Introduction to OSCAR

Max Horn

RPTU University Kaiserslautern-Landau

4 May, 2026

Overview for The OSCAR Minicourse

1. "Introduction to OSCAR" (this talk)

2. "Group Theory and Beyond in OSCAR"

3. "The OSCAR-GAP Interface – Behind the Scenes"

Overview for This Talk

What and Why Is OSCAR?

The Structure of OSCAR

Why Julia?

Getting Started & Getting Help

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

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

Installing OSCAR

1. Install `Julia`
   Recommendation: via Juliaup; enter this in a terminal, press return, answer questions:

   ```
   curl -fsSL https://install.julialang.org | sh
   ```

2. Open the Julia REPL in your terminal by typing the following:

   ```
   julia
   ```

3. Install OSCAR by running the following commands. This may take a while, as OSCAR and its dependencies will be downloaded and installed.

   ```
   using Pkg
   Pkg.add("Oscar")
   ```

Launching OSCAR from a terminal

```
$ julia
```

```
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.12.6 (2026-04-09)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |
```

```
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

julia>
```

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

```
julia> K = algebraic_closure(QQ)        # utilize number theory
Algebraic closure of rational field

julia> v = K(2//5)
{a1: 0.400000}

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)     # uses GAP under the hood
"D10"
```

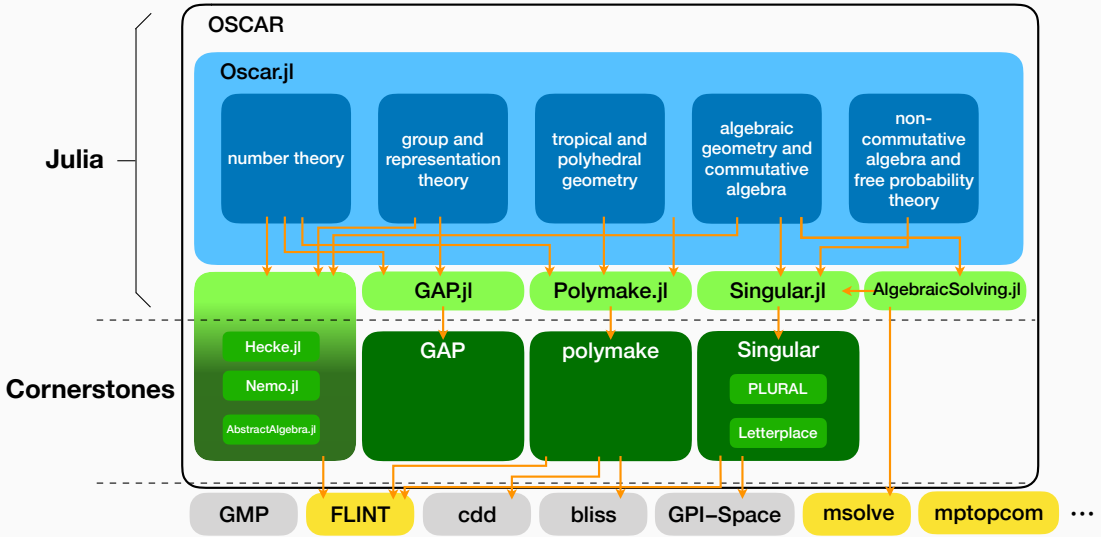
Can now also e.g. visualize the action on points $\rightsquigarrow$ Jupyter notebook

The Structure of OSCAR

OSCAR Cornerstones

- OSCAR is “new” ...
- ...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
- Here is a snippet from the GAP package HAP, interacting with `polymake`

```
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;
```

- For OSCAR we strive to create a coherent system, as seen in the "matrix group over algebraic closure" example
- This is a huge (possibly endless) job $\rightsquigarrow$ Demand Driven Development: you need it? then tell us (or do it and submit a patch)

Example: Polyhedral Geometry

Recall the HAP code snippet, which uses the `polymaking` package:

```
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;
```

In OSCAR, we aim for higher level abstractions (in this case, written by the polymake team):

```
julia> c = cube(3)    # just as an example
Polytope in ambient dimension 3

julia> println(f_vector(c))
ZZRingElem[8, 12, 6]

julia> fp = face_poset(c)
Partially ordered set of rank 4 on 28 elements

julia> visualize(fp)
```

Why Julia?

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

Why Julia?

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
- great C interoperability; good C++ support $\rightsquigarrow$ enables integration of software systems

Bonus points for friends of GAP

- actually looks not to different from GAP
- 1-based indexing is familiar for GAP users

Calling C code

Julia also integrates very well with the C programming language (lots of existing software is written in C), e.g. using the `@ccall` macro

```
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
- Generated machine code can be inspected interactively

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

julia> @code_native f(1,2)
; Function Signature: f(Int64, Int64)
; ┌ @ REPL[1]:1 within `f`
        ;DEBUG_VALUE: f:x <- $x0
        ;DEBUG_VALUE: f:y <- $x1
; │┌ @ int.jl:87 within `+`
        add         x8, x0, #42
; │└
; │┌ @ int.jl:88 within `*`
        mul         x0, x8, x1
        ret
; └└
```

Julia Speed: a simple combinatorics example

Below we measure using the macro `@b` from the `Julia` package `Chairmarks`

"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 ⇝ created a "lazy" object...

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

Rich ecosystem

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

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 out; use the help system
- Option 2: Look at the documentation
- Option 3: Ask someone (e.g. me or Thomas Breuer)

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 roadmap, 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
- ⇝ **Demand Driven Development**
- We appreciate contributions of all kinds: bug reports, feature requests, suggestions for improving the documentation, code contributions, etc.

The End

That's it for today!

Tomorrow: Group Theory and Beyond in OSCAR

Thank you for your attention!