Group Homomorphisms In GAP: Connecting And (De)Composing Groups

Alexander Hulpke
Department of Mathematics
Colorado State University
Fort Collins, CO, 80523, USA
www.math.colostate.edu/~hulpke

Aachen, November 21, 2018

Why Homomorphisms ?

Mathematics can be considered as the study of (classes of) functions between sets.

Structure-preserving functions between groups are group homomorphisms.

They allow to transfer results from one group to another, or to build new groups from two constituents.

Also: Many advanced calculations require them.

Language

A (structure preserving) ***general map*** is defined on a subset of its ***source*** and maps into its ***range***. It can be ***injective***, ***surjective***, ***bijective***. Its ***image*** is the set of all element images.

We can calculate the Image (or ImagesRepresentative) of a source element, PreImagesRepresentative (PreImage is a coset) of a range element, and its KernelOfMultiplicativeGeneralMapping.

Homomorphisms are general maps that are ***total*** and ***single valued***.

Basic Tasks

1. Create Homomorphisms (next)
2. Image of an Element: Image, ImagesRepresentative
3. Pre-Image, PreImagesRepresentative (element that yields particular image)
4. Kernel: KernelOfMultiplicativeGeneralMapping
5. Properties: IsInjective, IsSurjective, IsBijective
6. RestrictedMapping

...And How To Find Them

There are fundamentally three ways how homomorphisms can be created:

- Group Actions
- Listing generators and their images
- A recipe to calculate images
- Induced by conjugation (`InnerAutomorphism`, `ConjugatorAutomorphism`)

Group Actions

G acts on Ω, if $|\Omega|=n$, this gives $\varphi: G \to S_n$.

Operation ActionHomomorphism.

Arguments: Group, Domain, [generators, actors,] action function [,"surjective"].

Points arranged same way as Ω.

```
gap> g:=TransitiveGroup(8,20);;
gap> b:=Blocks(g,MovedPoints(g));
[ [ 1, 5 ], [ 2, 6 ], [ 3, 7 ], [ 4, 8 ] ]
gap> hom:=ActionHomomorphism(g,b,OnSets);
<action homomorphism>
gap> Index(Range(hom),Image(hom));
6
gap> g.2;Image(hom,g.2);
(1,2,3,8)(4,5,6,7)  //  (1,2,3,4)
```

Group Actions

G acts on Ω, if $|\Omega|=n$, this gives $\varphi: G \to S_n$.

Operation ActionHomomorphism.

Arguments: Group, Domain, [generators, actors,] action function [,"surjective"].

Points arranged same way as Ω.

```
gap> g:=TransitiveGroup(8,20);;
gap> b:=Blocks(g,MovedPoints(g
[ [ 1, 5 ], [ 2, 6 ], [ 3, 7
gap> hom:=ActionHomomorphi
<action homomorphism>
gap> Index(Range(hom),Ima
6
gap> g.2;Image(hom,g.2);
(1,2,3,8)(4,5,6,7)  //  (1,2,
```

When calculating images, a **Dictionary** is used to keep track.

Good dictionary methods are implemented for certain common types of objects, depending on domain.

Group A

If group elements act through homomorphic images.

G acts on Ω, if $|\Omega|=n$, this g

Operation ActionHomomorphism.

Arguments: Group, Domain, [generators, actors,] action function [,"surjective"].

Points arranged same way as Ω.

```
gap> g:=TransitiveGroup(8,20);;
gap> b:=Blocks(g,MovedPoints(g));
[ [ 1, 5 ], [ 2, 6 ], [ 3, 7 ], [ 4, 8 ] ]
gap> hom:=ActionHomomorphism(g,b,OnSets);
<action homomorphism>
gap> Index(Range(hom),Image(hom));
6
gap> g.2;Image(hom,g.2);
(1,2,3,8)(4,5,6,7)  //  (1,2,3,4)
```

An ordinary GAP function that describes how a point is mapped under a group element. Special treatment in the library for certain pre-defined actions, such as `OnRight`, `OnPoints` and `OnSets`.

