3 Algorithms of Orbit-Stabilizer Type

We introduce a way to calculate a sufficient part of an orbit and the stabilizer of a point.

3.1 Orbit Stabilizer for Crystallographic Groups

3.1-1 OrbitStabilizerInUnitCubeOnRight

> OrbitStabilizerInUnitCubeOnRight(*group, x*) (method)

Returns: A record containing

- `.stabilizer`: the stabilizer of *x*.
- `.orbit` set of vectors from [0,1)^n which represents the orbit.

Let *x* be a rational vector from [0,1)^n and *group* a space group in standard form. The function then calculates the part of the orbit which lies inside the cube [0,1)^n and the stabilizer of *x*. Observe that every element of the full orbit differs from a point in the returned orbit only by a pure translation.

Note that the restriction to points from [0,1)^n makes sense if orbits should be compared and the vector passed to `OrbitStabilizerInUnitCubeOnRight` should be an element of the returned orbit (part).

```
gap> S:=SpaceGroup(3,5);;
gap> OrbitStabilizerInUnitCubeOnRight(S,[1/2,0,9/11]);
rec( orbit := [ [ 0, 1/2, 2/11 ], [ 1/2, 0, 9/11 ] ],
  stabilizer := Group([ [ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ],
          [ 0, 0, 0, 1 ] ] ]) )
gap> OrbitStabilizerInUnitCubeOnRight(S,[0,0,0]);
rec( orbit := [ [ 0, 0, 0 ] ], stabilizer := <matrix group with 2 generators> )
```

If you are interested in other parts of the orbit, you can use `VectorModOne` (**2.1-2**) for the base point and the functions `ShiftedOrbitPart` (**3.1-9**), `TranslationsToOneCubeAroundCenter` (**3.1-10**) and `TranslationsToBox` (**3.1-11**) for the resulting orbit
Suppose we want to calculate the part of the orbit of `[4/3,5/3,7/3]` in the cube of sidelength `1` around this point:

```
gap> S:=SpaceGroup(3,5);;
gap> p:=[4/3,5/3,7/3];;
gap> o:=OrbitStabilizerInUnitCubeOnRight(S,VectorModOne(p)).orbit;
[ [ 1/3, 2/3, 1/3 ], [ 1/3, 2/3, 2/3 ] ]
gap> box:=p+[[-1,1],[-1,1],[-1,1]];
[ [ 1/3, 8/3, 7/3 ], [ 1/3, 8/3, 7/3 ], [ 1/3, 8/3, 7/3 ] ]
gap> o2:=Concatenation(List(o,i->i+TranslationsToBox(i,box)));;
gap> # This is what we looked for. But it is somewhat large:
gap> Size(o2);
54
```

3.1-2 OrbitStabilizerInUnitCubeOnRightOnSets

> OrbitStabilizerInUnitCubeOnRightOnSets(*group, set*) (method)

Returns: A record containing

- `.stabilizer`: the stabilizer of *set*.
- `.orbit` set of sets of vectors from [0,1)^n which represents the orbit.

Calculates orbit and stabilizer of a set of vectors. Just as `OrbitStabilizerInUnitCubeOnRight` (**3.1-1**), it needs input from [0,1)^n. The returned orbit part `.orbit` is a set of sets such that every element of `.orbit` has a non-trivial intersection with the cube [0,1)^n. In general, these sets will not lie inside [0,1)^n completely.

```
gap> S:=SpaceGroup(3,5);;
gap> OrbitStabilizerInUnitCubeOnRightOnSets(S,[[0,0,0],[0,1/2,0]]);
rec( orbit := [ [ [ -1/2, 0, 0 ], [ 0, 0, 0 ] ],
                [ [ 0, 0, 0 ], [ 0, 1/2, 0 ] ],
                [ [ 1/2, 0, 0 ], [ 1, 0, 0 ] ] ],
  stabilizer := Group([ [ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ],
                        [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ] ]) )
```

3.1-3 OrbitPartInVertexSetsStandardSpaceGroup

> `OrbitPartInVertexSetsStandardSpaceGroup`(*group, vertexset, allvertices*) (method)

Returns: Set of subsets of *allvertices*.

If *allvertices* is a set of vectors and *vertexset* is a subset thereof, then `OrbitPartInVertexSetsStandardSpaceGroup` returns that part of the orbit of *vertexset* which consists entirely of subsets of *allvertices*. Note that,unlike the other `OrbitStabilizer` algorithms, this does not require the input to lie in some particular part of the space.

```
gap> S:=SpaceGroup(3,5);;
gap> OrbitPartInVertexSetsStandardSpaceGroup(S,[[0,1,5],[1,2,0]],
> Set([[1,2,0],[2,3,1],[1,2,6],[1,1,0],[0,1,5],[3/5,7,12],[1/17,6,1/2]]));
[ [ [ 0, 1, 5 ], [ 1, 2, 0 ] ], [ [ 1, 2, 6 ], [ 2, 3, 1 ] ] ]
gap> OrbitPartInVertexSetsStandardSpaceGroup(S, [[1,2,0]],
> Set([[1,2,0],[2,3,1],[1,2,6],[1,1,0],[0,1,5],[3/5,7,12],[1/17,6,1/2]]));
[ [ [ 0, 1, 5 ] ], [ [ 1, 1, 0 ] ], [ [ 1, 2, 0 ] ], [ [ 1, 2, 6 ] ], [ [ 2, 3, 1 ] ] ]
```

3.1-4 OrbitPartInFacesStandardSpaceGroup

> `OrbitPartInFacesStandardSpaceGroup`(*group, vertexset, faceset*) (method)

Returns: Set of subsets of *faceset*.

