OSCAR: A visionary, new computer algebra system

Wolfram Decker, William Hart,
Sebastian Gutsche, Michael Joswig

December 11, 2017

Two Coordinated DFG-Programmes

Algorithmic and Experimental Methods

in Algebra, Geometry, and Number Theory

DFG Priority Project SPP 1489

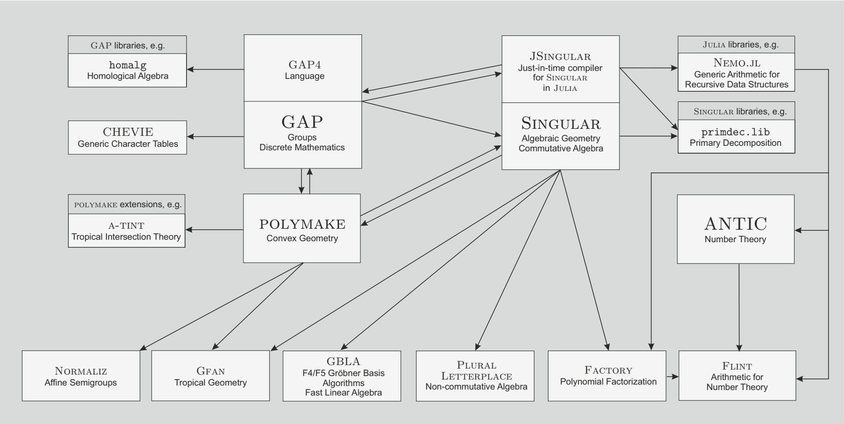
2010–2016

Collaborative Research Center Transregio TRR 195

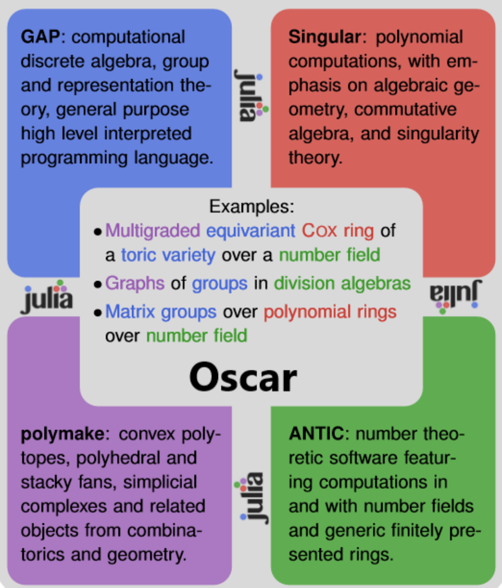
Symbolic Tools in Mathematics and their Application

2017–

Software Development Within SPP 1489



Software Development Within TRR 195



Software Development Within TRR 195: OSCAR

Central Tasks:

- Integrate all computer algebra systems, libraries and packages developed within the TRR 195 into OSCAR which will surpass the combined mathematical capabilities of the underlying systems.

Software Development Within TRR 195: OSCAR

Central Tasks:

- Integrate all computer algebra systems, libraries and packages developed within the TRR 195 into OSCAR which will surpass the combined mathematical capabilities of the underlying systems.

  Where do we stand: singular.jl, gap.jl (more in this talk)

Software Development Within TRR 195: OSCAR

Central Tasks:

- Integrate all computer algebra systems, libraries and packages developed within the TRR 195 into OSCAR which will surpass the combined mathematical capabilities of the underlying systems.

  Where do we stand: singular.jl, gap.jl (more in this talk)

  Experts among participants: Reimer Behrends, Thomas Breuer, Sebastian Gutsche, Bill Hart

Software Development Within TRR 195: OSCAR

Central Tasks:

- Integrate all computer algebra systems, libraries and packages developed within the TRR 195 into OSCAR which will surpass the combined mathematical capabilities of the underlying systems.

  Where do we stand: singular.jl, gap.jl (more in this talk)

  Experts among participants: Reimer Behrends, Thomas Breuer, Sebastian Gutsche, Bill Hart
- Boost the performance of OSCAR to a new level by parallelisation.

Software Development Within TRR 195: OSCAR

Central Tasks:

- Integrate all computer algebra systems, libraries and packages developed within the TRR 195 into OSCAR which will surpass the combined mathematical capabilities of the underlying systems.

  Where do we stand: singular.jl, gap.jl (more in this talk)

  Experts among participants: Reimer Behrends, Thomas Breuer, Sebastian Gutsche, Bill Hart

- Boost the performance of OSCAR to a new level by parallelisation.

  Where do we stand: HPC-GAP, framework for coarse grained parallelization in Singular, experimental framework for fine grained parallelization in Singular;

Software Development Within TRR 195: OSCAR

Central Tasks:

- Integrate all computer algebra systems, libraries and packages developed within the TRR 195 into OSCAR which will surpass the combined mathematical capabilities of the underlying systems.

  Where do we stand: singular.jl, gap.jl (more in this talk)

  Experts among participants: Reimer Behrends, Thomas Breuer, Sebastian Gutsche, Bill Hart

- Boost the performance of OSCAR to a new level by parallelisation.

