# Computing in Groups and Expanded Groups

Version 1

**Stephan Zweckinger**

**Stephan Zweckinger** Email: stephan.zweckinger@gmx.de

## Abstract

This documentation describes a function for finding the degree of a function, a package for handling expanded groups and functions for solving the Subpower Intersection Problem and the Subpower Membership Problem for groups. The package and all functions were developed in the frame of the master thesis "Computing in Direct Powers of Expanded Groups" by Stephan Zweckinger. More informations about the implementation of the described functions and the theory behind can be found in this master thesis.

## Copyright



## Acknowledgements

Thanks to Max Neunhöffer (St. Andrews) for help with designing the datastructure for expanded groups in GAP and to Peter Mayr (Linz) for supporting the whole development in the frame of my master thesis.

# Contents

# Chapter 1

# Expanded Groups

## 1.1 Functions with Finite Degree

Let $\mathbf{G} = \langle G, \cdot, ^{-1}, 1 \rangle$ be a group and let $f$ be a function from $G$ to $G$. We define the degree of $f$ as the smallest $d \in \mathbb{N}_0$ such that for all $x_1, \ldots, x_n \in G$:

$$f(\prod_{i=1}^{n} x_i) \text{ is a product of functions } f(\prod_{i \in S} x_i) \text{ or } f(\prod_{i \in S} x_i)^{-1}$$

for $S \subseteq \{1, \ldots, n\}, |S| \leq d$. If no such $d$ exists, we say that the degree of the function is infinite.

If $\mathbf{G}$ is an abelian group, the degree of $f$ is the smallest $d \in \mathbb{N}_0$ such that

$$\prod_{S \subseteq \{1,\ldots,d+1\}} (f(\prod_{i \in S} x_i))^{(-1)^{|S|}} = 1$$

for all $x_1, \ldots, x_{d+1} \in G$.

### 1.1.1 FindDegree

▷ `FindDegree(G, f, arity, max)` (function)

This function returns the degree of the function `f` with arity `arity` on the abelian group `G`, if the degree is less or equal `max`. Otherwise it returns `false`. If `G` is not abelian it returns an error message.

Example

```
gap> f1 := function(x) return x[1]*x[2]; end;
function( x ) ... end
gap> f2 := function(x) return Identity(x); end;
function( x ) ... end
gap> FindDegree(CyclicGroup(5), f1, 2, 10);
1
gap> FindDegree(CyclicGroup(5), f2, 1, 10);
0
gap> IsAbelian(SymmetricGroup(5));
false
gap> FindDegree(SymmetricGroup(5), f1, 2, 3);
Error, Group is not abelian called from
<function "FindDegree">( <arguments> )
 called from read-eval loop at line 4 of *stdin*
```

```
you can 'quit;' to quit to outer loop, or
you can 'return;' to continue
```

## 1.2 A Package for Expanded Groups

Let $\mathbf{G} = \langle G, \cdot, ^{-1}, 1 \rangle$ be a group. Let $f_1, \ldots, f_l$ be additional operations of arbitrary finite arities on $G$. Then the algebra $\mathbf{E} = \langle G, \cdot, ^{-1}, 1, f_1, \ldots, f_l \rangle$ is called an expanded group. The package ExpGroup provides functions for computing with expanded groups and direct powers of expanded groups. For using the package all unary functions must be represented as functions taking one element and all $k$-ary functions, $k > 1$, as functions taking a list with $k$ elements. For example, the ternary operation $f(x,y,z) = x \cdot y \cdot z$ is represented as follows:

Example

```
gap>  f := function(x) return x[1]*x[2]*x[3]; end;
function( x ) ... end
```

To avoid problems with the abstract representation of groups, all groups must be used as permutation groups. Therefore the package SONATA, which provides a command `AsPermGroup` for converting a group to a permutation group, must be loaded.