G acts on

Operation `ActionHomomorphism`.

Arguments: Group, Domain, [generators, actors,] action function [,"surjective"].

Points arranged same way as Ω.

```
gap> g:=TransitiveGroup(8,20);;
gap> b:=Blocks(g,MovedPoints(g));
[ [ 1, 5 ], [ 2, 6 ], [ 3, 7 ], [ 4, 8 ] ]
gap> hom:=ActionHomomorphism(g,b,OnSets);
<action homomorphism>
gap> Index(Range(hom),Image(hom));
6
gap> g.2;Image(hom,g.2);
(1,2,3,8)(4,5,6,7)  //  (1,2,3,4)
```

Group Actions

G acts on Ω, if $|\Omega|=n$, this gives $\varphi: G \to S_n$.

Operation ActionHomomorphism.

Arguments: Group, Domain, [generators, actors,] action function [,"surjective"].

otherwise into S_n

Points arranged same way as Ω.

```
gap> g:=TransitiveGroup(8,20);;
gap> b:=Blocks(g,MovedPoints(g));
[ [ 1, 5 ], [ 2, 6 ], [ 3, 7 ], [ 4, 8 ] ]
gap> hom:=ActionHomomorphism(g,b,OnSets);
<action homomorphism>
gap> Index(Range(hom),Image(hom));
6
gap> g.2;Image(hom,g.2);
(1,2,3,8)(4,5,6,7)   //   (1,2,3,4)
```

By Generators And Images

In same way as linear transformations yield a matrix:

Give Source, Range, Source generators, and their images. By default will check whether its is homom.

Needs to decompose into generators. Easy if source consists of words and generators are single letters.

```
gap> map:=GroupHomomorphismByImages(g,g,
> GeneratorsOfGroup(g),
> List(GeneratorsOfGroup(g),Inverse) #sheer dumb luck
gap> map in AutomorphismGroup(g);
true
gap> GroupHomomorphismByImages(g,g,GeneratorsOfGroup(g),
> Reversed(GeneratorsOfGroup(g))); #not a homom.
fail
```

By Generators And Images

In same way as linear transformations yield a matrix:

Give Source, Range, Source generators, and their images. By default will check whether its is homom.

Needs to decompose into generators. Easy if source consists of w ds and generators are single letters.

Otherwise variant of membership test: Divide off group elements and multiply together corresponding images.

Convenient Concept: Use Source × Range and do membership test using Source-part only.

By Generators And Images

In same way as linear transformations yield a matrix:

Give Source, Range, Source generators, and their images. By default will check whether its is homom.

Needs to decompose into generators. Easy if source consists of words and generators are single letters.

```
gap> map:=GroupHomomorphismByImages(g,g,
> GeneratorsOfGroup(g),
> List(GeneratorsOfGroup(g),Inverse) #sheer dumb luck
gap> map in AutomorphismGroup(g);
true
gap> GroupHomomorphismByImages(g,g,GeneratorsOfGroup(g),
> Reversed(GeneratorsOfGroup(g))); #not a homom.
fail
```

Interpret As Generators/Images

For a homomorphism, `MappingGeneratorsImages` returns generators of the source and a corresponding list of images. (For `GHBI` this is the stored data.)

`AsGroupGeneralMappingByImages` produces an equal `GHBI`. This can be useful if the action is expensive.

```
gap> hom; #action homomorphism from two slides ago
<action homomorphism>
gap> MappingGeneratorsImages(hom);
[ [ (2,6)(3,7), (1,2,3,8)(4,5,6,7) ], [ (), (1,2,3,4) ] ]
gap> AsGroupGeneralMappingByImages(hom);
[ (2,6)(3,7), (1,2,3,8)(4,5,6,7) ] -> [ (), (1,2,3,4) ]
```

Trust, But Verify

GroupHomomorphismByImages will test that the elements are in the groups we claim and that the map is a homomorphism. These tests can be expensive (or infeasible).