  Where do we stand: HPC-GAP, framework for coarse grained parallelization in Singular, experimental framework for fine grained parallelization in Singular; massive parallelization via GPI-Space (Fraunhofer ITWM Kaiserslautern, using Petri nets)

  Experts among participants: Reimer Behrends, Michael Joswig, Andreas Steenpass; see talk by Janko Böhm

Software Development Within TRR 195: OSCAR

Central Tasks:

- Integrate all computer algebra systems, libraries and packages developed within the TRR 195 into OSCAR which will surpass the combined mathematical capabilities of the underlying systems.

  Where do we stand: singular.jl, gap.jl (more in this talk)

  Experts among participants: Reimer Behrends, Thomas Breuer, Sebastian Gutsche, Bill Hart
- Boost the performance of OSCAR to a new level by parallelisation.

  Where do we stand: HPC-GAP, framework for coarse grained parallelization in Singular, experimental framework for fine grained parallelization in Singular; massive parallelization via GPI-Space (Fraunhofer ITWM Kaiserslautern, using Petri nets)

  Experts among participants: Reimer Behrends, Michael Joswig, Andreas Steenpass; see talk by Janko Böhm
- Create a central infrastructure for mathematical data.

Software Development Within TRR 195: OSCAR

Central Tasks:

- Integrate all computer algebra systems, libraries and packages developed within the TRR 195 into OSCAR which will surpass the combined mathematical capabilities of the underlying systems.

  Where do we stand: singular.jl, gap.jl (more in this talk)

  Experts among participants: Reimer Behrends, Thomas Breuer, Sebastian Gutsche, Bill Hart
- Boost the performance of OSCAR to a new level by parallelisation.

  Where do we stand: HPC-GAP, framework for coarse grained parallelization in Singular, experimental framework for fine grained parallelization in Singular; massive parallelization via GPI-Space (Fraunhofer ITWM Kaiserslautern, using Petri nets)

  Experts among participants: Reimer Behrends, Michael Joswig, Andreas Steenpass; see talk by Janko Böhm
- Create a central infrastructure for mathematical data.

Numerous small steps are needed to build OSCAR. Guiding principles:

Numerous small steps are needed to build OSCAR. Guiding principles:

- Take mathematical problems within TRR195 and international community into account.

Software Development Within TRR 195: OSCAR

Numerous small steps are needed to build OSCAR. Guiding principles:

- Take mathematical problems within TRR195 and international community into account.
- Most steps should be of immediate benefit for users (of current systems and OSCAR).

Software Development Within TRR 195: OSCAR

Numerous small steps are needed to build OSCAR. Guiding principles:

- Take mathematical problems within TRR195 and international community into account.
- Most steps should be of immediate benefit for users (of current systems and OSCAR).
- Rely on existing resources where possible (e.g. Julia).

Example for Immediate Benefits: Singular.jl

Julia access to Singular KERNEL functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\mathrm{GF}(p)$, etc.

Example for Immediate Benefits: Singular.jl

Julia access to Singular KERNEL functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, GF(p), etc.
- Polynomials, ideals, modules, matrices, etc.

Example for Immediate Benefits: Singular.jl

Julia access to Singular KERNEL functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\mathrm{GF}(p)$, etc.
- Polynomials, ideals, modules, matrices, etc.
- Gröbner bases, syzygies, free resolutions

Example for Immediate Benefits: Singular.jl

Julia access to Singular KERNEL functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, GF(p), etc.
- Polynomials, ideals, modules, matrices, etc.
- Gröbner bases, syzygies, free resolutions

Integration with number theory components (Experts among participants: Claus Fieker, Bill Hart, Tommy Hofman):

Example for Immediate Benefits: Singular.jl

Julia access to Singular KERNEL functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\mathrm{GF}(p)$, etc.
- Polynomials, ideals, modules, matrices, etc.
- Gröbner bases, syzygies, free resolutions

Integration with number theory components (Experts among participants: Claus Fieker, Bill Hart, Tommy Hofman):

- Singular polynomials over optimized coefficient rings, e.g. Gröbner bases over cyclotomic fields

Example for Immediate Benefits: Singular.jl

Julia access to Singular KERNEL functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\mathrm{GF}(p)$, etc.
- Polynomials, ideals, modules, matrices, etc.
- Gröbner bases, syzygies, free resolutions

Integration with number theory components (Experts among participants: Claus Fieker, Bill Hart, Tommy Hofman):

- Singular polynomials over optimized coefficient rings, e.g. Gröbner bases over cyclotomic fields
- Plenty of optimized basic functionality (e.g. linear algebra)

Example for Immediate Benefits: Singular.jl

Julia access to Singular KERNEL functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\mathrm{GF}(p)$, etc.
- Polynomials, ideals, modules, matrices, etc.
- Gröbner bases, syzygies, free resolutions

Integration with number theory components (Experts among participants: Claus Fieker, Bill Hart, Tommy Hofman):

- Singular polynomials over optimized coefficient rings, e.g. Gröbner bases over cyclotomic fields
- Plenty of optimized basic functionality (e.g. linear algebra)

Author of Singular.jl:

- Bill Hart
- Oleksandr Motsak

Example for Immediate Benefits: Singular.jl
Primary Decomposition of Binomial Ideals

Some History

- David Eisenbud and Bernd Sturmfels: *Binomial Ideals*, 1996
- Thomas Kahle: Macaulay2 package `Binomials.M2`, 2010
- Clara Petroll: Bachelor thesis, 2017

