Group Theory in OSCAR

Thomas Breuer

RWTH Aachen University

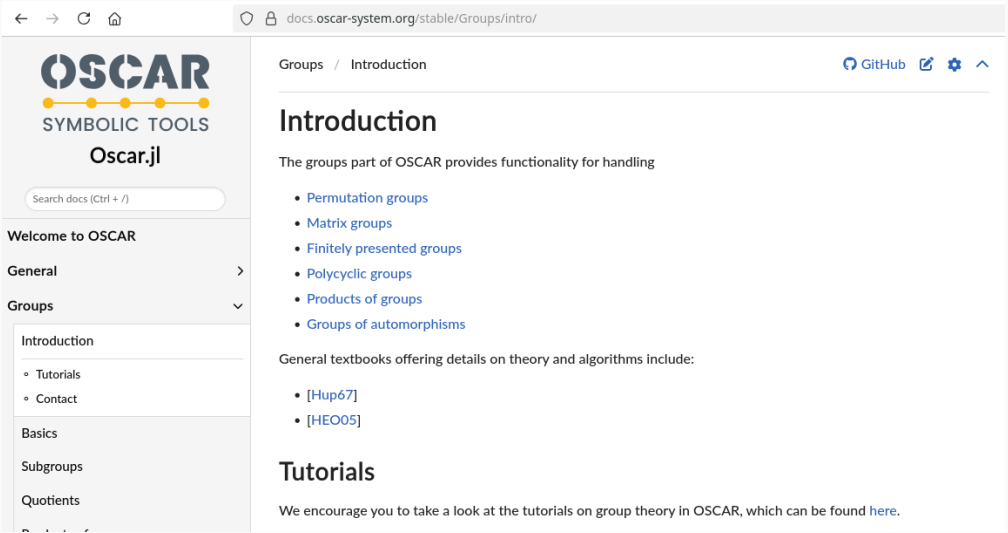
December 1, 2025

Overview

- Kinds of groups (permutation groups, matrix groups, ...)
- Attributes of groups
- Group libraries
- Group actions and G-sets
- Group homomorphisms

(And: explain general concepts, show code examples)

From the OSCAR Manual



In the Julia session

```
julia> using Oscar
  ___   ___   ___    _    ____
 / _ \ / __\ / __\  / \  |  _ \  | Combining and extending ANTIC, GAP,
| |_| |\__ \| |__  / ^ \ |  ´ /  | Polymake and Singular
 \___/ \___/ \___//_/ \_\|_|\_\  | Type "?Oscar" for more information
o--------o-----o-----o--------o  | Documentation: https://docs.oscar-system.org
  S Y M B O L I C   T O O L S    | Version 1.5.1

help?> symmetric_group
  symmetric_group(n::Int)

  Return the full symmetric group on the set {1, 2, ..., n}.

  Examples
  --------
  julia> G = symmetric_group(5)
  Symmetric group of degree 5

  julia> order(G)
  120
```

Kinds of Groups: Permutation groups

```
julia> G = symmetric_group(5)
Symmetric group of degree 5

julia> x, y = gens(G)
2-element Vector{PermGroupElem}:
 (1,2,3,4,5)
 (1,2)

julia> one(G), x * y, inv(x)
((), (2,3,4,5), (1,5,4,3,2))

julia> degree(y)
5

julia> parent(y)
Symmetric group of degree 5
```

Kinds of Groups: Permutation groups

Create permutation groups by generators

```
julia> G = symmetric_group(5);

julia> x = G([2,1], check = false)
(1,2)

julia> H, emb = sub(G, [x])
(Permutation group of degree 5, Hom: H -> G)

julia> gens(H)
1-element Vector{PermGroupElem}:
 (1,2)

julia> order(H)
2
```

Example: Permute a deck of cards

```
julia> function shuffle_group(n::Int)
         p1 = vcat(1:2:(2*n-1), 2:2:(2*n))
         p2 = vcat(2:2:(2*n), 1:2:(2*n-1))

         S = symmetric_group(2*n)
         return sub(S, [S(p1), S(p2)])[1]
       end;
```

