OSCAR: Integrating GAP and Julia

Sebastian Gutsche

University of Siegen

South Bend, July 24, 2018

SYMBOLIC TOOLS
Outline

1 Introduction to OSCAR

Gutsche
OSCAR: Integrating GAP and Julia
Outline

1. Introduction to OSCAR
2. GAP-Julia Integration
Outline

1. Introduction to OSCAR
2. GAP-Julia Integration
3. Integration vs. Interfacing
All of this is joint work with

- William Hart (TU Kaiserslautern)
- Thomas Breuer (RWTH Aachen)
- Reimer Behrends (TU Kaiserslautern)
- Max Horn (JLU Gießen)
- Markus Pfeiffer (University of St Andrews)
- ...
All of this is joint work with

- William Hart (TU Kaiserslautern)
- Thomas Breuer (RWTH Aachen)
- Reimer Behrends (TU Kaiserslautern)
- Max Horn (JLU Gießen)
- Markus Pfeiffer (University of St Andrews)
- ...

For a complete list of people involved in the various parts of OSCAR, see https://oscar.computeralgebra.de/credits.
1 Introduction to OSCAR

2 GAP-Julia Integration

3 Integration vs. Interfacing
Overview

Visionary system surpassing the combined capabilities of the underlying systems

**GAP**: computational discrete algebra, group and representation theory, general purpose high level interpreted programming language.

**Singular**: polynomial computations, with emphasis on algebraic geometry, commutative algebra, and singularity theory.

**Examples:**
- Multigraded equivariant Cox rings of toric varieties over number fields
- Graphs of groups in division algebras
- Matrix groups over polynomial rings with coefficients in number fields
- Gröbner fans over fields with discrete valuations

**polymake**: convex polytopes, polyhedral and stacky fans, simplicial complexes and related objects from combinatorics and geometry.

**ANTIC**: number theoretic software featuring computations in and with number fields and generic finitely presented rings.

**Examples:**
- Multigraded equivariant Cox rings of toric varieties over number fields
- Graphs of groups in division algebras
- Matrix groups over polynomial rings with coefficients in number fields
- Gröbner fans over fields with discrete valuations

**Gutsche OSCAR**: Integrating GAP and Julia
Create a new CAS integrating GAP, Singular, polymake, and ANTIC as tight as possible.
The vision for OSCAR

Create a new CAS integrating GAP, Singular, polymake, and ANTIC as tight as possible. This means

- removing the barriers between systems by unifying low-level data structures;
The vision for OSCAR

Create a new CAS integrating GAP, Singular, polymake, and ANTIC as tight as possible. This means

- removing the barriers between systems by unifying low-level data structures;
- make all functionality from each system available in every other system;
The vision for OSCAR

Create a new CAS integrating GAP, Singular, polymake, and ANTIC as tight as possible. This means

- removing the barriers between systems by unifying low-level data structures;
- make all functionality from each system available in every other system;
- make all systems share a common mid-level programming layer.
We use Julia as a powerful mid-level programming layer.
We use Julia as a powerful mid-level programming layer. This includes

- bi-directional interfaces from all systems to Julia, so Julia can be used as a communication layer;
We use Julia as a powerful mid-level programming layer. This includes

- bi-directional interfaces from all systems to Julia, so Julia can be used as a communication layer;
- possibility to extend systems with Julia code, making use of Julia’s powerful JIT-compiler,
We use Julia as a powerful mid-level programming layer. This includes

- bi-directional interfaces from all systems to Julia, so Julia can be used as a communication layer;
- possibility to extend systems with Julia code, making use of Julia’s powerful JIT-compiler, type system,
The role of Julia

We use Julia as a powerful mid-level programming layer. This includes

- bi-directional interfaces from all systems to Julia, so Julia can be used as a communication layer;
- possibility to extend systems with Julia code, making use of Julia’s powerful JIT-compiler, type system, and extensive library.
Current state of the integration

Singular
All kernel functionality is accessible via Singular.jl
Currently in preparation: A Singular interpreter written in Julia, using Singular.jl
Ring data structures implemented in Julia can be used as coefficient rings for polynomials
polymake
Prototype for accessing most polymake functionality from Julia
GAP: Second part of talk
OSCAR: Integrating GAP and Julia
Current state of the integration