Example for Immediate Benefits: Singular.jl
Primary Decomposition of Binomial Ideals

Some History

- David Eisenbud and Bernd Sturmfels: *Binomial Ideals*, 1996
- Thomas Kahle: Macaulay2 package `Binomials.M2`, 2010
- Clara Petroll: Bachelor thesis, 2017

Example (Singular functions for binomial ideals)

Consider pure binomial ideal in three variables:

$$I = \langle x - y, x^3 - 1, zy^2 - z \rangle \subset \mathbb{C}[x, y, z].$$

Example for Immediate Benefits: Singular.jl
Primary Decomposition of Binomial Ideals

Example (Singular functions for binomial ideals)

```
julia  >  R,(x,y,z) = Singular.PolynomialRing(QabField(), ["x","y","z"])
julia  >  I = Ideal(R,x-y,x^3-1,z*y^2-z)
julia  >  isCellular(I)
(false,3)
julia  >  bcd=cellularDecomp(I)
2-element Array{Singular.sideal,1}:
julia  >  Singular.intersection(bcd[1], bcd[2])==I
true
julia  >  binomialPrimaryDecomposition(I)
3-element Array{Any,1}:
Singular Ideal over Singular Polynomial Ring (Coeffs(18)),(x,y,z),(dp(3),C)
with generators (y+(-1 in Q(z_1)), x+(-1 in Q(z_1)))
Singular Ideal over Singular Polynomial Ring (Coeffs(18)),(x,y,z),(dp(3),C)
with generators (z, y+(-z_3 in Q(z_3)), x+(-z_3 in Q(z_3)))
Singular Ideal over Singular Polynomial Ring (Coeffs(18)),(x,y,z),(dp(3),C)
with generators (z, y+(z_3+1 in Q(z_3)), x+(z_3+1 in Q(z_3)))
```

Immediate Benefits: The Next Step for Singular

Rewrite the Singular Interpreter in Julia

Immediate Benefits: The Next Step for Singular

Rewrite the Singular Interpreter in Julia

Benefits:

- Speed-up due to Just-In-Time compilation;

Immediate Benefits: The Next Step for Singular

Rewrite the Singular Interpreter in Julia

Benefits:

- Speed-up due to Just-In-Time compilation;
- more expressive user language;

Immediate Benefits: The Next Step for Singular

Rewrite the Singular Interpreter in Julia

Benefits:

- Speed-up due to Just-In-Time compilation;
- more expressive user language;
- a wealth of Julia features can be used

julia

- JIT compilation : near C performance.
- Designed by mathematically minded people.
- Open Source (MIT License).
- Actively developed since 2009.
- Supports Windows, OSX, Linux, BSD.
- Friendly C/Python-like (imperative) syntax.

Nemo.jl:

- Flint : polynomials and matrices over $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/p\mathbb{Z}$, $\mathbb{F}_q$, $\mathbb{Q}_p$

Nemo.jl:

- Flint : polynomials and matrices over $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/p\mathbb{Z}$, $\mathbb{F}_q$, $\mathbb{Q}_p$
- Arb : ball arithmetic, univariate polys and matrices over $\mathbb{R}$ and $\mathbb{C}$, special and transcendental functions

Nemo.jl:

- Flint : polynomials and matrices over $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/p\mathbb{Z}$, $\mathbb{F}_q$, $\mathbb{Q}_p$
- Arb : ball arithmetic, univariate polys and matrices over $\mathbb{R}$ and $\mathbb{C}$, special and transcendental functions
- Antic : element arithmetic over absolute number fields

Nemo.jl:

- Flint : polynomials and matrices over $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/p\mathbb{Z}$, $\mathbb{F}_q$, $\mathbb{Q}_p$
- Arb : ball arithmetic, univariate polys and matrices over $\mathbb{R}$ and $\mathbb{C}$, special and transcendental functions
- Antic : element arithmetic over absolute number fields

AbstractAlgebra.jl:

- Generic rings: residue rings, fraction fields, dense univariate polynomials, sparse distributed multivariate polynomials

Nemo.jl:

- Flint : polynomials and matrices over $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/p\mathbb{Z}$, $\mathbb{F}_q$, $\mathbb{Q}_p$
- Arb : ball arithmetic, univariate polys and matrices over $\mathbb{R}$ and $\mathbb{C}$, special and transcendental functions
- Antic : element arithmetic over absolute number fields

AbstractAlgebra.jl:

- Generic rings: residue rings, fraction fields, dense univariate polynomials, sparse distributed multivariate polynomials, dense linear algebra, power series, permutation groups

Singular.jl

Access to Singular kernel functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\mathsf{GF}(p)$, etc.

Singular.jl

Access to Singular kernel functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\mathrm{GF}(p)$, etc.
- Polynomials, ideals, modules, matrices, etc.