Use GroupHomomorphismByImagesNC to avoid some tests, or even use GroupGeneralMappingByImagesNC

```
gap> g:=PGL(7,3);;NrMovedPoints(g);
1093
gap> GroupHomomorphismByImages(g,Group(()), gens,
> List(gens,x->()));;time;
9801
gap> GroupHomomorphismByImagesNC(g,Group(()), gens,
> List(gens,x->()));;time;
1
```

By Recipe

In some cases one can describe by a (GAP-)function what the homomorphism is supposed to do.

Gets around factoring in generators. Useful for Composition Tree etc.

Also can give function for pre-image or inverse.

No test on being a homomorphism is done.

```
gap> g:=Group((1,2,3,4)(5,6,7,8),(1,2)(6,7));;
gap> fun:=x->RestrictedPerm(x,[1..4]);;
gap> hom:=GroupHomomorphismByFunction(g,
> SymmetricGroup(4),fun);
MappingByFunction( Group([ (1,2,3,4)(5,6,7,8), (1,2)
(6,7) ]), Sym( [ 1 .. 4 ] ), function( x ) ... end )
gap> hom=ActionHomomorphism(g,[1..4]);
true
```

Homomorphisms As Output

Some operations return homomorphism, rather than just an object which would be source or range. GAP might choose the source or range, even what kind of groups it is.

- Conversion `IsomorphismPermGroup`, (caveat: potentially large degree), `IsomorphismPcGroup`, `IsomorphismSpecialPcGroup`, `IsomorphismFpGroup`,...
- Factor groups: `NaturalHomomorphismByNormalSubgroup`
- `EpimorphismFromFreeGroup` for `Factorization` into generators.
- Make nicer: `SmallerDegreePermutationRepresentation`, `IsomorphismSimplifiedFpGroup`.

Homomorphisms As Output

Some ope ther than just an object which wou t choose the source or range, eve

Represented internally in a better way - often calculations are significantly faster.

- Conversion IsomorphismPermGroup, (caveat: potentially large degree), IsomorphismPcGroup, IsomorphismSpecialPcGroup, IsomorphismFpGroup,...
- Factor groups: NaturalHomomorphismByNormalSubgroup
- EpimorphismFromFreeGroup for Factorization into generators.
- Make nicer: SmallerDegreePermutationRepresentation, IsomorphismSimplifiedFpGroup.

Homomorphisms As Output

Some operations return homomorphism, rather than just an object which would be source or range. GAP might choose the source or range, even what kind of groups it is.

- Conversion `IsomorphismPermGroup`, (caveat: potentially large degree), `IsomorphismPcGroup`, `IsomorphismSpecialPcGroup`, `IsomorphismFpGroup`,...
- Factor groups: `NaturalHomomorphismByNormalSubgroup`
- `EpimorphismFromFreeGroup` for `Factorization` into generators.
- Make nicer: `SmallerDegreePermutationRepresentation`, `IsomorphismSimplifiedFpGroup`.

Composition

Homomorphisms can be composed by product (left-to-right) or CompositionMapping, and may have an InverseGeneralMapping. GAP will try to represent the new homomorphism in a clever way, that might be a formal composition.

```
gap> g:=AtlasSubgroup("Fi22",11);;
gap> h:=AtlasSubgroup("Fi22",IsMatrixGroup,11);;
gap> hom:=GroupHomomorphismByImages(g,h,
> GeneratorsOfGroup(g),GeneratorsOfGroup(h));
[ (1,1450[…] ] -> [ <an immutable 78x78 matrix […]]
gap> hom:=GroupHomomorphismByImages(h,g,
> GeneratorsOfGroup(h), GeneratorsOfGroup(g));
CompositionMapping( [ (1,2)(3,6)) […]] ->
[ (1,1450)(2 […] ], <action isomorphism> ) )
```

Composition

Inverse only works for group automorphisms.