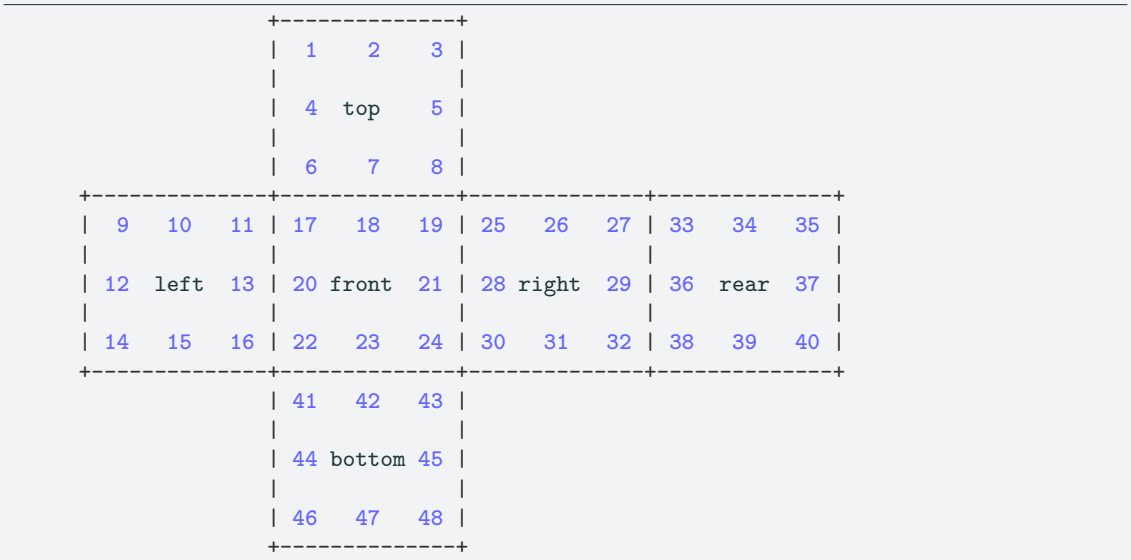
$$
\begin{array}{ccccc}
1 & \to & 1 & & \\
 & & 2 & \leftarrow & n+1 \\
2 & \to & 3 & & \\
 & & 4 & \leftarrow & n+2 \\
. & \to & . & & \\
 & & . & \leftarrow & . \\
. & \to & . & & \\
 & & . & \leftarrow & . \\
. & \to & . & & \\
 & & . & \leftarrow & . \\
n & \to & 2n-1 & & \\
 & & 2n & \leftarrow & 2n
\end{array}
\qquad
\begin{array}{ccccc}
 & & 1 & \leftarrow & n+1 \\
1 & \to & 2 & & \\
 & & 3 & \leftarrow & n+2 \\
2 & \to & 4 & & \\
 & & . & \leftarrow & . \\
. & \to & . & & \\
 & & . & \leftarrow & . \\
. & \to & . & & \\
 & & . & \leftarrow & . \\
. & \to & . & & \\
 & & 2n-1 & \leftarrow & 2n \\
n & \to & 2n & &
\end{array}
$$

Example: Permute a deck of cards

```
julia> println(vcat(1:2:7, 2:2:8))
[1, 3, 5, 7, 2, 4, 6, 8]

julia> symmetric_group(8)(vcat(1:2:7, 2:2:8))
(2,3,5)(4,7,6)
```

```
julia> print([order(shuffle_group(n)) for n in 1:10])
ZZRingElem[2, 8, 24, 24, 1920, 7680, 322560, 64, 92897280, 3715891200]

julia> G = shuffle_group(16)
Permutation group of degree 32

julia> gens(G)
2-element Vector{PermGroupElem}:
 (2,3,5,9,17)(4,7,13,25,18)(6,11,21,10,19)(8,15,29,26,20)(12,23,14,27,22)(16,31,30,28,24)
 (1,2,4,8,16,32,31,29,25,17)(3,6,12,24,15,30,27,21,9,18)(5,10,20,7,14,28,23,13,26,19)(11,22)

julia> order(G)
160
```

Example: Rubik's cube



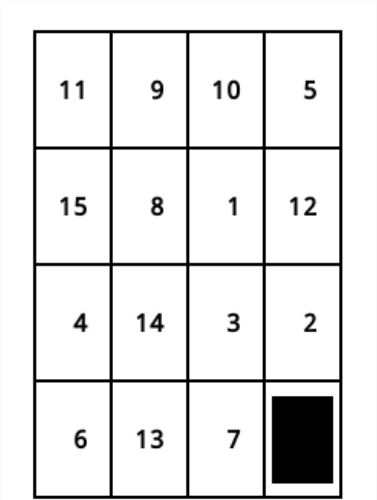
Example: Rubik's cube

```
G = symmetric_group(48);

l = [ [[ 1, 3, 8, 6], [ 2, 5, 7, 4], [ 9,33,25,17], [10,34,26,18], [11,35,27,19]],
      [[ 9,11,16,14], [10,13,15,12], [ 1,17,41,40], [ 4,20,44,37], [ 6,22,46,35]],
      [[17,19,24,22], [18,21,23,20], [ 6,25,43,16], [ 7,28,42,13], [ 8,30,41,11]],
      [[25,27,32,30], [26,29,31,28], [ 3,38,43,19], [ 5,36,45,21], [ 8,33,48,24]],
      [[33,35,40,38], [34,37,39,36], [ 3, 9,46,32], [ 2,12,47,29], [ 1,14,48,27]],
      [[41,43,48,46], [42,45,47,44], [14,22,30,38], [15,23,31,39], [16,24,32,40]] ];

cube = sub(G, [cperm(g, x) for x in l])[1]
```

https://nbviewer.org/github/oscar-system/OSCARBinder/blob/master/rubik.ipynb

Example: Loyd's Fifteen

```
julia> GAP.Globals.BrowsePuzzle()
```



Example: Proportion of fixed-point free permutations

```
julia> function fpf_proportion(n)
         return length(filter(x -> number_of_fixed_points(x) == 0,
                              collect(symmetric_group(n)))) // factorial(n)
       end;
```

```
julia> function fpf_proportion2(n)
         return count(x -> number_of_fixed_points(x) == 0,
                      symmetric_group(n)) // factorial(n)
       end;
```

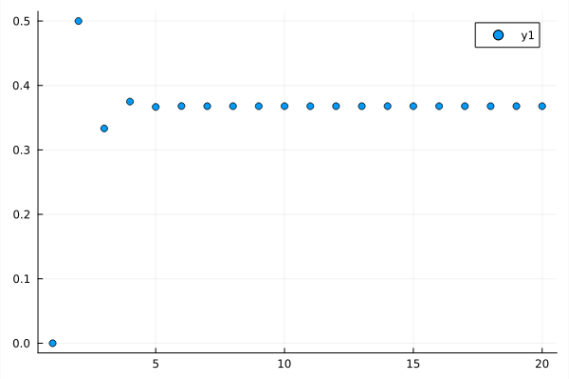
```
julia> function fpf_proportion3(n)
         G = symmetric_group(n)
         fpf = ZZ(0)
         for C in conjugacy_classes(G)
           x = representative(C)
           number_of_fixed_points(x) == 0 && (fpf = fpf + length(C))
         end
         return fpf // order(G)
       end;
```

Example: Proportion of fixed-point free permutations

```
julia> @time res1 = [fpf_proportion(n) for n in 1:10];
  3.946194 seconds (10.38 M allocations: 441.000 MiB, 2.74% gc time, 0.28% compilation time)

julia> @time res2 = [fpf_proportion2(n) for n in 1:10];
  4.128710 seconds (10.42 M allocations: 320.502 MiB, 0.72% compilation time)

julia> @time res3 = [fpf_proportion3(n) for n in 1:10];
  0.015792 seconds (39.64 k allocations: 2.444 MiB, 82.87% compilation time)

julia> res1 == res2 == res3
true
```

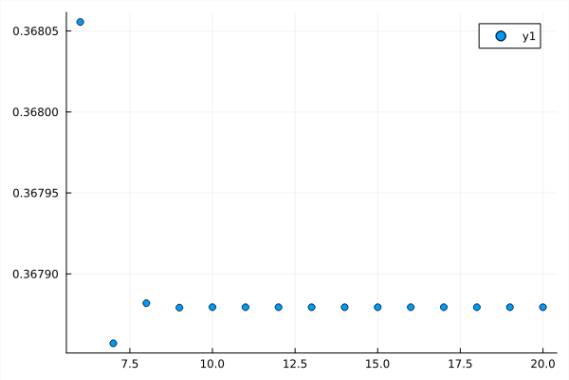
Example: Proportion of fixed-point free permutations

```
julia> using Plots

julia> @time vals = [(n, float(Rational(fpf_proportion3(n)))) for n in 1:20];
  0.077250 seconds (363.78 k allocations: 20.696 MiB, 26.81% compilation time)

julia> plot(first.(vals), last.(vals); seriestype = :scatter)
```



Example: Proportion of fixed-point free permutations

```
julia> vals2 = vals[6:20];

julia> plot(first.(vals2), last.(vals2); seriestype = :scatter)
```



Example: Proportion of fixed-point free permutations

```
julia> x = vals[end][2]
0.3678794411714423216142441641283153196560534804474355603449216202959675657396704

julia> inv(x)
2.7182818284590452352219614301374920588826491041137852613693205622504719693380 87
```

```
julia> exp(-1) - x
1.241003327882150871551164061134943943965507837970403243426032955518564634685 41e-17
```

More evidence needed?

If yes then perhaps rewrite the code: use formulae in terms of partitions

Proof?

Kinds of Groups: Matrix groups

```
julia> G = general_linear_group(2, GF(3))
GL(2,3)

julia> x, y = gens(G)
2-element Vector{MatrixGroupElem{FqFieldElem, FqMatrix}}:
 [2 0; 0 1]
 [2 1; 2 0]

julia> one(G), x * y, inv(y)
([1 0; 0 1], [1 2; 2 0], [0 2; 1 2])

julia> degree(G)
2

julia> base_ring(G)
Prime field of characteristic 3

julia> parent(x)
GL(2,3)
```

Kinds of Groups: Matrix groups

```
julia> mx, my = map(matrix, gens(G))
2-element Vector{FqMatrix}:
 [2 0; 0 1]
 [2 1; 2 0]

julia> x == mx
false

julia> solve(mx, my)  # NOT defined for x, y
[1   1]
[1   0]

julia> comm(x, y)  # NOT defined for mx, my
[2   0]
[2   2]

julia> det(x) == det(mx)  # for convenience
true

julia> G(mx, check = false) == x
true
```

Kinds of Groups: Finitely presented groups

```
julia> F = free_group(2)
Free group of rank 2

julia> x, y = gens(F)
2-element Vector{FPGroupElem}:
 f1
 f2

julia> g = (x^2*y)^6
(f1^2*f2)^6

julia> inv(g)
(f2^-1*f1^-2)^6

julia> G, epi = quo(F, [x^2, y^2, comm(x, y)])
(Finitely presented group, Hom: F -> G)

julia> describe(G)
"C2 x C2"
```

Kinds of Groups: Finitely presented groups

$G = \langle x, y \mid x^2, y^3, (xy)^6 \rangle$

```
julia> F = free_group(2);

julia> x, y = gens(F);

julia> G, epi = quo(F, [x^2, y^3, (x*y)^6]);

julia> is_finite(G)
```

(decidable?)

Kinds of Groups: Polycyclicly presented groups

- finite presentations of a special form
- normal form of elements can be computed
- different (efficient) algorithms for these groups
- definition (and creation by hand) is quite technical
- recommended for polycyclic groups

```
julia> G = small_group(24, 12)
Pc group of order 24
```

Attributes of Groups

boolean

```
is_finite, is_abelian, is_solvable, is_simple, ...
```

numerical

```
order, exponent, number_of_conjugacy_classes, ...
```

subgroups (return the subgroup and its embedding)

```
center, derived_subgroup, frattini_subgroup, sylow_subgroup, ...
```

quotients (return the factor group and the epimorphism)

```
quo, maximal_abelian_quotient, ...
```

subgroup series (return an array of subgroups)

```
derived_series, lower_central_series, ...
```

Group libraries (there are more)

Transitive permutation groups (degree up to 48)

```
julia> describe(transitive_group(12, 295))
"M12"
```

Primitive permutation groups (degree up to 8191)

```
julia> describe(primitive_group(12, 2))
"M12"
```

Perfect groups (order up to $2 \cdot 10^6$)

```
julia> describe(perfect_group(95040, 1))
"M12"
```

Groups of "small" order (all of order ≤ 2000, cubefree order ≤ 50000, ...)

```
julia> describe(small_group(24, 12))
"S4"
```

Group libraries

Similar structure of the libraries:

```
julia> number_of_small_groups(8)
5

julia> grps = all_small_groups(8, !is_abelian)
2-element Vector{PcGroup}:
 Pc group of order 8
 Pc group of order 8

julia> map(describe, grps)
2-element Vector{String}:
 "D8"
 "Q8"

julia> id = small_group_identification(symmetric_group(3))
(6, 1)

julia> small_group(id...)
Pc group of order 6
```

Group actions

G-action on the set Ω: map $\mu\colon \Omega \times G \to \Omega$
s. t. $\mu(\mu(\omega, g), h) = \mu(\omega, gh)$ and $\mu(\omega, 1_G) = \omega$ for all $g, h \in G$ and $\omega \in \Omega$

Natural actions:

```
<int> ^ <perm. gp. element>
<intvec> ^ <perm. gp. element>       # on_tuples
<set> ^ <perm. gp. element>          # on_sets
g ^ h                                # conjugation in G
<polynomial> ^ <perm. gp. element>   # on_indeterminates
<polynomial> ^ <mat. gp. element>    # on_indeterminates
<vector> * <mat. gp. element>
right_coset(H, g1) * g2
```

Custom actions:

```
permuted(<vec>, <perm. gp. element>)
```

G-sets

G-set: the set Ω together with the group G and the action function

```
julia> Omega = gset(symmetric_group(4), [[1, 2]])  # action on ordered pairs
```

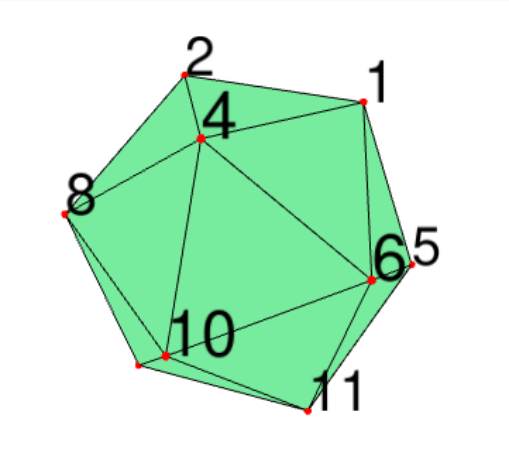
Functionality for G-sets:

```
omega = representative(Omega)
orbit(Omega, omega)
stabilizer(Omega, omega)
is_conjugate_with_data(Omega, omega1, omega2)
is_transitive(Omega)
is_primitive(Omega)
action_homomorphism(Omega)
```

```
julia> G = symmetric_group(4);  H, _ = sylow_subgroup(G, 2);

julia> Omega = right_cosets(G, H);

julia> is_primitive(Omega)
true
```

Example: Symmetries of a regular icosahedron

```
julia> T = platonic_solid("icosahedron")
Polytope in ambient dimension 3 with EmbeddedAbsSimpleNumFieldElem type coefficients

julia> visualize(T)
```



Example: Symmetries of a regular icosahedron

```
julia> aut = automorphism_group(T)
Dict{Symbol, PermGroup} with 2 entries:
  :on_vertices => Permutation group of degree 12
  :on_facets   => Permutation group of degree 20

julia> G = aut[:on_vertices];

julia> describe(G)
"C2 x A5"

julia> gens(G)
3-element Vector{PermGroupElem}:
 (2,5)(4,6)(7,9)(8,11)
 (1,4)(3,8)(5,10)(9,12)
 (1,6,11,12,7,2)(3,4,5,10,9,8)

julia> e = Set([1, 2]);
```

Example: Symmetries of a regular icosahedron

```
julia> e = Set([1, 2]);

julia> edges = orbit(G, e)
G-set of
  permutation group of degree 12
  with seeds [Set([2, 1])]

julia> length(edges)
30

julia> println(filter(x -> 1 in x, elements(edges)))
Set{Int64}[Set([2, 1]), Set([5, 1]), Set([6, 1]), Set([4, 1]), Set([3, 1])]

julia> gens(stabilizer(edges, e)[1])
3-element Vector{PermGroupElem}:
 ()
 (3,4)(5,6)(7,8)(9,10)
 (1,2)(5,7)(6,8)(11,12)
```

Group homomorphisms

A *group homomorphism* $f\colon G \to H$ is a map with domain G and codomain H such that $(g \cdot h)^f = g^f \cdot h^f$ and $(g^{-1)})^f = (g^f)^{-1}$ for all $g, h \in G$.

Define via a `Julia` function:

```
julia> G = cyclic_group(7);

julia> mp = hom(G, G, x -> x^2)
Group homomorphism
  from pc group of order 7
  to pc group of order 7
```

Define by prescribing images of generators:

```
julia> mp = hom(G, G, [x^2 for x in gens(G)])
Group homomorphism
  from pc group of order 7
  to pc group of order 7
```

Group homomorphisms

Define via a G-set:

```
julia> G = symmetric_group(4);

julia> Omega = gset(G, [[1, 2]]);  # action on ordered pairs

julia> acthom = action_homomorphism(Omega);

julia> codomain(acthom)
Symmetric group of degree 12

julia> order(image(acthom)[1])
24
```

Functionality of group homomorphisms:

`image` of element or subgroup,
`preimage` of element or subgroup,
`has_preimage_with_preimage` of element,
`is_injective`, `is_surjective`, `kernel`

Thanks ...

...for your attention.