OSCAR beyond groups

Thomas Breuer

RWTH Aachen University

December 2, 2025

Overview

essentially show examples

- Invariant Theory
- Symmetries of number fields
- Serialization of data
- Speedup via `Julia`
- Use OSCAR functionality in `GAP`
- (One more example)

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]);

julia> B = matrix(QQ, [0 0 1 0; 0 0 0 1; 1 0 0 0; 0 1 0 0]);

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: 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)
```

Serialization: Aims

- Save OSCAR data to a text file,
  such that loading the file into another OSCAR session produces objects with the same meaning and the same internal structure.

  (For example,
  think of several groups and group elements which are related to each other.)
- Exchange data with collaborators.
- Exchange data with other computer algebra systems.
- Provide a standardized format for research data.

  Note that many papers cite reaserch data collections.
  It should be easy to access these data.

Serialization: FAIR data

More generally, research data should be **FAIR** (2016):

- **F**indable (where? listed in some searchable tables of contents?)
- **A**ccessible (describe how to access the data)
- **I**nteroperable (use the data together with other data)
- **R**eusable (easy to use in other systems, data format)

MaRDI project, see `https://www.mardi4nfdi.de`

Della Vecchia, A., Joswig, M., Lorenz, B.,
A FAIR File Format for Mathematical Software,
`https://doi.org/10.1007/978-3-031-64529-7_25`

Think about such questions before you create huge amounts of data.

Rule of thumb:
Better do not rely on a particular language (`GAP`, `OSCAR`, `MAGMA`).

Serialization: a vector of integers

```
julia> v = [1, 2, 3];

julia> save("savev.json", v)

julia> l = load("savev.json");

julia> l == v
true
```

```
{"_ns":
   {"Oscar": ["https://github.com/oscar-system/Oscar.jl", "1.5.1"]},
 "_type":
   {"name": "Vector",
    "params": "Base.Int"
   },
 "data":
    ["1","2","3"]
}
```

Serialization: a vector over a finite field

```
julia> F = GF(3);

julia> v = [one(F), zero(F)];

julia> save("savev.json", v)
```

```
{"_ns":
   {"Oscar": ["https://github.com/oscar-system/Oscar.jl", "1.5.1"]},
 "_type":
   {"name": "Vector",
    "params": {"name": "FqFieldElem",
               "params":"80d095ef-d3fa-41bf-a105-79167828e81f"}},
 "data": ["1","0"],
 "_refs":
   {"80d095ef-d3fa-41bf-a105-79167828e81f":
      {"_type":
         {"name": "FiniteField",
          "_instance": "FqField"},
       "data": "3"}}}
```

Serialization: a vector over a finite field

```
julia> F = GF(3);

julia> v = [one(F), zero(F)];

julia> save("savev.json", v)

julia> l = load("savev.json");

julia> l == v
true

julia> l === v
false

julia> parent(v[1])
Prime field of characteristic 3

julia> parent(v[1]) === parent(l[1]) === F
true
```

Serialization: elements of a free group

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

julia> v = gens(F);

julia> save("savev.json", v)

julia> l = load("savev.json");

julia> l == v
true

julia> l === v
false

julia> parent(v[1]) === parent(l[1]) === F
true
```

Serialization: elements of a free group


```
{"_ns":
   {"Oscar": ["https://github.com/oscar-system/Oscar.jl", "1.5.1"]},
 "_type":
   {"name": "Vector",
    "params": {"name":"FPGroupElem",
               "params":"403f40df-e7b0-4059-acf8-5e6f09e9ea61"}},
 "data":[["1","1"],["2","1"]],
 "_refs":
   {"403f40df-e7b0-4059-acf8-5e6f09e9ea61":
      {"_type":
         {"name":"FPGroup",
          "params":"05420dbb-15b7-4403-91b0-5fa080c75c59"},
       "data": []},
    "05420dbb-15b7-4403-91b0-5fa080c75c59":
      {"_type": "GapObj",
       "data": {"GapType": "IsFreeGroup",
                "wfilt": "IsLetterWordsFamily",
                "names": ["f1","f2"]}}}}
```