Example

```
gap> LoadPackage("sonata");;
gap> AsPermGroup(DihedralGroup(8));
Group([ (1,2)(3,8)(4,6)(5,7), (1,3,4,7)(2,5,6,8), (1,4)(2,6)(3,7)(5,8) ])
```

The package consists of following functions:

### 1.2.1 ExpandedGroups

▷ `ExpandedGroups(`*g, arities, ops*`)` (function)

This function generates a component object of type `ExpandedGroup` which represents an expanded group and consists of the group *g* and additional operations contained in the list *ops* with corresponding arities contained in the list *arities*. The command `!.` gives access to the single components.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup(G, [1,1,2], [f1, f2, f3]);
<expanded group Sym( [ 1 .. 4 ] ) with 3 operations>
gap> e!.g;
Sym( [ 1 .. 4 ] )
gap> e!. arities;
```

```
[ 1, 1, 2 ]
gap> e!.ops;
[ function( x ) ... end, function( x ) ... end, function( x ) ... end ]
```

### 1.2.2 IsExpandedGroup

▷ IsExpandedGroup(*e*) (function)

This function returns true if *e* is of type ExpandedGroup and false otherwise.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup(G, [1,1,2], [f1, f2, f3]);
<expanded group Sym( [ 1 .. 4 ] ) with 3 operations>
gap> IsExpandedGroup(e);
true
gap> IsExpandedGroup(SymmetricGroup(5));
false
```

### 1.2.3 DirectPower

▷ DirectPower(*e*, *n*) (function)

This function returns a component object which is both of type ExpandedGroup and of type ExpandedGroupDirectPower and represents the *n*-th direct power of the expanded group *e*. It consists of the record entries g storing the direct power group, ops and arities storing the componentwise defined operations and their arities and *base*, *n* storing the base expanded group and the exponent.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup(G, [1,1,2], [f1, f2, f3]);
<expanded group Sym( [ 1 .. 4 ] ) with 3 operations>
gap> d := DirectPower(e, 3);
```

```
<expanded group <group of size 13824 with 6 generators> with 3 operations>
gap> d!.g;
Group([ (1,2,3,4), (1,2), (5,6,7,8), (5,6), (9,10,11,12), (9,10) ])
gap> d!.base;
<expanded group Sym( [ 1 .. 4 ] ) with 3 operations>
gap> d!.n;
3
gap> d!.ops;
[ function( x ) ... end, function( x ) ... end, function( x ) ... end ]
gap> d!.arities;
[ 1, 1, 2 ]
```

### 1.2.4 IsExpandedGroupDirectPower

▷ `IsExpandedGroupDirectPower(e)` (function)

This function returns `true` if *e* is of type `ExpandedGroupDirectPower` and `false` otherwise.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup(G, [1,1,2], [f1, f2, f3]);
<expanded group Sym( [ 1 .. 4 ] ) with 3 operations>
gap> d := DirectPower(e, 3);
<expanded group <group of size 13824 with 6 generators> with 3 operations>
gap> IsExpandedGroupDirectPower(d);
true
gap> IsExpandedGroupDirectPower(e);
false
```

### 1.2.5 Embedding

▷ `Embedding(d, n)` (function)

This function returns the embedding in the *n*-th component of the direct power of an expanded group *d*.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
```

```
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup(G, [1,1,2], [f1, f2, f3]);
<expanded group Sym( [ 1 .. 4 ] ) with 3 operations>
gap> d := DirectPower(e, 3);
<expanded group <group of size 13824 with 6 generators> with 3 operations>
gap> Image(Embedding(d, 3), (1,3));
(9,11)
```

### 1.2.6 Projection

▷ `Projection(`*d*`, `*n*`)` (function)

This function returns the projection of the direct power of an expanded group *d* onto its *n*-th component.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup(G, [1,1,2], [f1, f2, f3]);
<expanded group Sym( [ 1 .. 4 ] ) with 3 operations>
gap> d := DirectPower(e, 3);
<expanded group <group of size 13824 with 6 generators> with 3 operations>
gap> Image(Projection(d, 3), (9,11));
(1,3)
```