Homomorphisms can be composed by product (left-to-right) or CompositionMapping, and may have an InverseGeneralMapping. GAP will try to represent the new homomorphism in a clever way, that might be a formal composition.

```
gap> g:=AtlasSubgroup("Fi22",11);;
gap> h:=AtlasSubgroup("Fi22",IsMatrixGroup,11);;
gap> hom:=GroupHomomorphismByImages(g,h,
> GeneratorsOfGroup(g),GeneratorsOfGroup(h));
[ (1,1450[…] ] -> [ <an immutable 78x78 matrix […]]
gap> hom:=GroupHomomorphismByImages(h,g,
> GeneratorsOfGroup(h), GeneratorsOfGroup(g));
CompositionMapping( [ (1,2)(3,6)) […]] ->
[ (1,1450)(2 […] ], <action isomorphism> ) )
```

Composition

Homomorphisms can be composed by product (left-to-right) or CompositionMapping, and may have an InverseGeneralMapping. GAP will try to represent the new homomorphism in a clever way, that might be a formal composition.

```
gap> g:=AtlasSubgroup("Fi22",11);;
gap> h:=AtlasSubgroup("Fi22",IsMatrixGroup,11);;
gap> hom:=GroupHomomorphismByImages(g,h,
> GeneratorsOfGroup(g),GeneratorsOfGroup(h));
[ (1,1450[…] ] -> [ <an immutable 78x78 matrix […]]
gap> hom:=GroupHomomorphismByImages(h,g,
> GeneratorsOfGroup(h), GeneratorsOfGroup(g));
CompositionMapping( [ (1,2)(3,6)) […]] ->
[ (1,1450)(2 […] ], <action isomorphism> ) )
```

```
gap> iso:=IsomorphismPcGroup(g);;r:=Range(iso);;
gap> emb:=GroupHomomorphismByImages(s4,r,
> [(1,2,4,3),(1,4,3)],[r.1,r.2]);; #happens to work
gap> emb*InverseGeneralMapping(hom);
[ (1,2,4,3), (1,4,3) ] -> [ (3,3141)(4, […]
gap> chi:=First(Irr(g),x->x[1]=2);;
gap> rep:=IrreducibleRepresentationsDixon(g,chi);
gap> InverseGeneralMapping(hom)*rep;
[ f1,f2,f3,f4,f5,f6,f7, […] ] ->
[ [[0,E(3)],[E(3)^2,0]], [[E(3),0],[0,E(3)^2]], […] ]
gap> conj:=InverseGeneralMapping(iso)
> *InnerAutomorphism(g,g.1)*iso;
[ f1, f2, f3, f4, f5, f6, f7, […] ] ->
[ f1*f2^2*f5*f7*f10* […] ]
gap> IsInnerAutomorphism(conj);
true
gap> conj;
^f1*f2*f5^2*f6*f7*f11^2*f12*f15*f16*f17^2
```

Really! No cheat.

```
gap> iso:=IsomorphismPcGroup(g);;r:=Range(iso);;
gap> emb:=GroupHomomorphismByImages(s,r,
> [(1,2,4,3),(1,4,3)],[r.1,r.2]);; #happens to work
gap> emb*InverseGeneralMapping(hom);
[ (1,2,4,3), (1,4,3) ] -> [ (3,3141)(4, […]
gap> chi:=First(Irr(g),x->x[1]=2);;
gap> rep:=IrreducibleRepresentationsDixon(g,chi);
gap> InverseGeneralMapping(hom)*rep;
[ f1,f2,f3,f4,f5,f6,f7, […] ] ->
[ [[0,E(3)],[E(3)^2,0]], [[E(3),0],[0,E(3)^2]], […] ]
gap> conj:=InverseGeneralMapping(iso)
> *InnerAutomorphism(g,g.1)*iso;
[ f1, f2, f3, f4, f5, f6, f7, […] ] ->
[ f1*f2^2*f5*f7*f10* […] ]
gap> IsInnerAutomorphism(conj);
true
gap> conj;
^f1*f2*f5^2*f6*f7*f11^2*f12*f15*f16*f17^2
```

```
gap> iso:=IsomorphismPcGroup(g);;r:=Range(iso);;
gap> emb:=GroupHomomorphismByImages(s4,r,
> [(1,2,4,3),(1,4,3)],[r.1,r.2]);; #happens to work
gap> emb*InverseGeneralMapping(hom);
[ (1,2,4,3), (1,4,3) ] -> [ (3,3141)(4, […]
gap> chi:=First(Irr(g),x->x[1]=2);;
gap> rep:=IrreducibleRepresentationsDixon(g,chi);
gap> InverseGeneralMapping(hom)*rep;
[ f1,f2,f3,f4,f5,f6,f7, […] ] ->
[ [[0,E(3)],[E(3)^2,0]], [[E(3),0],[0,E(3)^2]], […] ]
gap> conj:=InverseGeneralMapping(iso)
> *InnerAutomorphism(g,g.1)*iso;
[ f1, f2, f3, f4, f5, f6, f7, […] ] ->
[ f1*f2^2*f5*f7*f10* […] ]
gap> IsInnerAutomorphism(conj);
true
gap> conj;
^f1*f2*f5^2*f6*f7*f11^2*f12*f15*f16*f17^2
```

