The OSCAR Computer Algebra System

Max Horn

OSCAR SYMBOLIC TOOLS

TR 195 SYMBOLIC TOOLS

March 9, 2022

What is OSCAR?

https://oscar.computeralgebra.de/

- OSCAR is an Open Source Computer Algebra Research system
- funded by SFB-TRR 195 of the DFG, planned in three phases 2017-2028

  Develop a visionary, next generation, open source computer algebra system, integrating all systems, libraries and packages developed within the TRR.

- built using the Julia language, see https://julialang.org
- a tool for interdisciplinary research and computations in algebra, geometry, and number theory

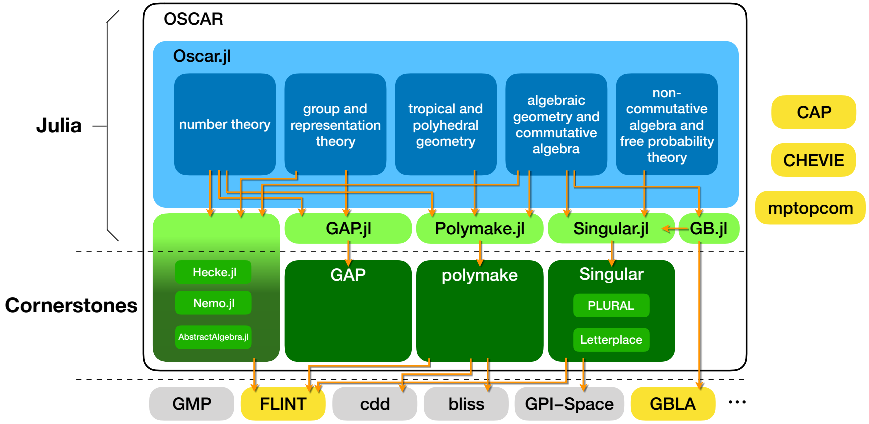
Some features of `OSCAR`

To give some flavor of what `OSCAR` is or aims to be:

- efficient basic arithmetic (polynomials, matrices, finite fields, number fields, power series, groups, ...) with common interfaces
- generic and specialised optimised linear algebra
- factorisation (integers, polynomials)
- commutative algebra: Gröbner bases, (graded) modules, affine algebras, primary decomposition, ...
- number theory: class groups, Galois groups, ...
- algebraic geometry: curves, toric varieties, ...
- group theory: permutation/finitely presented/matrix groups, group cohomology, ...
- invariant theory of groups, ...
- ...and much more to come

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

The structure of OSCAR



gray: externally developed; yellow: developed by members of SFB; green: cornerstones and interfaces; blue: new additions in phase 2;

Why Julia?

- Want to write code in a modern high-level language, but which one?
- Not a custom one: Want to develop computer algebra, not a language!
- Open Source (MIT License)
- friendly (imperative) syntax
- modern features, vibrant ecosystem
- JIT compilation: near C performance
- $\rightsquigarrow$ solves the “two language problem”
- easy/efficient C interoperability; good C++ support
- excellent console/REPL mode, but also e.g. Jupyter support

Installing OSCAR

```
julia> using Pkg ; Pkg.add("Oscar")  # install it first time around
... [wait some time] ...
julia> using Oscar
 -----    -----    -----      -      -----
|     |  |     |  |     |    | |    |     |
|     |  |        |         |   |   |     |
|     |   -----   |        |     |  |-----
|     |        |  |        |-----|  |   |
|     |  |     |  |     |  |     |  |    |
 -----    -----    -----   -     -  -     -

...combining (and extending) ANTIC, GAP, Polymake and Singular
Version 0.8.1 ...
 ... which comes with absolutely no warranty whatsoever
Type: ’?Oscar’ for more information
(c) 2019-2022 by The Oscar Development Team

julia>
```

Interfaces: Status

- The primary systems interfaces are:
  - `GAP.jl`: complete, all functionality of all packages is available, `GAP` can call any `Julia` function and vice versa
  - `Polymake.jl`: complete, all functions can be used
  - `Singular.jl`: the core functionality is available, some kernel and library functions lack wrappers.
- For `GAP` everything and `polymake` the wrappers are automated, for `Singular` manual work (possibly involving C++ code) is needed – the "meta-data" (types) is missing.

Interface example: GAP

- Lowest level: use GAP commands in OSCAR, but they look like GAP:

```
julia> GAP.Globals.SymmetricGroup(5)
GAP: Sym( [ 1 .. 5 ] )

julia> GAP.Globals.DerivedSeries(ans)
GAP: [ Sym( [ 1 .. 5 ] ), Alt( [ 1 .. 5 ] ) ]

julia> typeof(ans)
GAP.GapObj
```

- Similar for polymake and Singular

OSCAR-ification

- Current work: "OSCAR-ify" the GAP, Singular and polymake objects.
- For a CA system to be usable for non-specialists, the notation (commands) and behaviour need to follow as closely as possible a standard text-book and to be consistent as far as possible.
- On the other hand, the "specialist" might want to have access to the implementation details and specific algorithms...
- Also: different implementation languages (Singular: C++, Singular; GAP: C, GAP; polymake: C++, perl) force different presentations and choices. We need to integrate these and possibly change them again for Julia ...

Examples (using latest development version)

```
julia> symmetric_group(5)
Sym( [ 1 .. 5 ] )

julia> derived_series(ans)
2-element Vector{PermGroup}:
 Sym( [ 1 .. 5 ] )
 Alt( [ 1 .. 5 ] )
```

```
julia> t = torus()
Abstract simplicial complex of dimension 2 on 7 vertices

julia> describe(fundamental_group(t))
"Z x Z"

julia> f_vector(t)
3-element Vector{Int64}:
  7
 21
 14
```

More: https://bit.ly/OscarDemo2022

The End

Interested? Talk to us!
`https://oscar.computeralgebra.de/community/`

We are hiring! $\rightsquigarrow$ talk to Claus Fieker or me!

Thank you!