### 1.2.7 ExpandedSubgroup

▷ `ExpandedSubgroup(`*e*`, `*gen*`)` (function)

This function returns a new expanded group, which is a subalgebra of *e* and generated by the elements of the list *gen*. Therefore the new expanded subgroup has the same operations and arities as *e*. Note that there exists no record entry g in the newly generated object, but an entry `generators` which is equal to the input list *gen*. If *e* is of type `ExpandedGroupDirectPower`, then the new expanded subgroup contains a record `parent` which stores *e*. Note that direct powers which are constructed by `DirectPower` do not have a record `parent`.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
```

```
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup (G, [1,1],[f1, f2]);
<expanded group Sym( [ 1 .. 4 ] ) with 2 operations>
gap> s := ExpandedSubgroup(e, [(1,2,3)]);
<expanded group with 3 operations>
gap> IsBound(s!.g);
false
gap> s!.arities;
[ 1, 1, 2 ]
gap> s!.ops;
[ function( x ) ... end, function( x ) ... end, function( x ) ... end ]
gap> s!.generators;
[ (1,2,3) ]
gap> d := DirectPower(e, 3);
<expanded group <group of size 13824 with 6 generators> with 3 operations>
gap> sd := ExpandedSubgroup(d, [(1,2,3)(5,6)]);
<expanded group with 3 operations>
gap> sd!.parent;
Group([ (1,2,3,4), (1,2), (5,6,7,8), (5,6), (9,10,11,12), (9,10) ])
```

### 1.2.8 GroupOfExpandedGroup

▷ `GroupOfExpandedGroup(`*e[, deg]*`)` (function)

This function returns the group contained in the expanded group *e*. If `e!.g` is not set, `e!.g` is computed. A faster algorithm is used if all expanded group operations have finite degree and those degrees are given by an additional argument list *deg*.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup (G, [1,1],[f1, f2]);
<expanded group Sym( [ 1 .. 4 ] ) with 2 operations>
gap> s := ExpandedSubgroup(e, [(1,2,3)]);
<expanded group with 3 operations>
gap> IsBound(s!.g);
false
gap> GroupOfExpandedGroup(s);
Group([ (1,2,3), (1,3) ])
gap> IsBound(s!.g);
```

```
true
```

### 1.2.9 \in

▷ `\in(x, e)` (function)

This function returns `true` if $x$ is an element of $e$ and `false` otherwise.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup (G, [1,1],[f1, f2]);
<expanded group Sym( [ 1 .. 4 ] ) with 2 operations>
gap> (1,2,3) in e;
true
gap> (1,5,3) in e;
false
```

### 1.2.10 Size

▷ `Size(e)` (function)

This function returns the order of the expanded group $e$, that is, the size of `e!.g`.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup (G, [1,1],[f1, f2]);
<expanded group Sym( [ 1 .. 4 ] ) with 2 operations>
gap> s := ExpandedSubgroup(e, [(1,2,3)]);
<expanded group with 3 operations>
gap> Size(e);
24
gap> Size(s);
6
```

### 1.2.11 Identity

▷ `Identity(e)` (function)

This function returns the identity element of the expanded group `e`.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup (G, [1,1],[f1, f2]);
<expanded group Sym( [ 1 .. 4 ] ) with 2 operations>
gap> Identity(e);
()
```

### 1.2.12 Random

▷ `Random(e)` (function)

This function returns a randomly chosen element of the universe of the expanded group `e`.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup (G, [1,1],[f1, f2]);
<expanded group Sym( [ 1 .. 4 ] ) with 2 operations>
gap> Random(e);
(2,4,3)
```

### 1.2.13 IsGroupHomomorphismOfExpandedGroup

▷ `IsGroupHomomorphismOfExpandedGroup(e, h)` (function)

