OSCAR: Integrating GAP and Julia

Sebastian Gutsche

University of Siegen

South Bend, July 24, 2018

Outline

Outline

Outline

People

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)
- ...

People

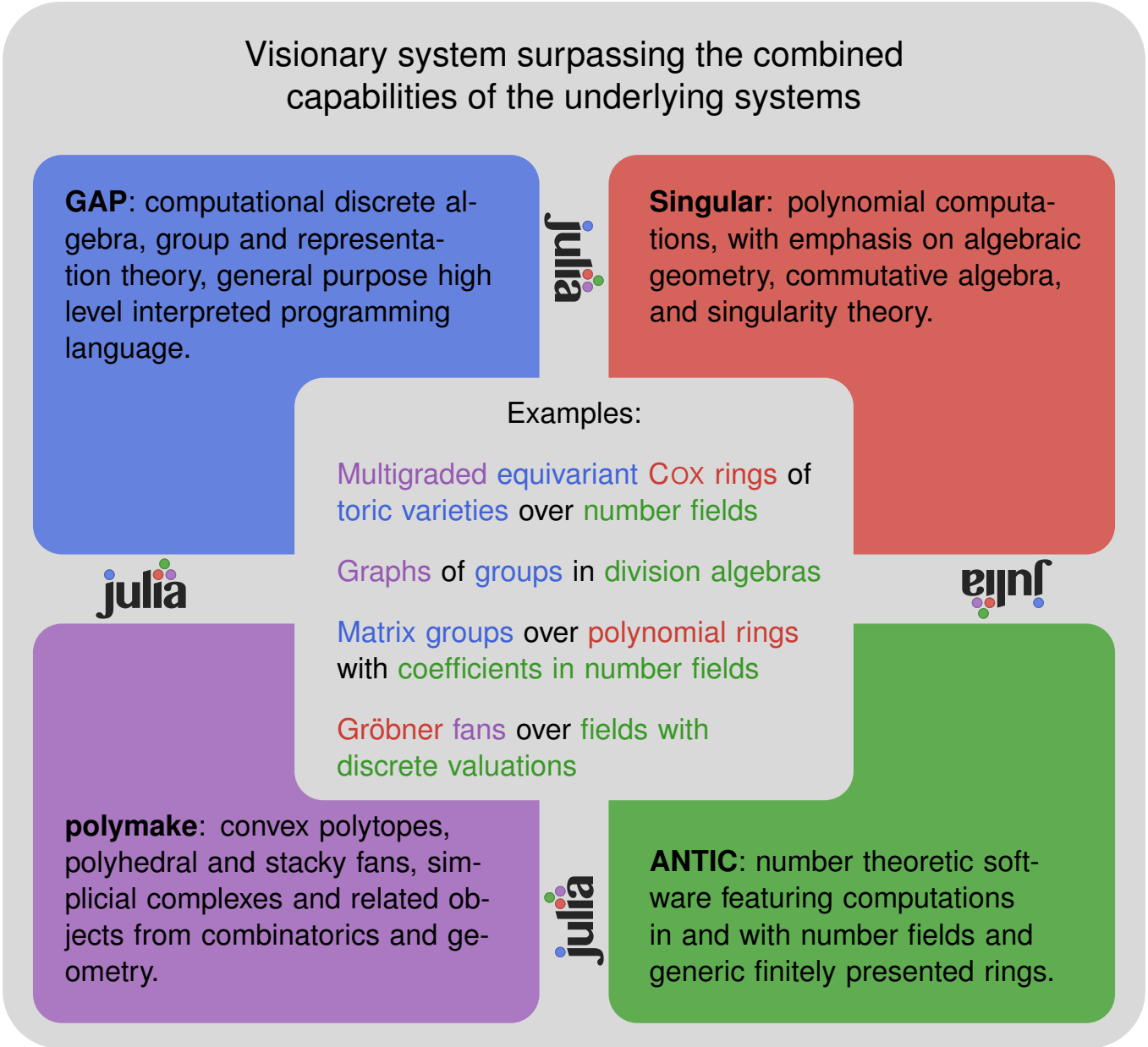
All of this is joint work with

- 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.

Outline

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.
julia
julia
julia
julia

The vision for OSCAR

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.

The role of Julia

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

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;

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,

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,

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

Current state of the integration

Current state

Current state of the integration

- ANTIC is written in Julia

Current state

Current state of the integration

- ANTIC is written in Julia
- Singular

Current state

Current state of the integration

- ANTIC is written in Julia
- Singular
  - All kernel functionality is accessible via 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

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`

Resources

All information about the OSCAR project can be found on

https://oscar.computeralgebra.de

Resources

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.