Singular.jl

Access to Singular kernel functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\mathrm{GF}(p)$, etc.
- Polynomials, ideals, modules, matrices, etc.
- Groebner basis, resolutions, syzygies

Singular.jl

Access to Singular kernel functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\mathrm{GF}(p)$, etc.
- Polynomials, ideals, modules, matrices, etc.
- Groebner basis, resolutions, syzygies

Integration with Nemo.jl:

- Singular polynomials over any Nemo coefficient ring, e.g. Groebner bases over cyclotomic fields

Singular.jl

Access to Singular kernel functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\mathrm{GF}(p)$, etc.
- Polynomials, ideals, modules, matrices, etc.
- Groebner basis, resolutions, syzygies

Integration with Nemo.jl:

- Singular polynomials over any Nemo coefficient ring, e.g. Groebner bases over cyclotomic fields
- Nemo generics over any Singular ring

GAP.jl

- Group theory functionality

GAP.jl

- Group theory functionality
- Integration with Nemo/Julia

GAP.jl

- Group theory functionality
- Integration with Nemo/Julia
- Interface with Gap: ability to call Julia functions from Gap and vice versa

Hecke.jl

Algebraic number theory for Julia, built on Nemo.jl, AbstractAlgebra.jl, Flint, Antic, etc.

Hecke.jl

Algebraic number theory for Julia, built on Nemo.jl, AbstractAlgebra.jl, Flint, Antic, etc.

- Orders and ideals in absolute number fields

Hecke.jl

Algebraic number theory for Julia, built on Nemo.jl, AbstractAlgebra.jl, Flint, Antic, etc.

- Orders and ideals in absolute number fields
- Fast ideal and element arithmetic in absolute number fields

Hecke.jl

Algebraic number theory for Julia, built on Nemo.jl, AbstractAlgebra.jl, Flint, Antic, etc.

- Orders and ideals in absolute number fields
- Fast ideal and element arithmetic in absolute number fields
- Verified computations with approximations using interval arithmetic whenever necessary (e.g. computation with embeddings or residue computation of Dedekind zeta functions)

Hecke.jl

Algebraic number theory for Julia, built on Nemo.jl, AbstractAlgebra.jl, Flint, Antic, etc.

- Orders and ideals in absolute number fields
- Fast ideal and element arithmetic in absolute number fields
- Verified computations with approximations using interval arithmetic whenever necessary (e.g. computation with embeddings or residue computation of Dedekind zeta functions)
- Sparse linear algebra over $\mathbb{Z}$

Hecke.jl

Algebraic number theory for Julia, built on Nemo.jl, AbstractAlgebra.jl, Flint, Antic, etc.

- Orders and ideals in absolute number fields
- Fast ideal and element arithmetic in absolute number fields
- Verified computations with approximations using interval arithmetic whenever necessary (e.g. computation with embeddings or residue computation of Dedekind zeta functions)
- Sparse linear algebra over $\mathbb{Z}$
- Class and unit group computation

Hecke.jl

Algebraic number theory for Julia, built on Nemo.jl, AbstractAlgebra.jl, Flint, Antic, etc.

- Orders and ideals in absolute number fields
- Fast ideal and element arithmetic in absolute number fields
- Verified computations with approximations using interval arithmetic whenever necessary (e.g. computation with embeddings or residue computation of Dedekind zeta functions)
- Sparse linear algebra over $\mathbb{Z}$
- Class and unit group computation
- Pseudo-Hermite normal form for modules over Dedekind domains

Hecke.jl

Algebraic number theory for Julia, built on Nemo.jl, AbstractAlgebra.jl, Flint, Antic, etc.

- Orders and ideals in absolute number fields
- Fast ideal and element arithmetic in absolute number fields
- Verified computations with approximations using interval arithmetic whenever necessary (e.g. computation with embeddings or residue computation of Dedekind zeta functions)
- Sparse linear algebra over $\mathbb{Z}$
- Class and unit group computation
- Pseudo-Hermite normal form for modules over Dedekind domains
- Beginnings of class field theory and relative extensions

Oscar.jl (future work)

- Projects that make use of all of the above
- Present a consistent view of mathematics to the user: no need to worry about which implementation of the integers is being used behind the scenes
- Explore how far the Julia language can be pushed for computer algebra

Example improvement: Minpoly over $\mathbb{Z}$

Theorem

Suppose M is a linear operator on a K-vector space V, and that $V = W_1 + W_2 + \cdots + W_n$ for invariant subspaces W_i. Then the minimal polynomial of M is $LCM(m_1, m_2, \ldots, m_n)$, where m_i is the minimal polynomial of M restricted to W_i.

Example improvement: Minpoly over $\mathbb{Z}$

The subspaces we have in mind are the following:

Definition

Given a vector v in a vector space V the *Krylov subspace* $K(V, v)$ associated to v is the linear subspace spanned by $\{v, Mv, M^2v, \ldots\}$.

Example improvement: Minpoly over $\mathbb{Z}$

Idea:

- Reduce M modulo many small primes p and apply the method above

Example improvement: Minpoly over $\mathbb{Z}$

Idea:

- Reduce M modulo many small primes p and apply the method above
- Recombine using Chinese remaindering

Example improvement: Minpoly over $\mathbb{Z}$

Idea:

- Reduce M modulo many small primes p and apply the method above
- Recombine using Chinese remaindering
- (Giesbrecht) Can be finitely many "bad" primes, but these can be detected

Example improvement: Minpoly over $\mathbb{Z}$

Idea:

- Reduce M modulo many small primes p and apply the method above
- Recombine using Chinese remaindering
- (Giesbrecht) Can be finitely many "bad" primes, but these can be detected

Unfortunately, bounds on number of primes (e.g. Ovals of Cassini) are extremely pessimistic.