- ANTIC is written in Julia
Current state of the integration

- ANTIC is written in Julia
- Singular
Current state of the integration

- ANTIC is written in Julia
- Singular
  - All kernel functionality is accessible via Singular.jl
Current state of the integration

- ANTIC is written in Julia
- Singular
  - All kernel functionality is accessible via Singular.jl
  - Currently in preparation: A Singular interpreter written in Julia, using Singular.jl
Current state

Current state of the integration

- ANTIC is written in Julia
- Singular
  - All kernel functionality is accessible via Singular.jl
  - Currently in preparation: A Singular interpreter written in Julia, using Singular.jl
  - Ring data structures implemented in Julia can be used as coefficient rings for polynomials
Current state

Current state of the integration

- ANTIC is written in Julia
- Singular
  - All kernel functionality is accessible via Singular.jl
  - Currently in preparation: A Singular interpreter written in Julia, using Singular.jl
  - Ring data structures implemented in Julia can be used as coefficient rings for polynomials
- polymake
Current state

Current state of the integration

- ANTiC is written in Julia
- Singular
  - All kernel functionality is accessible via Singular.jl
  - Currently in preparation: A Singular interpreter written in Julia, using Singular.jl
  - Ring data structures implemented in Julia can be used as coefficient rings for polynomials
- polymake
  - Prototype for accessing most polymake functionality from Julia
Current state

Current state of the integration

- ANTIC is written in Julia
- Singular
  - All kernel functionality is accessible via Singular.jl
  - Currently in preparation: A Singular interpreter written in Julia, using Singular.jl
  - Ring data structures implemented in Julia can be used as coefficient rings for polynomials
- polymake
  - Prototype for accessing most polymake functionality from Julia
- GAP: Second part of talk
Example: Using Singular with Nemo rings

Example for using Nemo number fields as coefficient rings in Singular

Julia_rings_with_Singular.ipynb
All information about the OSCAR project can be found on

https://oscar.computeralgebra.de
All information about the OSCAR project can be found on

https://oscar.computeralgebra.de

On the page you can find

- news,
- blog posts,
- examples,
- and installation instructions.
1. Introduction to OSCAR

2. GAP-Julia Integration

3. Integration vs. Interfacing
JuliaInterface and GAP.jl

GAP package JuliaInterface and Julia module GAP.jl

Conversion of basic data types (e.g., integers, lists, permutations) between GAP and Julia
Use of GAP data types in Julia and Julia data types in GAP
Use of Julia functions in GAP and GAP functions in Julia
Possibility to add compiled Julia functions as kernel functions to GAP

https://github.com/oscar-system

Gutsche

OSCAR: Integrating GAP and Julia
JuliaInterface and GAP.jl provide conversion of basic data types (e.g., integers, lists, permutations) between GAP and Julia, use of GAP data types in Julia and Julia data types in GAP, use of Julia functions in GAP and GAP functions in Julia, and the possibility to add compiled Julia functions as kernel functions to GAP.

https://github.com/oscar-system

GAP ↔ Julia
GAP package JuliaInterface and Julia module GAP.jl

GAP $\leftrightarrow$ Julia

JuliaInterface and GAP.jl provide
JuliaInterface and GAP.jl provide

Conversion of basic data types (e.g., integers, lists, permutations) between GAP and Julia
JuliaInterface and GAP.jl provide

- Conversion of basic data types (e.g., integers, lists, permutations) between GAP and Julia
- Use of GAP data types in Julia and Julia data types in GAP

GAP ↔ Julia
JuliaInterface and GAP.jl provide

- Conversion of basic data types (e.g., integers, lists, permutations) between GAP and Julia
- Use of GAP data types in Julia and Julia data types in GAP
- Use of Julia functions in GAP and GAP functions in Julia

GAP ↔ Julia
JuliaInterface and GAP.jl provide

- Conversion of basic data types (e.g., integers, lists, permutations) between GAP and Julia
- Use of GAP data types in Julia and Julia data types in GAP
- Use of Julia functions in GAP and GAP functions in Julia
- Possibility to add compiled Julia functions as kernel functions to GAP
JuliaInterface and GAP.jl

