out[120]: {{{1}, {2}, {3, 4}}, {{1, 2, 3}, {4}}, {{1, 4}, {2, 3}}}

In [121]:
```
E.sublattice(xyz)
```

Out[121]:



In [123]:
```
Sublattices = []
Plots = []
# for xyz in non_distributive_triples:
for (position, xyz) in enumerate(non_distributive_triples):
    print position,
    S = E.sublattice(xyz)
    Sublattices.append(S)
    P = S.plot(label_elements=False,
               border=True,
               element_colors={'black': xyz,
                               'white': [s for s in S if s not in xyz]}
              )
    Plots.append(P)
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 2
8 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 5
3 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 7
8 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

In [124]:
```
ga = graphics_array(Plots, 11, 10)
ga.show(figsize=20)
```

In [ ]: