Group Theory and Beyond in OSCAR

Max Horn

RPTU University Kaiserslautern-Landau

5 May, 2026

Overview

Permutation Groups

Group Libraries

Symmetries of a regular icosahedron

Invariant Theory

Symmetries of Number Fields

Speedup via `Julia`

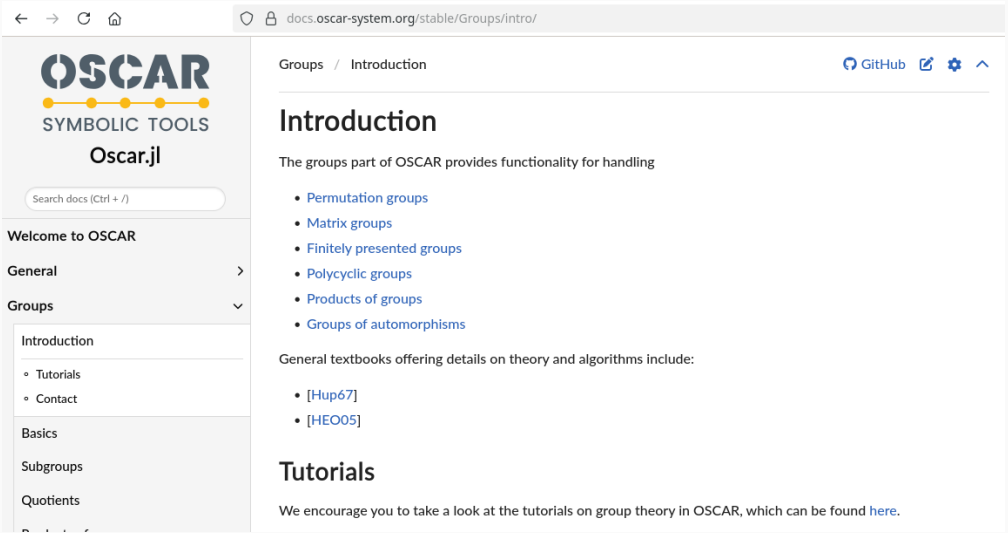
Use `OSCAR` functionality in `GAP`

GAP

Let's look at some examples for things OSCAR can do

Disclaimer: most group theoretical functionality in OSCAR is derived from GAP

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 1.7.2

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
```

Permutation Groups

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: Proportion of fixed-point free permutations

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

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

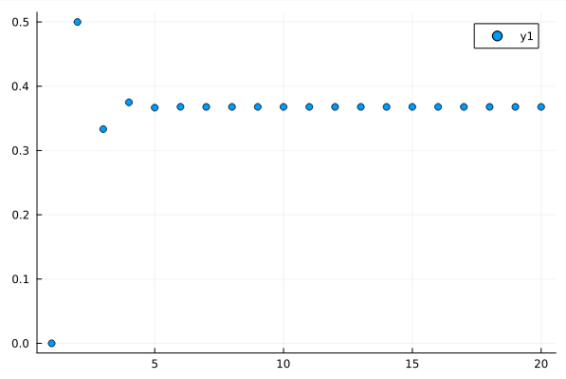
```
function fpf_proportion3(n)
  G = symmetric_group(n)
  fpf = ZZ(0)
  for C in conjugacy_classes(G)
    x = representative(C)
    number_of_fixed_points(x) == 0 && (fpf += length(C))
  end
  return fpf // order(G)
end
```

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
```

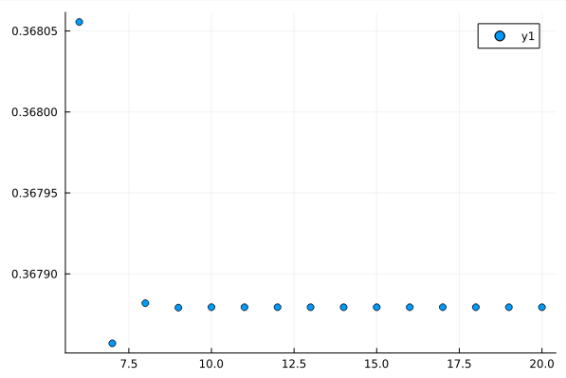
Example: Proportion of fixed-point free permutations

```
julia> using Plots

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

julia> plot(vals; seriestype = :scatter)
```



Example: Proportion of fixed-point free permutations

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

julia> plot(6:20, vals2; seriestype = :scatter)
```



Example: Proportion of fixed-point free permutations

```
julia> x = vals[end][2]
0.367879441171442321614244164128315319656053480447435560344921620295967565739670 4
```

```
julia> inv(x)
2.718281828459045235221961430137492058882649104113785261369320562250471969338087
```

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

Group Libraries

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

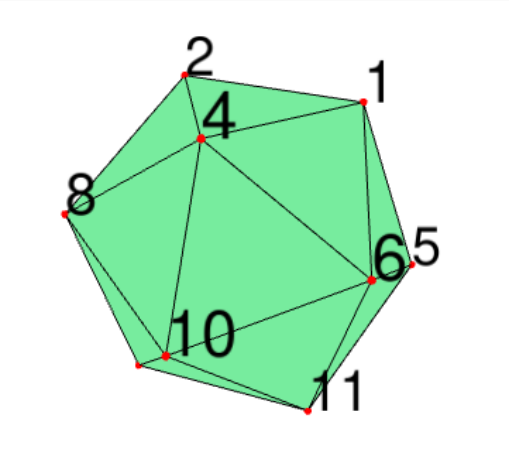
Similar structure of the libraries:

```
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
```

Symmetries of a regular icosahedron

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> G = automorphism_group(T; action=:on_vertices)
Permutation group of degree 12

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]);

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

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)
```

Invariant Theory

Invariant Theory

- k field
- $G \leq \mathrm{GL}_d(k)$ matrix group
- $R := k[\underline{x}] = k[x_1, \ldots, x_d]$
- $A \in G$ acts on $f \in R$ via $f^A(\underline{x}) = f(A.\underline{x})$
- Invariant ring: $R^G := \{f \in R : f^A = f \text{ for all } A \in G\}$
- When G is finite then R^G is finitely generated.

Task: Determine a finite generating set of R^G.

algorithmic solution requires Gröbner bases, group theory

Invariants for the Klein four group

```
julia> A = matrix(QQ, [0 1 0 0; 1 0 0 0; 0 0 0 1; 0 0 1 0]);  # permute (12)(34)

julia> B = matrix(QQ, [0 0 1 0; 0 0 0 1; 1 0 0 0; 0 1 0 0]);  # permute (13)(24)

julia> G = matrix_group(A, B);

julia> describe(G)
"C2 x C2"

julia> RG = invariant_ring(G);

julia> fundamental_invariants(RG)
5-element Vector{MPolyDecRingElem{QQFieldElem, QQMPolyRingElem}}:
 x[1] + x[2] + x[3] + x[4]
 x[1]^2 + x[2]^2 + x[3]^2 + x[4]^2
 x[1]*x[2] + x[3]*x[4]
 x[1]*x[3] + x[2]*x[4]
 x[1]^3 + x[2]^3 + x[3]^3 + x[4]^3

julia> molser = molien_series(RG)
(t^2 - t + 1)//(t^6 - 2*t^5 - t^4 + 4*t^3 - t^2 - 2*t + 1)

julia> expand(molser, 8)
1 + t + 4*t^2 + 5*t^3 + 11*t^4 + 14*t^5 + 24*t^6 + 30*t^7 + 45*t^8 + O(t^9)
```

Symmetries of Number Fields

Symmetries of number fields: explicit automorphisms

```
julia> _, x = polynomial_ring(QQ, :x)
(Univariate polynomial ring in x over QQ, x)

julia> K, z = number_field(x^2 - 7, :z);

julia> G, mp = automorphism_group(PermGroup, K)
(Permutation group of degree 2, Map: G -> generic group of order 2 with multiplication table

julia> g = gen(G, 1)
(1,2)

julia> alpha = mp(g)
Map
  from number field of degree 2 over QQ
  to number field of degree 2 over QQ
defined by
  z -> -z

julia> alpha(z)
-z
```

Symmetries of number fields: Galois groups

```
julia> _, x = polynomial_ring(QQ, :x);

julia> K, a = number_field(x^4 - 2);

julia> G, _ = automorphism_group(K);  describe(G)
"C2"

julia> G, C = galois_group(K); describe(G)
"D8"

julia> G
Permutation group of degree 4 and order 8

julia> C
Galois context for x^4 - 2 and prime 11

julia> roots(C, 2)
4-element Vector{QadicFieldElem}:
 (8*11^0 + O(11^2))*a + 8*11^0 + 9*11^1 + O(11^2)
 (3*11^0 + 10*11^1 + O(11^2))*a + 7*11^0 + 4*11^1 + O(11^2)
 (3*11^0 + 10*11^1 + O(11^2))*a + 3*11^0 + 11^1 + O(11^2)
 (8*11^0 + O(11^2))*a + 4*11^0 + 6*11^1 + O(11^2)
```

Speedup via Julia

Low level computations: speedup via Julia?

Let q, n, z be positive integers, where z divides $q^n - 1$.

Compute the q-adic expansion of $(q^n - 1)/z$.

Why is this interesting:

Define an $\mathbb{Z}$-algebra $A[q, z]$

whose basis is parametrized by the q-adic expansions of $i \cdot (q^n - 1)/z$, for $0 \leq i \leq z$.

See
The Loewy structure of certain fixpoint algebras, Part II,
Int. Electron. J. Algebra 30 (2021), 16–65.

Julia Speed: Naive GAP solution

Task: Compute q-adic expansion of $e = (q^n - 1)/z$

First naive version in pure GAP.

```
gap> CoefficientsQadic( 1000, 3 );
[ 1, 0, 0, 1, 0, 1, 1 ]
gap> z:= 9439;; q:= 22;; n:= z - 1;; e:= (q^n-1) / z;;
gap> coeffs:= CoefficientsQadic( e, q );;
gap> Length( coeffs );
9436
gap> for i in [1..1000] do CoefficientsQadic( e, q );; od; time;
1625  # in milliseconds; thus 1625 mu for one run
```

Julia Speed: Improved GAP solution

Better: Compute the expansion of e without creating e

```
CoeffsQadicReversed:= function( z, q, n )
  local v, i, r, rq, d;
  v:= [];
  r:= 1;
  for i in [ 1 .. n ] do
    rq:= r * q;
    d:= QuoInt( rq, z );
    r:= rq - d * z;
    v[i]:= d;
  od;
  return v;
```

```
gap> for i in [1..1000] do CoeffsQadicReversed(z, q, n); od;
gap> time;
354  # milliseconds, thus 354 mu for one run
```

Before it took 1625 μs $\rightsquigarrow$ speed-up by a factor of ≈ 4.6

Julia Speed: Julia solution

An analogous Julia function

```
function coeffs_qadic_reversed(z::T, q::T, n::Int) where T
  v = [zero(z) for i in 1:n]
  r = one(z)
  for i in 1:n
    v[i], r = divrem(r * q, z)
  end
  return v
end
```

```
julia> z = 9439;  q = 22;  n = z-1;
julia> using Chairmarks
julia> @b coeffs_qadic_reversed(z, q, n)
33.600 mu (3 allocs: 73.820 KiB
```

Before it took 354 μs $\rightsquigarrow$ another speed-up by a factor of ≈ 10.5

Use OSCAR functionality in GAP

Use OSCAR's QQBarField functionality in GAP

GAP provides number fields generated by square roots of integers,
as subfields of cyclotomic fields.

The GAP package CoRelG provides a more efficient implementation of such fields.

In June 2025, Willem de Graaf and Heiko Dietrich asked for an easy way to provide the functionality of OSCAR's QQBarField (the algebraic closure of the field of Rationals) in GAP:

- implement GAP objects that wrap OSCAR's QQBarFieldElem,
- define field arithmetics for the new objects, delegating the work to OSCAR,
- provide a function that computes roots of a polynomial,
- provide complex conjugation,
- provide a positivity test for real elements in the field,
- provide a GAP field object QQBarField

This task required about 200 lines of straightforward GAP code.

Use OSCAR's QQBarField functionality in GAP

The code can be found in the `OscarInterface` GAP package which is part of OSCAR.

```
gap> F:= QQBarField;
QQBarField
gap> x:= One( F );
<{a1: 1.00000}>
gap> z:= Zero( F );
<{a1: 0}>
gap> z < x;
true
gap> y:= 2 * x;;
gap> r:= Sqrt( y );
<{a2: 1.41421}>
gap> M:= [ [ z, r ], [ r, z ] ];;
gap> Determinant( M ) = -2;
true
gap> MinimalPolynomial( M );
x_1^2+(<{a1: -2.00000}>)
gap> Eigenvalues( F, M );
[ <{a2: 1.41421}>, <{a2: -1.41421}> ]
```

The End

That's it for today!

Tomorrow: The OSCAR-GAP Interface – Behind the Scenes

Thank you for your attention!