Automorphism Groups

Bijective homomorphisms from a group to itself can form a group of automorphisms, one can also calculate the full `AutomorphismGroup`.

When creating a group from homomorphisms, use `SetIsGroupOfAutomorphismsFiniteGroup` to ensure fast methods.

```
gap> au:=AutomorphismGroup(SmallGroup(128,1234));;
<group of size 32768 with 15 generators>
gap> sub:=Subgroup(au,GeneratorsOfGroup(au){[1..14]});;
gap> Size(sub); #immediate
32768
gap> sub:=Group(GeneratorsOfGroup(au){[1..14]});;
gap> Size(sub); #notable delay
32768
```

Automorphism Groups

Call `SetIsGroupOfAutomorphismsFiniteGroup(sub,true);` before calculation to get faster processing.

calculate the full AutomorphismGroup.

When creating a group from homomorphisms, use SetIsGroupOfAutomorphismsFiniteGroup to ensure fast methods.

```
gap> au:=AutomorphismGroup(SmallGroup(128,1234));;
<group of size 32768 with 15 generators>
gap> sub:=Subgroup(au,GeneratorsOfGroup(au){[1..14]});;
gap> Size(sub); #immediate
32768
gap> sub:=Group(GeneratorsOfGroup(au){[1..14]});;
gap> Size(sub); #notable delay
32768
```

Practical Part 1

1. Form some homomorphisms to quotients and compare their images.
2. A homomorphism given by a rule.
3. Create a presentation and transfer to another free group.
4. Stabilizer of a set of sets by change of perm.rep.
5. Automorphisms induced by normalizer in S_n
6. Outer automorphism of S_6
7. Smallest faithful permutation degree.

Searching For Homomorphisms

Operations that search for possible homom.^{s} by running, up to conjugacy, through potential image tuples for a generating sequence:

- `IsomorphismGroups`: Isomorphism or fail, `AutomorphismGroup`. (Non-solvable bit)
- `GQuotients`: Epimorphisms list, different kernel.
- `IsomorphicSubgroups`: Monom., diff. images.
- `AllHomomorphismClasses`:Same kernels,Images

Cost depends on search space, # of generators

Thus cost is exponential in number of generators. Luckily all simple groups can be generated by two elements.

ms

Operations that search for possible homom.s by running, up to conjugacy, through potential image tuples for a generating sequence:

- IsomorphismGroups: Isomorphism or fail, AutomorphismGroup. (Non-solvable bit)
- GQuotients: Epimorphisms list, different kernel.
- IsomorphicSubgroups: Monom., diff. images.
- AllHomomorphismClasses:Same kernels,Images

Cost depends on search space, # of generators

Searching F

In solvable case: Lift through elementary abelian layers using cohomology.

Operations that search for possible ho nom.s by running, up to conjugacy, through pot ntial image tuples for a generating sequence:

- IsomorphismGroups: Isomorphism or fail, AutomorphismGroup. (Non-solvable bit)
- GQuotients: Epimorphisms list, different kernel.
- IsomorphicSubgroups: Monom., diff. images.
- AllHomomorphismClasses:Same kernels,Images

Cost depends on search space, # of generators

```
gap> f:=FreeGroup(2);;
gap> GQuotients(f,SymmetricGroup(3));
[ [f1,f2] -> [(2,3),(1,2)], [f1,f2] -> [(2,3),(1,2,3)],
  [f1,f2] -> [(1,2,3),(1,2)] ]
gap> GQuotients(f,SymmetricGroup(3) :findall:=false);
[ [ f1, f2 ] -> [ (2,3), (1,2) ] ]

gap> IsomorphicSubgroups(Alternat.Gp.(6),Alternat.Gp.(5));
[ [(1,5,3,2,4),(1,5)(2,4)] -> [(2,3,6,5,4),(1,2)(3,4)],
  [(1,5,3,2,4),(1,5)(2,4)] -> [(1,2,3,4,5),(1,2)(3,4)] ]

gap> AllHomomorphismClasses(DihedralGroup(6),
> DihedralGroup(12));
[ [f1,f2] -> [<identity>,<identity>],
  [f1,f2] -> [f1,<identity>],  [f1,f2] -> [f1,f3],
  [f1,f2] -> [f1*f2,<identity>],  [f1,f2] -> [f1*f2,f3],
  [f1,f2] -> [f2*f3,<identity>] ]
gap> Length(AllHomomorphismClasses(DihedralGroup(12),
> DihedralGroup(6)));
5
```

Further Quotient Algorithms

Quotient algorithms also are a main tool in finding certain quotients of finitely presented groups. They work by describing a "*generic image*" and then use linear algebra to impose constraints: `MaximalAbelianQuotient`(also for other groups), `EpimorphismPGroup`, `EpimorphismSolvableGroup`.

```
gap> f:=FibonacciGroup(8);;
gap> MaximalAbelianQuotient(f); #image 3 x 3 x 5
[ f1,f2,f3,f4,f5,f6,f7,f8 ] -> [ f1*f2*f3^4,[…]
gap> Size(Image(EpimorphismPGroup(f,3,5)));
6561
gap> Size(Image(EpimorphismSolvableQuotient(f,2^5*3^5*5)));
38880
```

Pre-Image Subgroups

Subgroups of finitely presented groups are represented as pre-image of a group under a homomorphism (e.g. point stabilizer in action on cosets). Represent huge index without a coset table.

Calculations with subgroups work with the direct product of the homomorphisms.

```
gap> f:=FreeGroup("a","b");;
gap> rels:=ParseRelators(f,"a2,b3,(abababaBaBabaBaB)2");
gap> g:=f/rels;;
gap> l:=Intersection(LowIndexSubgroups(g,20));;
gap> Index(g,l);
3317760
gap> def:=DefiningQuotientHomomorphism(l);
[ a, b ] -> [ (3,4)(5,6)(7,8)(10,11) […]
```

```
gap> f:=FreeGroup("x","y","z");;
gap> g:=f/ParseRelators(f,
   "x,y3,z2,(xyxyxyxYxYxyxYxY)2,(yz)2,(xz)2");;
gap> l:=LowIndexSubgroupsFpGroup(g,20);;Length(l);
133
gap> k:=Intersection(l);; #index 6635520, normal
gap> q:=DefiningQuotientHomomorphism(k);;p:=Image(q);;
gap> m:=ShallowCopy(LowLayerSubgroups(p,2));;
gap> SortBy(m,x->Index(p,x));
gap> sub:=First(m,x->AbelianInvariants(x)
>         <>AbelianInvariants(PreImage(q,x)));
gap> Index(p,sub);
16
gap> AbelianInvariants(sub);
[ 2, 2 ]
gap> AbelianInvariants(PreImage(q,sub));
[ 2, 4 ]
gap> new:=LargerQuotientBySubgroupAbelianization(q,sub);;
gap> k2:=Intersection(k,new);;
gap> q2:=DefiningQuotientHomomorphism(k);;p2:=Image(q2);;
gap> Size(p2); #larger by factor 2
46006272
```

Nice Monomorphisms

For some classes of groups: (e.g. matrix (still...), automorphisms, ...) GAP performs all calculations though image under a `NiceMonomophism` (that typically is given by an action or function).