GAP package JuliaInterface and Julia module GAP.jl

GAP ↔ Julia

JuliaInterface and GAP.jl provide

- Conversion of basic data types (e.g., integers, lists, permutations) between GAP and Julia
- Use of GAP data types in Julia and Julia data types in GAP
- Use of Julia functions in GAP and GAP functions in Julia
- Possibility to add compiled Julia functions as kernel functions to GAP

https://github.com/oscar-system
JuliaInterface contains GAP data structures that can hold pointers to Julia objects:
JuliaInterface contains GAP data structures that can hold pointers to Julia objects:

gap> a := 2;
2
JuliaInterface contains GAP data structures that can hold pointers to Julia objects:

```gap
gap> a := 2;
2

gap> b := ConvertedToJulia( a );
<Julia: 2>
```
JuliaInterface data structures: Objects

JuliaInterface contains GAP data structures that can hold pointers to Julia objects:

```gap
gap> a := 2;
2

gap> b := ConvertedToJulia( a );
<Julia: 2>

gap> ConvertedFromJulia( b );
2
```
JuliaInterface contains GAP data structures that can hold pointers to Julia objects:

\[\text{gap: a := 2;}\]
2

\[\text{gap: b := ConvertedToJulia(a);}\]
<Julia: 2>

\[\text{gap: ConvertedFromJulia(b);}\]
2

Possible conversions:

- Integers
JuliaInterface data structures: Objects

JuliaInterface contains GAP data structures that can hold pointers to Julia objects:

\begin{verbatim}
gap> a := 2;
2

gap> b := ConvertedToJulia( a );
<Julia: 2>

gap> ConvertedFromJulia( b );
2
\end{verbatim}

Possible conversions:

- Integers
- Floats
JuliaInterface contains GAP data structures that can hold pointers to Julia objects:

```gap
gap> a := 2;
2

gap> b := ConvertedToJulia( a );
<Julia: 2>

gap> ConvertedFromJulia( b );
2
```

Possible conversions:
- Integers
- Floats
- Strings
JuliaInterface contains GAP data structures that can hold pointers to Julia objects:

```gap
gap> a := 2;
2

gap> b := ConvertedToJulia( a );
<Julia: 2>

gap> ConvertedFromJulia( b );
2
```

Possible conversions:

- Integers
- Floats
- Strings
- Booleans
JuliaInterface data structures: Objects

JuliaInterface contains GAP data structures that can hold pointers to Julia objects:

```
gap> a := 2;
2

gap> b := ConvertedToJulia( a );
<Julia: 2>

gap> ConvertedFromJulia( b );
2
```

Possible conversions:

- Integers
- Floats
- Strings
- Booleans
- Nested lists of the above to Arrays or Tuples
JuliaInterface provides the possibility to call Julia functions by converting GAP objects:
JuliaInterface provides the possibility to call Julia functions by converting GAP objects:

```
gap> ImportJuliaModuleIntoGAP( "Base" );
```

```text
<Julia: 2.0>
```
JuliaInterface provides the possibility to call Julia functions by converting GAP objects:

\begin{verbatim}
gap> ImportJuliaModuleIntoGAP( "Base" );

gap> Julia.Base.sqrt( 4 );
<Julia: 2.0>
\end{verbatim}
JuliaInterface provides the possibility to call Julia functions by converting GAP objects:

```
gap> ImportJuliaModuleIntoGAP( "Base" );

gap> Julia.Base.sqrt( 4 );
<Julia: 2.0>
```

- Julia functions can be used like GAP functions
JuliaInterface provides the possibility to call Julia functions by converting GAP objects:

\begin{verbatim}
gap> ImportJuliaModuleIntoGAP( "Base" );

gap> Julia.Base.sqrt( 4 );
<Julia: 2.0>
\end{verbatim}

- Julia functions can be used like GAP functions
- If necessary and possible, input data is converted to Julia
JuliaInterface provides the possibility to call Julia functions by converting GAP objects:

```gap
gap> ImportJuliaModuleIntoGAP( "Base" );

gap> Julia.Base.sqrt( 4 );
<Julia: 2.0>
```

- Julia functions can be used like GAP functions
- If necessary and possible, input data is converted to Julia
- Calling only possible for convertible types and Julia objects
Using JuliaInterface, it is possible to write Julia functions and use them as GAP kernel functions (from orbits.jl):

```julia
function orbit( element, generators, action )
    work_set = [ element ]
    return_set = [ element ]
    generator_length = gap_LengthPlist(generators)
    while length(work_set) != 0
        current_element = pop!(work_set)
        for current_generator_number = 1:generator_length
            current_generator = gap_ListElement(generators, current_generator_number)
            current_result = gap_CallFunc2Args(action, current_element, current_generator)
            is_in_set = false
            for i in return_set
                if i == current_result
                    is_in_set = true
                    break
                end
            end
            if !is_in_set
                push!(work_set, current_result)
                push!(return_set, current_result)
            end
        end
    end
    return return_set
end
```
Using JuliaInterface, it is possible to write Julia functions and use them as GAP kernel functions (from orbits.jl):

```julia
function orbit( element, generators, action )
    work_set = [ element ]
    return_set = [ element ]
generator_length = gap_LengthPlist(generators)
while length(work_set) != 0
    current_element = pop!(work_set)
    for current_generator_number = 1:generator_length
        current_generator = gap_ListElement(generators,
                                            current_generator_number)
        current_result = gap_CallFunc2Args(action,
                                             current_element,
                                             current_generator)
        is_in_set = false
        for i in return_set
            if i == current_result
                is_in_set = true
                break
            end
        end
        if ! is_in_set
            push!( work_set, current_result )
            push!( return_set, current_result )
        end
    end
end
return return_set
end
```
Using JuliaInterface, it is possible to write Julia functions and use them as GAP kernel functions:
Using JuliaInterface, it is possible to write Julia functions and use them as GAP kernel functions:

```gap
gap> JuliaIncludeFile( "orbits.jl" );
gap> JuliaBindCFunction( "orbit", "orbit_jl", 3 );
```
Using JuliaInterface, it is possible to write Julia functions and use them as GAP kernel functions:

```gap
gap> JuliaIncludeFile( "orbits.jl" );
gap> JuliaBindCFunction( "orbit", "orbit_jl", 3 );
```

Compiled Julia functions come close to the performance of kernel functions:
Using JuliaInterface, it is possible to write Julia functions and use them as GAP kernel functions:

```
gap> JuliaIncludeFile( "orbits.jl" );
gap> JuliaBindCFunction( "orbit", "orbit_jl", 3 );
```

Compiled Julia functions come close to the performance of kernel functions:

```
gap> S := GeneratorsOfGroup( SymmetricGroup( 10000 ) );;
```
Using JuliaInterface, it is possible to write Julia functions and use them as GAP kernel functions:

```gap
gap> JuliaIncludeFile( "orbits.jl" );
gap> JuliaBindCFunction( "orbit", "orbit_jl", 3 );
```

Compiled Julia functions come close to the performance of kernel functions:

```gap
gap> S := GeneratorsOfGroup( SymmetricGroup( 10000 ) );;
gap> orbit_gap( 1, S, OnPoints );; time;
5769
```
Using JuliaInterface, it is possible to write Julia functions and use them as GAP kernel functions:

```gap
gap> JuliaIncludeFile( "orbits.jl" );
gap> JuliaBindCFunction( "orbit", "orbit_jl", 3 );
```

Compiled Julia functions come close to the performance of kernel functions:

```gap
gap> S := GeneratorsOfGroup( SymmetricGroup( 10000 ) );;

gap> orbit_gap( 1, S, OnPoints );; time;
5769

gap> orbit_jl( 1, S, OnPoints );; time;
84
```
Using JuliaInterface, it is possible to write Julia functions and use them as GAP kernel functions:

```gap
JuliaIncludeFile( "orbits.jl" );
JuliaBindCFunction( "orbit", "orbit_jl", 3 );
```