Example improvement: Minpoly over $\mathbb{Z}$

Idea:

- Reduce M modulo many small primes p and apply the method above
- Recombine using Chinese remaindering
- (Giesbrecht) Can be finitely many "bad" primes, but these can be detected

Unfortunately, bounds on number of primes (e.g. Ovals of Cassini) are extremely pessimistic.

Too expensive to evaluate the minpoly $m(T)$ at M. Need a better termination condition.

Example improvement: Minpoly over $\mathbb{Z}$

Idea:

- Record which standard basis vectors v_i were used to generate the Krylov subspaces W_i modulo p

Example improvement: Minpoly over $\mathbb{Z}$

Idea:

- Record which standard basis vectors v_i were used to generate the Krylov subspaces W_i modulo p
- When Chinese remaindering stabilises, lift all the v_i to $\mathbb{Z}$ and check $m(M)v_i = 0$

Example improvement: Minpoly over $\mathbb{Z}$

Idea:

- Record which standard basis vectors v_i were used to generate the Krylov subspaces W_i modulo p
- When Chinese remaindering stabilises, lift all the v_i to $\mathbb{Z}$ and check $m(M)v_i = 0$
- Can be checked using Matrix-Vector products, which are cheap

Example improvement: Minpoly over $\mathbb{Z}$

Idea:

- Record which standard basis vectors v_i were used to generate the Krylov subspaces W_i modulo p
- When Chinese remaindering stabilises, lift all the v_i to $\mathbb{Z}$ and check $m(M)v_i = 0$
- Can be checked using Matrix-Vector products, which are cheap
- Leads to worst case $O(n^4)$ algorithm, but generically $O(n^3)$

Characteristic and minimal polynomial

Table: Charpoly and minpoly timings

Op	Sage 6.9	Pari 2.7.4	Magma 2.21-4	Giac 1.2.2	Flint
Charpoly	0.2s	0.6s	0.06s	0.06s	0.04s
Minpoly	0.07s	>160 hrs	0.05s	0.06s	0.04s

for 80×80 matrix over $\mathbb{Z}$ with entries in $[-20, 20]$ and minpoly of degree 40.

Minimal polynomial over $\mathbb{Z}[x]$

Table: Minpoly timings

Op	Sage 6.9	Pari 2.7.4	Magma 2.21-4	Nemo-0.4
Minpoly	—	$>$ 160 hrs	—	0.04s

JuliaInterface and GAP.jl

GAP package JuliaInterface and Julia module GAP.jl

JuliaInterface and GAP.jl

GAP package JuliaInterface and Julia module GAP.jl

GAP $\longleftrightarrow$ Julia

JuliaInterface and GAP.jl

GAP package JuliaInterface and Julia module GAP.jl

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

JuliaInterface and GAP.jl provide

JuliaInterface and GAP.jl

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

JuliaInterface and GAP.jl

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

JuliaInterface and GAP.jl

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

JuliaInterface and GAP.jl

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

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

gap> JuliaUnbox( b );
2
```

JuliaInterface data structures: Objects

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

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

gap> JuliaUnbox( 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 := JuliaBox( a );
<Julia: 2>

gap> JuliaUnbox( 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 := JuliaBox( a );
<Julia: 2>

gap> JuliaUnbox( b );
2
```

Possible conversions:

- Integers
- Floats
- Permutations

JuliaInterface data structures: Objects

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

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

gap> JuliaUnbox( b );
2
```

Possible conversions:

- Integers
- Floats
- Permutations
- Finite field elements

JuliaInterface data structures: Objects

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

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

gap> JuliaUnbox( b );
2
```

Possible conversions:

- Integers
- Floats
- Permutations
- Finite field elements
- Nested lists of the above to Arrays

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>  jl_sqrt := JuliaFunction( "sqrt" );
<Julia function: sqrt>
```

JuliaInterface data structures: Functions

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

```
gap>  jl_sqrt := JuliaFunction( "sqrt" );
<Julia function: sqrt>

gap>  jl_sqrt( 4 );
2.
```

JuliaInterface data structures: Functions

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

```
gap> jl_sqrt := JuliaFunction( "sqrt" );
<Julia function: sqrt>

gap> jl_sqrt( 4 );
2.
```

- 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>  jl_sqrt := JuliaFunction( "sqrt" );
<Julia function: sqrt>

gap>  jl_sqrt( 4 );
2.
```

- Julia functions can be used like GAP functions
- Input data is converted to Julia, return value is converted back to GAP

JuliaInterface data structures: Functions

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

```
gap>  jl_sqrt := JuliaFunction( "sqrt" );
<Julia function: sqrt>

gap>  jl_sqrt( 4 );
2.
```

- Julia functions can be used like GAP functions
- Input data is converted to Julia, return value is converted back to GAP
- Calling only possible for convertible types

JuliaInterface data structures: Functions

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

```
gap> jl_sqrt := JuliaFunction( "sqrt" );
<Julia function: sqrt>

gap> jl_sqrt( 4 );
2.
```

- Julia functions can be used like GAP functions
- Input data is converted to Julia, return value is converted back to GAP
- Calling only possible for convertible types