This function returns `true` if `h` is a group homomorphism of the expanded group `e`, i.e. if `h` is a group homomorphism of `e!.g` and the homomorphism property holds also for the additional operations of the expanded group. Otherwise it returns `false`.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup (G, [1,1],[f1, f2]);
<expanded group Sym( [ 1 .. 4 ] ) with 2 operations>
gap> h := function(x) return (1,2,3)^(-1)*x*(1,2,3); end;
function( x ) ... end
gap> h := MappingByFunction(e!.g, e!.g, h);
MappingByFunction( Sym( [ 1 .. 4 ] ), Sym( [ 1 .. 4 ] ), function( x ) ... en\
d )
gap> IsGroupHomomorphismOfExpandedGroup(e, h);
false
```

### 1.2.14 IsHomomorphismOfExpandedGroup

▷ `IsHomomorphismOfExpandedGroup(`*e*, *h*`)` (function)

This function returns `true` if the homomorphism property of *h* holds for the additional operations of the expanded group *e*. Otherwise it returns `false`.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup (G, [1,1],[f1, f2]);
<expanded group Sym( [ 1 .. 4 ] ) with 2 operations>
gap> h := function(x) return (1,2,3)^(-1)*x*(1,2,3); end;
function( x ) ... end
gap> h := MappingByFunction(e!.g, e!.g, h);
MappingByFunction( Sym( [ 1 .. 4 ] ), Sym( [ 1 .. 4 ] ), function( x ) ... en\
d )
gap> IsHomomorphismOfExpandedGroup(e, h);
false
```

### 1.2.15 IsReductOfExpandedSubgroup

▷ IsReductOfExpandedSubgroup(e, s) (function)

This function returns true if the group s is a reduct of the expanded group e, that is, if s is closed under all operations of e. Otherwise it returns false.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f1 := function(x) return (1,2)^(-1)*x*(1,2); end;
function( x ) ... end
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup (G, [1,1],[f1, f2]);
<expanded group Sym( [ 1 .. 4 ] ) with 2 operations>
gap> IsReductOfExpandedGroup(e, Group((1,2,3)) );
false
```

### 1.2.16 IsIdealOfExpandedGroup

▷ IsIdealOfExpandedGroup(e, s) (function)

This function returns true if the group s forms an ideal of the expanded group e. Otherwise it returns false. A set $I$ is an ideal of an expanded group $\mathbf{E} = \langle G, \cdot, ^{-1}, 1, f_1, \ldots, f_l \rangle$ if $I$ is a normal subgroup of $\langle G, \cdot, ^{-1}, 1 \rangle$ and $\forall f \in \{f_1, \ldots, f_l\} \forall x \in A^{k_i}, i \in I^{k_i} : f(x \cdot i) \cdot (f(x))^{-1} \in I$ where $k_i$ is the arity of $f_i$. In GAP ideals are given as groups with universe $I$ instead of a set $I$.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup(G, [1], [f2]);
<expanded group Sym( [ 1 .. 4 ] ) with 1 operations>
gap> NormalSubgroups(G);
[ Sym( [ 1 .. 4 ] ), Group([ (2,4,3), (1,4)(2,3), (1,3)(2,4) ]),
  Group([ (1,4)(2,3), (1,3)(2,4) ]), Group(()) ]
gap> IsIdealOfExpandedGroup(e, Group([ (2,4,3), (1,4)(2,3), (1,3)(2,4) ]));
false;
```

### 1.2.17 IdealsOfExpandedGroup

▷ IdealsOfExpandedGroup(e) (function)

This function returns a list containing all ideals of the expanded group e. A set $I$ is an ideal of an expanded group $\mathbf{E} = \langle G, \cdot, ^{-1}, 1, f_1, \ldots, f_l \rangle$ if $I$ is a normal subgroup of $\langle G, \cdot, ^{-1}, 1 \rangle$ and $\forall f \in$

