

Introduction to OSCAR

Max Horn

December 1, 2025



Overview For The OSCAR Minicourse

1. "Introduction to OSCAR" (Max)
2. "Group Theory in OSCAR" (Thomas)
3. "The OSCAR-GAP Interface – Behind the Scenes" (Max)
4. "OSCAR Beyond Group Theory" (Thomas)

Overview for this talk

What and Why Is OSCAR?

The Structure of OSCAR

Why Julia?

Getting Started & Getting Help

Summary

What and Why Is OSCAR?

What is OSCAR?

<https://oscar-system.org>

- Open Source Computer Algebra Research system
- For interdisciplinary research & computations in algebra, geometry, and number theory
- Funded by German Research Council (DFG) from 2017-2028 via the Collaborative Research Center SFB-TRR 195
- Contributors from all over the world
- All parts of OSCAR are open source (like GAP, PARI, ...)
- Open to contributions from anyone, curated via code reviews

Who uses OSCAR?

- OSCAR's primary audience are researchers in pure maths and adjacent field
- usually *not* software experts
- They use it e.g. for ...
 - experiments and exploration;
 - proofs (e.g. theory handles all but finitely many exceptions, software does the rest)
 - creating or accessing databases of objects
 - and much more, including many unexpected things
- OSCAR is also used for teaching; by math enthusiast; and many others
- All are welcome, but design decisions are driven by primary audience

Some OSCAR Capabilities

- *efficient basic arithmetic*: polynomials, matrices, finite fields, number fields, power series, groups, ...with common interfaces
- generic and specialized optimized *linear algebra*
- *group theory*: permutation groups, finitely presented groups, matrix groups, group cohomology, ...
- *commutative algebra*: Gröbner bases, (graded) modules over fin. gen. rings, affine algebras, primary decomposition, ...
- *number theory*: class groups, Galois groups, ...
- *algebraic geometry*: schemes, (elliptic) curves, toric varieties, ...
- *polyhedral geometry, tropical geometry*
- *noncommutative algebra*: PBW-algebras, GR-algebras, ...
- *Lie theory*: root systems, Weyl groups, Lie algebras, modules, ...
- ...and much more is there, or to come

A Tiny Taste: Matrix Groups over $\overline{\mathbb{Q}}$

```
julia> K = algebraic_closure(QQ)
Algebraic closure of rational field

julia> v = K(2//5)
{a1: 0.4000000}

julia> s, c = sinpi(v), cospi(v)
({a4: 0.951057}, {a2: 0.309017})

julia> mat_rotation = matrix([ c -s ; s c ]);

julia> mat_reflection = matrix(K, [ -1 0 ; 0 1 ]);

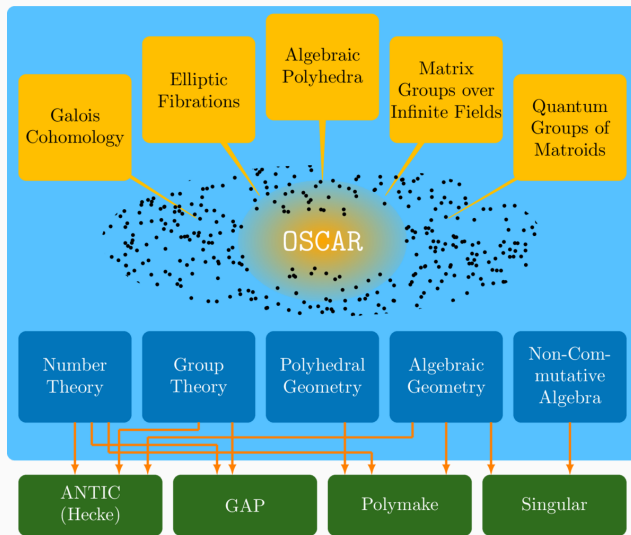
julia> G = matrix_group(mat_rotation, mat_reflection)
Matrix group of degree 2
over algebraic closure of rational field

julia> describe(G)
"D10"
```

Can now also e.g. visualize the action on points \rightsquigarrow Jupyter notebook

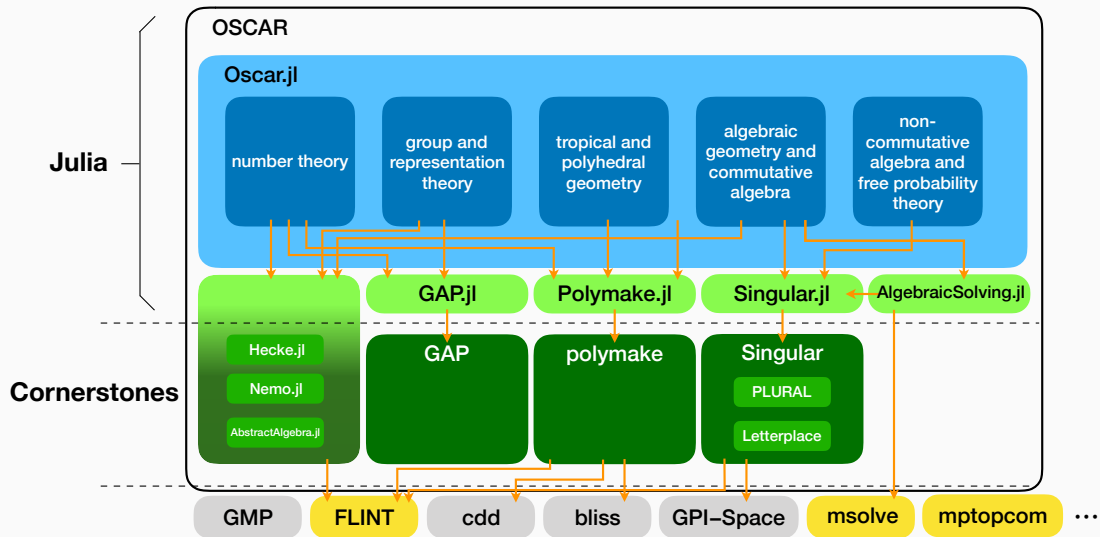
The Structure of OSCAR

The Structure of OSCAR (topical)