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:

```
function orbit( self, 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( 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( 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( 1, S, OnPoints );; time;
5769

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

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

From the GAP side

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

From the GAP side

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

Speedup

From the GAP side

How does GAP benefit from 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 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 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 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:

- Julia is more flexible then C

From the GAP side

How does GAP benefit from 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:

- Julia is more flexible then C
- Julia has more functionality available in its standard library than C

From the GAP side

How does GAP benefit from 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:

- Julia is more flexible then C
- Julia has more functionality available in its standard library than C
- Julia may be easier to use then C

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

Category theory as programming language

Category theory

Category theory as programming language

Category theory

- abstracts mathematical structures

Category theory as programming language

Category theory

- abstracts mathematical structures
- defines a *language* to formulate theorems and algorithms for different structures *at the same time*

Category theory as programming language

Category theory

- abstracts mathematical structures
- defines a *language* to formulate theorems and algorithms for different structures *at the same time*

CAP - Categories, Algorithms, Programming

Category theory as programming language

Category theory

- abstracts mathematical structures
- defines a *language* to formulate theorems and algorithms for different structures *at the same time*

CAP - Categories, Algorithms, Programming

CAP implements a
categorical programming language
(j/w Sebastian Posur)

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$

A B C

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$

$$A \qquad B \qquad C$$

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$

$$
\begin{tikzcd}
A \arrow[r, red] & B & C
\end{tikzcd}
$$

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$

$$A \longrightarrow B \longrightarrow C$$

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)

$$A \longrightarrow B \longrightarrow C$$

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)

$$\begin{tikzcd} A \arrow[r] \arrow[rr, bend left] & B \arrow[r] & C \end{tikzcd}$$

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

$$\begin{tikzcd} A \arrow[r] \arrow[rr, bend left] & B \arrow[r] & C \end{tikzcd}$$

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

$$\begin{tikzcd} A \arrow[r] \arrow[rr, bend left] \arrow[loop below] & B \arrow[r] \arrow[loop below] & C \arrow[loop below] \end{tikzcd}$$

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

$$\begin{tikzcd}
A \arrow[r] \arrow[rr, bend left] \arrow[loop below] & B \arrow[r] \arrow[loop below] & C \arrow[loop below]
\end{tikzcd}$$

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

$$\begin{tikzcd} A \arrow[r] \arrow[rr, bend left] \arrow[loop below] & B \arrow[r] \arrow[loop below] & C \arrow[loop below] \end{tikzcd}$$

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

Computable Category

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

Computable Category

A category becomes computable by making the existential quantifiers from the definition of a category constructive,

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

Computable Category

A category becomes computable by making the existential quantifiers from the definition of a category constructive, i.e., giving

- data structures for objects and morphisms

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

Computable Category

A category becomes computable by making the existential quantifiers from the definition of a category constructive, i.e., giving

- data structures for objects and morphisms

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

Computable Category

A category becomes computable by making the existential quantifiers from the definition of a category constructive, i.e., giving

- data structures for objects and morphisms

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

Computable Category

A category becomes computable by making the existential quantifiers from the definition of a category constructive, i.e., giving

- data structures for objects and morphisms
- algorithms for composition and identity morphism

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

Computable Category

A category becomes computable by making the existential quantifiers from the definition of a category constructive, i.e., giving

- data structures for objects and morphisms
- algorithms for composition and identity morphism

Computable categories

Definition

A category $\mathcal{A}$ contains the following data:

- $\mathrm{Obj}_{\mathcal{A}}$
- $\mathrm{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \mathrm{Hom}_{\mathcal{A}}(B, C) \times \mathrm{Hom}_{\mathcal{A}}(A, B) \to \mathrm{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\mathrm{id}_A \in \mathrm{Hom}_{\mathcal{A}}(A, A)$