Compiled Julia functions come close to the performance of kernel functions:

```gap
S := GeneratorsOfGroup( SymmetricGroup( 10000 ) );

orbit_gap( 1, S, OnPoints );; time;
5769

orbit_jl( 1, S, OnPoints );; time;
84

orbit_c( 1, S, OnPoints );; time;
46
```
The Julia module GAP.jl provides access to GAP’s data structures and functions from Julia.
The Julia module GAP.jl provides access to GAP’s data structures and functions from Julia

```julia
julia> S3 = GAP.SymmetricGroup( LibGAP.to_gap( 3 ) )
GAP: SymmetricGroup( [ 1 .. 3 ] )
```
The Julia module GAP.jl provides access to GAP’s data structures and functions from Julia

```julia
julia> S3 = GAP.SymmetricGroup( LibGAP.to_gap( 3 ) )
GAP: SymmetricGroup( [ 1 .. 3 ] )

julia> size_gap = GAP.Size( S3 )
GAP: 6
```
The Julia module GAP.jl provides access to GAP’s data structures and functions from Julia

```julia
julia> S3 = GAP.SymmetricGroup( LibGAP.to_gap( 3 ) )
GAP: SymmetricGroup( [ 1 .. 3 ] )

julia> size_gap = GAP.Size( S3 )
GAP: 6

julia> LibGAP.from_gap( size_gap, Int64 )
6
```
From the GAP side

How does GAP benefit from Julia/OSCAR (except mathematical algorithms)?
From the GAP side

How does GAP benefit from Julia/OSCAR (except mathematical algorithms)?

**Speedup**
From the GAP side

How does GAP benefit from Julia/OSCAR (except mathematical algorithms)?

**Speedup**
- Now: Find time critical parts of algorithms, rewrite them in C.
From the GAP side

How does GAP benefit from Julia/OSCAR (except mathematical algorithms)?

**Speedup**
- Now: Find time critical parts of algorithms, rewrite them in C.
- Future: Find time critical parts of algorithms, rewrite them in Julia.
From the GAP side

How does GAP benefit from Julia/OSCAR (except mathematical algorithms)?

**Speedup**

- Now: Find time critical parts of algorithms, rewrite them in C.
- Future: Find time critical parts of algorithms, rewrite them in Julia.

Benefits:
From the GAP side

How does GAP benefit from Julia/OSCAR (except mathematical algorithms)?

<table>
<thead>
<tr>
<th>Speedup</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Now:</strong> Find time critical parts of algorithms, rewrite them in C.</td>
<td></td>
</tr>
<tr>
<td><strong>Future:</strong> Find time critical parts of algorithms, rewrite them in Julia.</td>
<td></td>
</tr>
</tbody>
</table>

Benefits:
- Higher level language, which may be easier to use than C
How does GAP benefit from Julia/OSCAR (except mathematical algorithms)?

**Speedup**
- Now: Find time critical parts of algorithms, rewrite them in C.
- Future: Find time critical parts of algorithms, rewrite them in Julia.

**Benefits:**
- Higher level language, which may be easier to use than C
- Extensive functionality available in standard modules
How does GAP benefit from Julia/OSCAR (except mathematical algorithms)?

**Speedup**
- Now: Find time critical parts of algorithms, rewrite them in C.
- Future: Find time critical parts of algorithms, rewrite them in Julia.

**Benefits:**
- Higher level language, which may be easier to use than C
- Extensive functionality available in standard modules
How does OSCAR benefit from GAP (except mathematical algorithms)?
How does OSCAR benefit from GAP (except mathematical algorithms)?

Language features
From the OSCAR side

How does OSCAR benefit from GAP (except mathematical algorithms)?

Language features

- Flexible type system: Objects can learn about themselves
How does OSCAR benefit from GAP (except mathematical algorithms)?

<table>
<thead>
<tr>
<th>Language features</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Flexible type system: Objects can learn about themselves</td>
</tr>
<tr>
<td>- Built-in traits: Known properties of objects decide which variant of an algorithm to use</td>
</tr>
</tbody>
</table>
How does OSCAR benefit from GAP (except mathematical algorithms)?

**Language features**

- **Flexible type system**: Objects can learn about themselves
- **Built-in traits**: Known properties of objects decide which variant of an algorithm to use
- **Immediate propagation**: Second execution layer is used to spread properties between objects
From the OSCAR side

How does OSCAR benefit from GAP (except mathematical algorithms)?