Outline

JuliaInterface and GAP.jl

GAP package JuliaInterface and Julia module GAP.jl

GAP package JuliaInterface and Julia module GAP.jl

GAP $\longleftrightarrow$ Julia

GAP package JuliaInterface and Julia module GAP.jl

$$\text{GAP} \longleftrightarrow \text{Julia}$$

JuliaInterface and GAP.jl provide

GAP package JuliaInterface and Julia module GAP.jl

$$\text{GAP} \longleftrightarrow \text{Julia}$$

JuliaInterface and GAP.jl provide

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

GAP package JuliaInterface and Julia module GAP.jl

$$\text{GAP} \longleftrightarrow \text{Julia}$$

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 package JuliaInterface and Julia module GAP.jl

$$\text{GAP} \longleftrightarrow \text{Julia}$$

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 package JuliaInterface and Julia module GAP.jl

$$\text{GAP} \longleftrightarrow \text{Julia}$$

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

GAP package JuliaInterface and Julia module GAP.jl

$$\text{GAP} \longleftrightarrow \text{Julia}$$

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 data structures: Objects

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

JuliaInterface data structures: Objects

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

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

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>
```

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
```

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

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

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

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

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 data structures: Functions

JuliaInterface provides the possibility to call Julia functions by converting GAP objects:

JuliaInterface data structures: Functions

JuliaInterface provides the possibility to call Julia functions by converting GAP objects:

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

JuliaInterface data structures: Functions

JuliaInterface provides the possibility to call Julia functions by converting GAP objects:

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

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

JuliaInterface data structures: Functions

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 data structures: Functions

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
- If necessary and possible, input data is converted to Julia

JuliaInterface data structures: Functions

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
- If necessary and possible, input data is converted to Julia
- Calling only possible for convertible types and Julia objects

JuliaInterface: Julia functions as kernel modules

Using JuliaInterface, it is possible to write Julia functions and use them as GAP kernel functions (from orbits.jl):

JuliaInterface: Julia functions as kernel modules

Using JuliaInterface, it is possible to write Julia functions and use them as GAP kernel functions (from orbits.jl):

```
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
```

JuliaInterface: Julia functions as kernel modules

Using JuliaInterface, it is possible to write Julia functions and use them as GAP kernel functions:

JuliaInterface: Julia functions as kernel modules

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 );
```

JuliaInterface: Julia functions as kernel modules

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:

JuliaInterface: Julia functions as kernel modules

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 ) );;
```

JuliaInterface: Julia functions as kernel modules

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 ) );;

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

JuliaInterface: Julia functions as kernel modules

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 ) );;

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

gap> orbit_jl( 1, S, OnPoints );; time;
84
```

JuliaInterface: Julia functions as kernel modules

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 ) );;

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

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

gap> orbit_c( 1, S, OnPoints );; time;
46
```

GAP.jl: using GAP from Julia

The Julia module GAP.jl provides access to GAP's data structures and functions from Julia

GAP.jl: using GAP from Julia

The Julia module GAP.jl provides access to GAP's data structures and functions from Julia

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

GAP.jl: using GAP from Julia

The Julia module GAP.jl provides access to GAP's data structures and functions from Julia

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

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

GAP.jl: using GAP from Julia

The Julia module GAP.jl provides access to GAP's data structures and functions from 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)?

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

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:

- Higher level language, which may be easier to use than C
- Extensive functionality available in standard modules

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:

- Higher level language, which may be easier to use than C
- Extensive functionality available in standard modules

From the OSCAR side

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

From the OSCAR side

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

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

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

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

Integration vs. Interfacing

Problems when interfacing two garbage collected systems

Integration vs. Interfacing

Problems when interfacing two garbage collected systems

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

- This approach can be implemented using build-in techniques in GAP and Julia, but

Integration vs. Interfacing

Problems when interfacing two garbage collected systems

Primitive approach: System A holds a list of objects otherwise only refered 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 refered 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.

Integration vs. Interfacing

Q: How does OSCAR's Julia-GAP integration differ from classical interfacing?

Integration vs. Interfacing

Q: How does OSCAR's Julia-GAP integration differ from classical interfacing?

Using the same GC for GAP and Julia

Integration vs. Interfacing

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)

Integration vs. Interfacing

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

Integration vs. Interfacing

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)

Integration vs. Interfacing

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

Integration vs. Interfacing

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)

Acknowledgements

TR 195
SYMBOLIC TOOLS

OPEN
DREAMKIT