Computable Category

A category becomes computable by making the existential quantifiers from the definition of a category constructive, i.e., giving

- data structures for objects and morphisms
- algorithms for composition and identity morphism

Computable categories

Some categorical operations in abelian categories

Computable categories

Some categorical operations in abelian categories

- Zero morphisms

Computable categories

Some categorical operations in abelian categories

- Zero morphisms
- Addition and subtraction of morphisms

Computable categories

Some categorical operations in abelian categories

- Zero morphisms
- Addition and subtraction of morphisms
- Direct sums

Computable categories

Some categorical operations in abelian categories

- Zero morphisms
- Addition and subtraction of morphisms
- Direct sums
- Kernels and Cokernels of morphisms

Computable categories

Some categorical operations in abelian categories

- Zero morphisms
- Addition and subtraction of morphisms
- Direct sums
- Kernels and Cokernels of morphisms
- ...

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$.

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$.

$$A \xrightarrow{\varphi} B$$

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

$$A \xrightarrow{\varphi} B$$

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker \varphi$,

$$\begin{array}{ccc} \ker \varphi & & \\ & A \xrightarrow{\varphi} B \end{array}$$

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker \varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,

$$\begin{tikzcd} \ker \varphi \arrow[dr, hook, "\kappa"] & & \\ & A \arrow[r, "\varphi"] & B \end{tikzcd}$$

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker\varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,
and for every test morphism τ

$$\begin{tikzcd}
\ker\varphi \arrow[dr, hook, "\kappa"] \arrow[drr, bend left, "0"] & & \\
& A \arrow[r, "\varphi"] & B \\
T \arrow[ur, "\tau"] \arrow[urr, bend right, "0"'] & &
\end{tikzcd}$$

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker\varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,
and for every test morphism τ
a *unique* morphism $\lambda = \mathrm{KernelLift}(\varphi, \tau)$

$$\begin{tikzcd}
\ker\varphi \arrow[dr, hook, "\kappa"] \arrow[drr, bend left, "0"] & & \\
 & A \arrow[r, "\varphi"] & B \\
T \arrow[uu, dashed, "\lambda"] \arrow[ur, "\tau"] \arrow[urr, bend right, "0"'] & &
\end{tikzcd}$$

Implementation of the kernel

Let $\varphi \in \mathrm{Hom}(A, B)$. To fully describe the kernel of φ ...

... one needs an object $\ker\varphi$,
its embedding $\kappa = \mathrm{KernelEmbedding}(\varphi)$,
and for every test morphism τ
a *unique* morphism $\lambda = \mathrm{KernelLift}(\varphi, \tau)$, such that

$$
\begin{tikzcd}
\ker\varphi \arrow[dr, hook, "\kappa"] \arrow[drr, bend left, "0"] \arrow[d, phantom, "\circlearrowright", shift left=4ex] & & \\
 & A \arrow[r, "\varphi"] & B \\
T \arrow[uu, dashed, "\lambda"] \arrow[ur, "\tau"] \arrow[urr, bend right, "0"'] & &
\end{tikzcd}
$$

What is CAP ?

CAP - Categories, Algorithms, and Programming

What is CAP ?

CAP - Categories, Algorithms, and Programming

CAP is a framework to implement computable categories and provides

What is CAP ?

CAP - Categories, Algorithms, and Programming

CAP is a framework to implement computable categories and provides

- specifications of categorical operations

What is CAP ?

CAP - Categories, Algorithms, and Programming

CAP is a framework to implement computable categories and provides

- specifications of categorical operations
- generic algorithms based on basic categorical operations

What is CAP ?

CAP - Categories, Algorithms, and Programming

CAP is a framework to implement computable categories and provides

- specifications of categorical operations
- generic algorithms based on basic categorical operations
- a categorical programming language having categorical operations as syntax elements

Computing the intersection

Let $M_1 \subseteq N$ and $M_2 \subseteq N$ subobjects.

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.
Compute their intersection $\gamma : M_1 \cap M_2 \hookrightarrow N$.

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.
Compute their intersection $\gamma : M_1 \cap M_2 \hookrightarrow N$.

$$
\begin{tikzcd}
M_1 \arrow[dr, hook, "\iota_1"] & \\
& N \\
M_2 \arrow[ur, hook, "\iota_2"'] &
\end{tikzcd}
$$

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.
Compute their intersection $\gamma : M_1 \cap M_2 \hookrightarrow N$.

$$\begin{tikzcd}
 & M_1 \arrow[dr, hook, "\iota_1"] & \\
M_1 \oplus M_2 & & N \\
 & M_2 \arrow[ur, hook, "\iota_2"'] &
\end{tikzcd}$$

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.
Compute their intersection $\gamma : M_1 \cap M_2 \hookrightarrow N$.

$$\begin{tikzcd}
 & M_1 \arrow[dr, hook, "\iota_1"] & \\
M_1 \oplus M_2 \arrow[ur, "\pi_1"] & & N \\
 & M_2 \arrow[ur, hook, "\iota_2"'] &
\end{tikzcd}$$

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.
Compute their intersection $\gamma : M_1 \cap M_2 \hookrightarrow N$.

$$\begin{tikzcd}
 & M_1 \arrow[dr, hook, "\iota_1"] & \\
M_1 \oplus M_2 \arrow[ur, two heads, "\pi_1"] \arrow[dr, two heads, "\pi_2"'] & & N \\
 & M_2 \arrow[ur, hook, "\iota_2"'] &
\end{tikzcd}$$

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.
Compute their intersection $\gamma : M_1 \cap M_2 \hookrightarrow N$.