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;
end;;
```

```
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'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

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

```
gap> F:= QQBarField;
QQBarField
gap> x:= One( F );
<Root 1.00000 of x - 1>
gap> z:= Zero( F );
<Root 0 of x>
gap> z < x;
true
gap> y:= 2 * x;;
gap> r:= Sqrt( y );
<Root 1.41421 of x^2 - 2>
gap> M:= [ [ z, r ], [ r, z ] ];;
gap> Determinant( M ) = -2;
true
gap> MinimalPolynomial( M );
x_1^2+(<Root -2.00000 of x + 2>)
gap> Eigenvalues( F, M );
[ <Root 1.41421 of x^2 - 2>, <Root -1.41421 of x^2 - 2> ]
```

Example: Certain S-characters of groups

G group
$A = \mathrm{RatIrr}(G) \in \mathbb{Z}^{m \times n}$, $A_{1,j} = 1$ for all j
columns of A indexed by the conj. classes of G: $g_1^G, \ldots, g_n^G$
let $J = \{j; g_j \text{ has prime power order}\}$.

Question:
Is there $v \in \mathbb{Z}^m \setminus \{(1, 0, \ldots, 0)\}$ such that

- $\psi = v \cdot A \geq 0, v_1 = 1$ and
- $\psi_j \neq 0$ for all $j \in J$? $\qquad (*)$

Idea 1:
$\{x \in \mathbb{R}^m; x \cdot A \geq 0\}$ is a simplex.
Enumerate its lattice points, check which of them satisfy $(*)$.
Better force **a priori** $(x \cdot A)_j \geq 1$ for $j \in J$,
then the lattice points of this simplex are exactly the solutions.
This functionality is available in `OSCAR`, via Polymake.

Example: Certain S-characters of groups

Example: the group A_8

```
julia> using S_characters

julia> t = character_table("A8");

julia> P, _, _ = s_character_simplex(t, ppow_nonzero = true);

julia> lattice_points(P)
2-element SubObjectIterator{PointVector{ZZRingElem}}:
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
 [1, 1, 1, 1, 1, 1, 2, 2, 3, 3, 3]
```

Example: Certain S-characters of groups

Idea 2:
If we are interested just in the *existence* of a solution then we can argue as follows:

A *mixed integer linear program* (MILP) yields a solution (if one exists) that is optimal w. r. t. a target function.

```
function one_solution_via_milp(tbl::Oscar.GAPGroupCharacterTable)
  P, galoissums, _ = s_character_simplex(tbl,
                       rational = true,
                       ppow_nonzero = true)
  d = length(galoissums)
  milp = mixed_integer_linear_program(P, repeat([1], d))
  _, sol = solve_milp(milp)
  return !is_zero(sol), sol
end;

julia> one_solution_via_milp(character_table("A8"))
(true, QQFieldElem[1, 1, 1, 1, 1, 1, 2, 2, 3, 3, 3])
```

Example: Certain S-characters of groups

The MILP-approach is much faster than computing all solutions ...

...**and** works also for larger examples,
where enumerating all solutions fails.

```
julia> one_solution_via_milp(character_table("ON"))
(true, QQFieldElem[688, 840, 1626, 1679, 2033, 2033, 2363, 3304, 3659, 3658  …
 3678, 4063, 5334, 7337, 8995, 10617, 11015, 11027, 13006, 14681])
```

Example: Certain S-characters of groups

Idea 3:

In fact, we are interested in a more general problem.
Our matrix A is *in general* real but not integral, with entries in some number field.

The exhaustive approach still works, **because** Polymake's programs can deal with matrices over `OSCAR`'s embedded number fields.

(We need field arithmetics and positivity checks.)

Drawback:
We have to enumerate the lattice points of an in general too big simplex, and can check condition $(*)$ only afterwards.
Thus the MILP-approach does **not** work in this more general situation.

See `https://github.com/oscar-system/FriendsOfOscar` for the code.

Thanks ...

...for your attention.