- OSCAR is “young” ...
- ...but based on existing tools, foremost its four *cornerstones*:
 - ANTIC (Nemo, Hecke)
 - GAP
 - polymake
 - Singular
- ...but also GMP, FLINT, msolve, and many more
- coupled together and extended by code written in the Julia programming language

The Structure of OSCAR (technical)



Goal: combining the corner stones seamlessly

- In GAP there are interfaces for calling Singular, PARI, polymake, ...
- To use them requires some expertise in both GAP and the other system

```
FF:=Polymake(polygon,"FACES");
FA:=Polymake(polygon,"ADJACENCY");
D:=Polymake(polygon,"F_VECTOR");
if not Size(Filtered(FF,x->Size(x)=1))=D[1] then D:=Reversed(D); fi;
if not FF[1]=[] then FF:=Reversed(FF); FA:=Reversed(FA); bool:=true; fi;
if toggle then F:=FF; else F:=FA; fi;
```

- Our goal is to create a coherent system

```
julia> c = cube(3)
Polytope in ambient dimension 3

julia> visualize(face_poset(cube(3)))
```

Why Julia?

Performance

- Solves the “2-language problem”: Want language that ...
 1. is easy to write & use (like GAP, Python, ...)
 2. offers near C performance due to “just-in-time” compilation
- supports parallel computing

Community and Ecosystem

- open source
- supports multiple platforms
- designed by mathematically minded people
- large, steadily growing ecosystem

Our primary audience are researchers who are not software experts ...

Julia features

- good support for interactive use in the REPL:
 - searchable history, tab-completion, help mode
- first-class Jupyter notebook support
- automatic memory management (garbage collector)
- dynamically typed
- multiple dispatch

For developers: great C interoperability; good C++ support \rightsquigarrow enables integration of software systems

Calling C code

Julia also integrates very well with the C programming language (lots of existing software is written in C).

```
julia> x = 1.0;
```

```
julia> sin(x)      # call native Julia function
0.8414709848078965
```

```
julia> @ccall sin(x::Float64)::Float64  # call C math library
0.8414709848078965
```

```
julia> u = Int64[1,2,3,4,5,6];
```

```
julia> @ccall memset(u::Ptr{Int}, 0::Int, sizeof(u)::UInt)::Cvoid
```

```
julia> show(u)
[0, 0, 0, 0, 0, 0]
```

Native code

- Julia code is compiled just in time to performant machine code
- Callbacks from C into Julia are easy
- Allows easy access to high-performance C or fortran libraries
- Machine code can be inspect interactively

```
julia> f(x,y) = (x+42)*y;
```

```
julia> @code_native debuginfo=:none binary=false dump_module=false f(1,2)
```

```
.section      __TEXT,__text,regular,pure_instructions
ldr          x8, [x20, #16]
ldr          x8, [x8, #16]
ldr          xzr, [x8]
add          x8, x0, #42
mul          x0, x8, x1
ret
```

Julia Speed: a simple combinatorics example

“Pure” GAP:

```
Julia> @b GAP.Globals.Combinations(GAP.Obj(1:10), 5)
86.917 μs (1715 allocs: 76.391 KiB)
```

Native OSCAR version:

```
Julia> @b combinations(1:10, 5)
1.232 ns
```

Actually that was cheating \rightsquigarrow created a “lazy” object...

```
Julia> @b collect(combinations(1:10, 5))
6.097 μs (509 allocs: 25.781 KiB)
```

We get full access to Julia and its many packages, and thus for example functionality for:

- Jupyter notebooks (Jupyter = JULia, PYthon, R)
- Parsing and writing many file formats (JSON, YAML, CSV, TOML, ...)
- Databases (PostgreSQL, MongoDB, ...)
- Numerics tools (e.g. solvers for ODEs, PDEs, ...)
- Statics tools, charts, plotting, ...
- GPU acceleration

Getting Started & Getting Help

Where to begin learning OSCAR

- You (hopefully) already started in the past weeks via one of our tutorials
- ...and of course the
- And of course in the next three sessions we'll “get our hands dirty”
- But in the end: just try to use it, and ask many questions!

Can OSCAR do X?

- So you want to do something, e.g. compute invariants of a group
- Question: Can OSCAR do this? And if so, how?
- How can we find out?
- Option 1: Try it interactively, use the help system
- Option 2: Look at the documentation
- Option 3: Ask someone
 - in person (colleagues in the hallway, experts here at the school, ...)
 - via email
 - via Slack

docs.oscar-system.org



oscar-system.org/slack



And what if OSCAR can't do it?

- What if your favorite feature is not there, or at least you can't find it?
- First make sure it *really* isn't there ~→ talk to us
- It might be on the road map, though ~→ talk to us
- The fact that *you* need it might put it on the roadmap! ~→ talk to us
- Perhaps *you* could be add it (with help from the team)? ~→ talk to us
- We appreciate contributions of all kinds (that very much includes bug reports, feature requests, suggestions for improving the documentation, etc.)
- (By the way: the same holds true for GAP, Singular, Pari, ...)

Summary

- OSCAR brings together a collection of distinct tools and molds them into one
- Wide variety of topics supported, more are being added
- Combines them with help of Julia and engineering effort
- All this is possible because all involved components are **open source**
- Reward: can tackle problems the parts on their own cannot
- Next talk: group theory in OSCAR