**Language features**

- Flexible type system: Objects can learn about themselves
- Built-in traits: Known properties of objects decide which variant of an algorithm to use
- Immediate propagation: Second execution layer is used to spread properties between objects
- Categorical programming language as defined in the CAP project
Example: Using Singular in GAP via Julia

Example for using Singular in GAP via Julia

Using Singular from GAP.ipynb
Outline

1. Introduction to OSCAR
2. GAP-Julia Integration
3. Integration vs. Interfacing
Integration vs. Interfacing

Problems when interfacing two garbage collected systems

Primitive approach: System A holds a list of objects otherwise only referred to by System B, and vice versa. This approach can be implemented using build-in techniques in GAP and Julia, but it adds a layer of indirections and causes inefficiencies and unreachable cycles that involve both GAP and Julia objects cannot be reclaimed, so it leads to memory leaks.
Problems when interfacing two garbage collected systems

Primitive approach: System A holds a list of objects otherwise only referred to by System B, and vice versa.
Integration vs. Interfacing

Problems when interfacing two garbage collected systems

Primitive approach: System A holds a list of objects otherwise only referred to by System B, and vice versa.

- This approach can be implemented using build-in techniques in GAP and Julia, but
Problems when interfacing two garbage collected systems

Primitive approach: System A holds a list of objects otherwise only referred to by System B, and vice versa.

- This approach can be implemented using build-in techniques in GAP and Julia, but
- it adds a layer of indirections and causes inefficiencies and
Integration vs. Interfacing

Problems when interfacing two garbage collected systems

Primitive approach: System A holds a list of objects otherwise only referred to by System B, and vice versa.

- This approach can be implemented using built-in techniques in GAP and Julia, but
- it adds a layer of indirections and causes inefficiencies and
- unreachable cycles that involve both GAP and Julia objects cannot be reclaimed, so it leads to memory leaks.
Q: How does OSCAR’s Julia-GAP integration differ from classical interfacing?

Using the same GC for GAP and Julia

Changes to GAP and Julia to make it possible to use Julia’s GC simultaneously for GAP and Julia (Behrends/Horn)

Changes to Julia accepted, will be part of 1.1

Changes to GAP accepted, will be part of 4.10 (TBR in December)

This way, all GAP objects are first-class citizens in Julia, and Julia objects are first-class citizens in GAP

Thus using Julia objects from GAP and GAP objects from Julia works without any GC overhead (essentially no penalty at all)
Q: How does OSCAR’s Julia-GAP integration differ from classical interfacing?

Using the same GC for GAP and Julia

Thus using Julia objects from GAP and GAP objects from Julia works without any GC overhead (essentially no penalty at all)
Q: How does OSCAR’s Julia-GAP integration differ from classical interfacing?

**Using the same GC for GAP and Julia**

- Changes to GAP and Julia to make it possible to use Julia’s GC simultaneously for GAP and Julia (Behrends/Horn)
Q: How does OSCAR’s Julia-GAP integration differ from classical interfacing?

Using the same GC for GAP and Julia

- Changes to GAP and Julia to make it possible to use Julia’s GC simultaneously for GAP and Julia (Behrends/Horn)
  - Changes to Julia accepted, will be part of 1.1
Q: How does OSCAR’s Julia-GAP integration differ from classical interfacing?

Using the same GC for GAP and Julia

- Changes to GAP and Julia to make it possible to use Julia’s GC simultaneously for GAP and Julia (Behrends/Horn)
  - Changes to Julia accepted, will be part of 1.1
  - Changes to GAP accepted, will be part of 4.10 (TBR in December)
Q: How does OSCAR’s Julia-GAP integration differ from classical interfacing?

Using the same GC for GAP and Julia

- Changes to GAP and Julia to make it possible to use Julia’s GC simultaneously for GAP and Julia (Behrends/Horn)
  - Changes to Julia accepted, will be part of 1.1
  - Changes to GAP accepted, will be part of 4.10 (TBR in December)

- This way, all GAP objects are first-class citizens in Julia, and Julia objects are first class citizens in GAP
Q: How does OSCAR’s Julia-GAP integration differ from classical interfacing?

**Using the same GC for GAP and Julia**

- Changes to GAP and Julia to make it possible to use Julia’s GC simultaneously for GAP and Julia (Behrends/Horn)
  - Changes to Julia accepted, will be part of 1.1
  - Changes to GAP accepted, will be part of 4.10 (TBR in December)
- This way, all GAP objects are first-class citizens in Julia, and Julia objects are first class citizens in GAP
- Thus using Julia objects from GAP and GAP objects from Julia works without any GC overhead (essentially no penalty at all)