$\{f_1,\ldots,f_l\}\forall x\in A^{k_i}, i\in I^{k_i} : f(x\cdot i)\cdot(f(x))^{-1}\in I$ where $k_i$ is the arity of $f_i$. In GAP ideals are given as groups with universe $I$ instead of a set $I$.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup(G, [1], [f2]);
<expanded group Sym( [ 1 .. 4 ] ) with 1 operations>
gap> IdealsOfExpandedGroup(e);
[ Sym( [ 1 .. 4 ] ), Group(()) ]
gap> NormalSubgroups(G);
[ Sym( [ 1 .. 4 ] ), Group([ (2,4,3), (1,4)(2,3), (1,3)(2,4) ]),
  Group([ (1,4)(2,3), (1,3)(2,4) ]), Group(()) ]
```

### 1.2.18 IdealByGenerators

▷ `IdealByGenerators(e, gen)` (function)

This function returns the ideal of the expanded group `e` that is generated by the elements contained in the list `gen`. A set $I$ is an ideal of an expanded group $\mathbf{E}=\langle G,\cdot,^{-1},1,f_1,\ldots,f_l\rangle$ if $I$ is a normal subgroup of $\langle G,\cdot,^{-1},1\rangle$ and $\forall f\in\{f_1,\ldots,f_l\}\forall x\in A^{k_i}, i\in I^{k_i} : f(x\cdot i)\cdot(f(x))^{-1}\in I$ where $k_i$ is the arity of $f_i$. In GAP ideals are given as groups with universe $I$ instead of a set $I$.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
gap> f2 := function(x) if x = () then return (1,2, 3)^(-1)*x*(1,2);
> else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup(G, [1], [f2]);
<expanded group Sym( [ 1 .. 4 ] ) with 1 operations>
gap> IsIdealOfExpandedGroup(e, Group([ (2,4,3), (1,4)(2,3), (1,3)(2,4) ]));
false
gap> IdealByGenerators(e, [(2,4),(1,4)]);
Group([ (2,4), (1,2,4), (1,2,3) ])
gap> Group([ (2,4), (1,2,4), (1,2,3) ]) = e!.g;
true
```

### 1.2.19 Factor

▷ `Factor(e, s)` (function)

This function returns the factor algebra `e/s` of an expanded group `e` and an ideal `s` of `e`. A set $I$ is an ideal of an expanded group $\mathbf{E}=\langle G,\cdot,^{-1},1,f_1,\ldots,f_l\rangle$ if $I$ is a normal subgroup of $\langle G,\cdot,^{-1},1\rangle$ and $\forall f\in\{f_1,\ldots,f_l\}\forall x\in A^{k_i}, i\in I^{k_i} : f(x\cdot i)\cdot(f(x))^{-1}\in I$ where $k_i$ is the arity of $f_i$. In GAP ideals are given as groups with universe $I$ instead of a set $I$.

Example

```
gap> G:= SymmetricGroup(4);
Sym( [ 1 .. 4 ] )
```

```
gap> f3 := function(x) if x[1] in AlternatingGroup(4) and x[2] in
> AlternatingGroup(4) then return x[1]*x[2]; else return (); fi; end;
function( x ) ... end
gap> e := ExpandedGroup(G, [2], [f3]);
<expanded group Sym( [ 1 .. 4 ] ) with 1 operations>
gap> Factor(e, Group([ (2,4,3), (1,4)(2,3), (1,3)(2,4) ]));
<expanded group <group of size 2 with 1 generators> with 1 operations>
```

# Chapter 2

# Strong Generators of Groups and their Applications

Note that all elements of direct powers in this chapter must be represented as a tuple of permutations.

Example

```
gap> Tuple([(1,2),(1,2,3)]);
DirectProductElement( [ (1,2), (1,2,3) ] )
```

## 2.1 Strong Generators

Let $G = A_1 \times A_2 \times \ldots \times A_n$ be a direct power of groups. For $i \in \{1,\ldots,n\}$, let $G_i = \{a \in G | a_j = 1$ for $j \leq i$ and $a_j \in A_j$ for $j > i\}$ and $T_i$ the set of coset representatives of $G_{i-1}/G_i$. The elements of $T_1,\ldots,T_n$ are the strong generators of $G$. Every element $g \in G$ can be written uniquely in the form $g = g_1 \cdot g_2 \cdot \ldots \cdot g_n$, where $g_i \in T_i$.