This calculates the orbit of a space group on sets restricted to a set of faces.
If *faceset* is a set of sets of vectors and *vertexset* is an element of *faceset*, then `OrbitPartInFacesStandardSpaceGroup` returns that part of the orbit of *vertexset* which consists entirely of elements of *faceset*.
Note that,unlike the other `OrbitStabilizer` algorithms, this does not require the input to lie in some particular part of the space.

3.1-5 OrbitPartAndRepresentativesInFacesStandardSpaceGroup

> `OrbitPartAndRepresentativesInFacesStandardSpaceGroup`(*group, vertexset, faceset*) (method)

Returns: A set of face-matrix pairs .

This is a slight variation of `OrbitPartInFacesStandardSpaceGroup` (**3.1-4**) that also returns a representative for every orbit element.


```
gap> S:=SpaceGroup(3,5);;
gap> OrbitPartInVertexSetsStandardSpaceGroup(S,[[0,1,5],[1,2,0]],
> Set([[1,2,0],[2,3,1],[1,2,6],[1,1,0],[0,1,5],[3/5,7,12],[1/17,6,1/2]]));
[ [ [ 0, 1, 5 ], [ 1, 2, 0 ] ], [ [ 1, 2, 6 ], [ 2, 3, 1 ] ] ]
gap> OrbitPartInFacesStandardSpaceGroup(S,[[0,1,5],[1,2,0]],
> Set( [ [ [ 0, 1, 5 ], [ 1, 2, 0 ] ], [[1/17,6,1/2],[1,2,7]]]));
[ [ [ 0, 1, 5 ], [ 1, 2, 0 ] ] ]
gap> OrbitPartAndRepresentativesInFacesStandardSpaceGroup(S,[[0,1,5],[1,2,0]],
> Set( [ [ [ 0, 1, 5 ], [ 1, 2, 0 ] ], [[1/17,6,1/2],[1,2,7]]]));
[ [ [ [ 0, 1, 5 ], [ 1, 2, 0 ] ],
      [ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ] ] ]
```

3.1-6 StabilizerOnSetsStandardSpaceGroup

> `StabilizerOnSetsStandardSpaceGroup`(*group, set*) (method)

Returns: finite group of affine matrices (OnRight)

Given a set *set* of vectors and a space group *group* in standard form, this method calculates the stabilizer of that set in the full crystallographic group.

```
gap> G:=SpaceGroup(3,12);;
gap> v:=[ 0, 0,0 ];;
gap> s:=StabilizerOnSetsStandardSpaceGroup(G,[v]);
<matrix group with 2 generators>
gap> s2:=OrbitStabilizerInUnitCubeOnRight(G,v).stabilizer;
<matrix group with 2 generators>
gap> s2=s;
true
```

3.1-7 RepresentativeActionOnRightOnSets

> `RepresentativeActionOnRightOnSets`(*group, set, imageset*) (method)

Returns: Affine matrix.

Returns an element of the space group S which takes the set *set* to the set *imageset*. The group must be in standard form and act on the right.

```
gap> S:=SpaceGroup(3,5);;
gap> RepresentativeActionOnRightOnSets(G, [[0,0,0],[0,1/2,0]],
>         [ [ 0, 1/2, 0 ], [ 0, 1, 0 ] ]);
[ [ 0, -1, 0, 0 ], [ -1, 0, 0, 0 ], [ 0, 0, -1, 0 ], [ 0, 1, 0, 1 ] ]
```

3.1-8 Getting other orbit parts

HAPcryst does not calculate the full orbit but only the part of it having coefficients between -1/2 and 1/2. The other parts of the orbit can be calculated using the following functions.

3.1-9 ShiftedOrbitPart

> ShiftedOrbitPart(*point, orbitpart*) (method)

Returns: Set of vectors

Takes each vector in *orbitpart* to the cube unit cube centered in *point*.


```
gap> ShiftedOrbitPart([0,0,0],[[1/2,1/2,1/3],-[1/2,1/2,1/2],[19,3,1]]);
[ [ 1/2, 1/2, 1/3 ], [ 1/2, 1/2, 1/2 ], [ 0, 0, 0 ] ]
gap> ShiftedOrbitPart([1,1,1],[[1/2,1/2,1/2],-[1/2,1/2,1/2]]);
[ [ 3/2, 3/2, 3/2 ] ]
```

3.1-10 TranslationsToOneCubeAroundCenter

> TranslationsToOneCubeAroundCenter(*point, center*) (method)

Returns: List of integer vectors

This method returns the list of all integer vectors which translate *point* into the box *center*+[-1/2,1/2]^n

```
gap> TranslationsToOneCubeAroundCenter([1/2,1/2,1/3],[0,0,0]);
[ [ 0, 0, 0 ], [ 0, -1, 0 ], [ -1, 0, 0 ], [ -1, -1, 0 ] ]
gap> TranslationsToOneCubeAroundCenter([1,0,1],[0,0,0]);
[ [ -1, 0, -1 ] ]
```

3.1-11 TranslationsToBox

> TranslationsToBox(*point, box*) (method)

Returns: An iterator of integer vectors or the empty iterator

Given a vector v and a list of pairs, this function returns the translation vectors (integer vectors) which take v into the box *box*. The box *box* has to be given as a list of pairs.

```
gap> TranslationsToBox([0,0],[[1/2,2/3],[1/2,2/3]]);
[  ]
gap> TranslationsToBox([0,0],[[-3/2,1/2],[1,4/3]]);
[ [ -1, 1 ], [ 0, 1 ] ]
gap> TranslationsToBox([0,0],[[-3/2,1/2],[2,1]]);
Error, Box must not be empty called from
...
```