Trigger: `IsHandledByNiceMonomorphism`.

Can use for own objects, the default method for NiceMonomorphism is the right regular action.

```
gap> g:=GL(4,7);; IsHandledByNiceMonomorphism(g);
true
gap> SylowSubgroup(g,2);
<matrix group of size 2048 with 3 generators>
gap> NiceMonomorphism(g); #on 7^4-1 vectors
<action isomorphism>
gap> NrMovedPoints(Range(NiceMonomorphism(g)));
2400
```

Group Products

When forming new groups as products, GAP might use homomorphisms to specify the product. It will represent the product in a "suitable" way, and will provide homomorphisms to allow for decomposition as product: Embedding(prd,nr) and Projection(prd,nr).

```
gap> g:=AlternatingGroup(5);h:=SmallGroup(24,12);
Alt( [ 1 .. 5 ] )
<pc group of size 24 with 4 generators>
gap> dir:=DirectProduct(g,h);;
gap> MappingGeneratorsImages(Embedding(dir,1));
[ [ (1,2,3,4,5), (3,4,5) ],
  [ DirectProductElement( [ (1,2,3,4,5), <id> ] ),
    DirectProductElement( [ (3,4,5), <id> of ... ] ) ] ]
```

```
gap> dir:=DirectProduct(g,Group((1,2,3,4),(1,2)));
Group([ (1,2,3,4,5), (3,4,5), (6,7,8,9), (6,7) ])
gap> a:=Image(Embedding(dir,1),(1,2,3))
>          *Image(Embedding(dir,2),(1,2)(3,4));
(1,2,3)(6,7)(8,9)
gap> Image(Projection(dir,1),a);
(1,2,3)
gap> Image(Projection(dir,2),a);
(1,2)(3,4)
```

Semidirect Products

A SemidirectProduct $C \ltimes N$ is specified by constructing a homomorphism from C to a group of automorphisms of N.

Again, Embedding maps into the product, there is only Projection(prd).

The same holds for WreathProduct.

```
gap> C:=AbelianGroup([2]);;N:=AbelianGroup([3]);;
gap> aut:=GroupHomomorphismByImages(N,N,[N.1],[N.1^-1]);
[ f1 ] -> [ f1^2 ]
gap> map:=GroupHom[…]ByImages(C,Group(aut),[C.1],[aut]);;
gap> sdp:=SemidirectProduct(C,map,N);
<pc group of size 6 with 2 generators>
gap> IsomorphismGroups(sdp,SymmetricGroup(3));
[ f1, f2 ] -> [ (2,3), (1,2,3) ]
```

Practical Part 2

8. Construct an (external) semidirect product.
9. Decompose a group formally as (internal) semidirect product.
10. Quotients of a particular isomorphism type - generating direct products.
11. Forming larger and larger quotients of finitely presented groups.
12. Constructing and inducing representations.

Template

```
gap> LoadPackage(“matgrp”); #use recog
    […]
gap> g:=AtlasSubgroup("Co1",IsMatrixGroup,3); #2^11.M_24
<matrix group of size 501397585920 with 2 generators>
gap> FittingFreeLiftSetup(g); #2 seconds
rec( depths:=[ 1, 12 ],
  factorhom:=[ [<immutable 24x24 matrix GF2>, […] ]] ->
    [ (1,11[…]],
```

Too Clever By Half

Use different tool

Permgroup acting linearly

Niceo (transfer conj classes?)

IsomorphismSpecialPcGroup