### 2.1.1 StrongGen

▷ `StrongGen(l)` (function)

This function takes a list `l` which contains tuples of permutations and returns the strong generators of the group generated by the elements of `l` as a list of the transversals $(T_1,\ldots,T_n)$.

Example

```
gap> StrongGen([Tuple([(1,2),(2,3,1)]), Tuple([(1,3),(2,3,1)])]);
[ [ DirectProductElement( [ (), () ] ),
      DirectProductElement( [ (1,2), (1,2,3) ] ),
      DirectProductElement( [ (1,3), (1,2,3) ] ),
      DirectProductElement( [ (1,3,2), (1,3,2) ] ),
      DirectProductElement( [ (1,2,3), (1,3,2) ] ),
      DirectProductElement( [ (2,3), () ] ) ],
  [ DirectProductElement( [ (), () ] ),
      DirectProductElement( [ (), (1,3,2) ] ),
      DirectProductElement( [ (), (1,2,3) ] ) ] ]
```

## 2.2 The Subpower Membership Problem

The Subpower Memberhip Problem is the question whether a given element is contained in a direct product given by generators.

### 2.2.1 IsRepresentable

▷ `IsRepresentable(e, strong)` (function)

This function returns `true` if `e` is an element of the direct product generated by the strong generators *strong* given as list of transversals. Otherwise it returns `[n, s]` such that adding the element `s` to the `n`-th transversal gives the strong generators of the direct product generated by *strong* and `e`.

Example

```
gap> T := StrongGen([Tuple([(1,2),(2,3,1)]), Tuple([(1,3),(2,3,1)])]);
[ [ DirectProductElement( [ (), () ] ),
      DirectProductElement( [ (1,2), (1,2,3) ] ),
      DirectProductElement( [ (1,3), (1,2,3) ] ),
      DirectProductElement( [ (1,3,2), (1,3,2) ] ),
      DirectProductElement( [ (1,2,3), (1,3,2) ] ),
      DirectProductElement( [ (2,3), () ] ) ],
  [ DirectProductElement( [ (), () ] ),
      DirectProductElement( [ (), (1,3,2) ] ),
      DirectProductElement( [ (), (1,2,3) ] ) ] ]
gap> IsRepresentable(Tuple([(1,3), (1,3,2)]), T);
true
gap> IsRepresentable(Tuple([(1,3), (1,2)]), T);
[ 2, DirectProductElement( [ (), (1,3) ] ) ]
```

## 2.3 The Subpower Intersection Problem

The Subpower Intersection Problem is the problem of finding the universe of the intersection of two groups given by their generators.

### 2.3.1 GenIntersect

▷ `GenIntersect(a, b)` (function)

This function returns a set of generators of the intersection of two groups generated by the elements of the lists `a` and `b`.

Example

```
gap>  a := [Tuple([(2,3),()]),Tuple([(1,2),()]), Tuple([(),(1,2,3)])];
[ DirectProductElement( [ (2,3), () ] ),
  DirectProductElement( [ (1,2), () ] ),
  DirectProductElement( [ (), (1,2,3) ] ) ]
gap>  b := [Tuple([(1,3,2),()]),Tuple([(1,2),()]), Tuple([(),(1,2,3)])];
[ DirectProductElement( [ (1,3,2), () ] ),
  DirectProductElement( [ (1,2), () ] ),
  DirectProductElement( [ (), (1,2,3) ] ) ]
gap> GenIntersect(a,b);
[ DirectProductElement( [ (), () ] ), DirectProductElement( [ (), (1,2,3) ] ),
```

```
    DirectProductElement( [ (), (1,3,2) ] ),
    DirectProductElement( [ (2,3), () ] ),
    DirectProductElement( [ (1,2), () ] ),
    DirectProductElement( [ (1,2,3), () ] ),
    DirectProductElement( [ (1,3,2), () ] ),
    DirectProductElement( [ (1,3), () ] ) ]
```

# Index