$$\begin{tikzcd} & M_1 \arrow[dr, hook, "\iota_1"] & \\ M_1 \oplus M_2 \arrow[ur, "\pi_1"] \arrow[dr, "\pi_2"'] & & N \\ & M_2 \arrow[ur, hook, "\iota_2"'] & \end{tikzcd}$$

- $\pi_i := \mathrm{ProjectionInFactorOfDirectSum}\left(\left(M_1, M_2\right), i\right),\ i = 1, 2$

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.
Compute their intersection $\gamma : M_1 \cap M_2 \hookrightarrow N$.

$$\begin{tikzcd}
 & M_1 \arrow[dr, hook, "\iota_1"] & \\
M_1 \oplus M_2 \arrow[ur, two heads, "\pi_1"] \arrow[dr, two heads, "\pi_2"'] \arrow[rr, "\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2"] & & N \\
 & M_2 \arrow[ur, hook, "\iota_2"'] &
\end{tikzcd}$$

- $\pi_i := \mathrm{ProjectionInFactorOfDirectSum}\left(\left(M_1, M_2\right), i\right),\ i = 1, 2$

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.
Compute their intersection $\gamma : M_1 \cap M_2 \hookrightarrow N$.

$$\begin{tikzcd} & M_1 \arrow[dr, hook, "\iota_1"] & \\ M_1 \oplus M_2 \arrow[ur, two heads, "\pi_1"] \arrow[rr, "\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2"] \arrow[dr, two heads, "\pi_2"'] & & N \\ & M_2 \arrow[ur, hook, "\iota_2"'] & \end{tikzcd}$$

- $\pi_i := \mathrm{ProjectionInFactorOfDirectSum}\left(\left(M_1, M_2\right), i\right),\ i = 1, 2$
- $\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2$

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.
Compute their intersection $\gamma : M_1 \cap M_2 \hookrightarrow N$.

$$\begin{tikzcd}
 & & M_1 \arrow[dr, hook, "\iota_1"] & \\
M_1 \cap M_2 \arrow[r, hook, "\kappa"] & M_1 \oplus M_2 \arrow[ur, "\pi_1"] \arrow[dr, "\pi_2"'] \arrow[rr, "\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2"] & & N \\
 & & M_2 \arrow[ur, hook, "\iota_2"'] &
\end{tikzcd}$$

- $\pi_i := \mathrm{ProjectionInFactorOfDirectSum}\left(\left(M_1, M_2\right), i\right), \; i = 1, 2$
- $\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2$

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.
Compute their intersection $\gamma : M_1 \cap M_2 \hookrightarrow N$.

$$\begin{tikzcd}
 & & M_1 \arrow[dr, hook, "\iota_1"] & \\
M_1 \cap M_2 \arrow[r, hook, "\kappa"] & M_1 \oplus M_2 \arrow[ur, two heads, "\pi_1"] \arrow[rr, "\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2"] \arrow[dr, two heads, "\pi_2"'] & & N \\
 & & M_2 \arrow[ur, hook, "\iota_2"'] &
\end{tikzcd}$$

- $\pi_i := \mathrm{ProjectionInFactorOfDirectSum}\left((M_1, M_2), i\right), \ i = 1, 2$
- $\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2$
- $\kappa := \mathrm{KernelEmbedding}(\varphi)$

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.
Compute their intersection $\gamma : M_1 \cap M_2 \hookrightarrow N$.

$$\begin{tikzcd}
 & & M_1 \arrow[dr, hook, "\iota_1"] & \\
M_1 \cap M_2 \arrow[r, hook, "\kappa"] & M_1 \oplus M_2 \arrow[ur, "\pi_1"] \arrow[dr, "\pi_2"'] \arrow[rr, "\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2"] & & N \\
 & & M_2 \arrow[ur, hook, "\iota_2"'] &
\end{tikzcd}$$

- $\pi_i := \mathrm{ProjectionInFactorOfDirectSum}\left(\left(M_1, M_2\right), i\right), \; i = 1, 2$
- $\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2$
- $\kappa := \mathrm{KernelEmbedding}\left(\varphi\right)$

Computing the intersection

Let $M_1 \hookrightarrow N$ and $M_2 \hookrightarrow N$ subobjects.
Compute their intersection $\gamma : M_1 \cap M_2 \hookrightarrow N$.

$$\begin{tikzcd}
 & & M_1 \arrow[dr, hook, "\iota_1"] & \\
M_1 \cap M_2 \arrow[r, hook, "\kappa"] & M_1 \oplus M_2 \arrow[ur, two heads, "\pi_1"] \arrow[rr, "\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2"] \arrow[dr, two heads, "\pi_2"'] & & N \\
 & & M_2 \arrow[ur, hook, "\iota_2"'] &
\end{tikzcd}$$

- $\pi_i := \mathrm{ProjectionInFactorOfDirectSum}\left(\left(M_1, M_2\right), i\right),\ i = 1, 2$
- $\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2$
- $\kappa := \mathrm{KernelEmbedding}\left(\varphi\right)$
- $\gamma := \iota_1 \circ \pi_1 \circ \kappa$

Translation to CAP

$\pi_i := \mathrm{ProjectionInFactorOfDirectSum}\left((M_1, M_2), i\right),\ i = 1, 2$

$\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2$

$\kappa := \mathrm{KernelEmbedding}\left(\varphi\right)$

$\gamma := \iota_1 \circ \pi_1 \circ \kappa$

Translation to CAP

$\pi_i := \text{ProjectionInFactorOfDirectSum}\left(\left(M_1, M_2\right), i\right)$, $i = 1, 2$

```
pi1 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 1 );
pi2 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 2 );
```

$\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2$

$\kappa := \text{KernelEmbedding}\left(\varphi\right)$

$\gamma := \iota_1 \circ \pi_1 \circ \kappa$

Translation to CAP

$\pi_i := \text{ProjectionInFactorOfDirectSum}\left(\left(M_1, M_2\right), i\right),\ i = 1, 2$

```
pi1 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 1 );
pi2 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 2 );
```

$\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2$

```
lambda := PostCompose( iota1, pi1 );
phi := lambda - PostCompose( iota2, pi2 );
```

$\kappa := \text{KernelEmbedding}\left(\varphi\right)$

$\gamma := \iota_1 \circ \pi_1 \circ \kappa$

Translation to CAP

$\pi_i := \mathrm{ProjectionInFactorOfDirectSum}\left(\left(M_1, M_2\right), i\right),\ i = 1, 2$

```
pi1 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 1 );
pi2 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 2 );
```

$\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2$

```
lambda := PostCompose( iota1, pi1 );
phi := lambda - PostCompose( iota2, pi2 );
```

$\kappa := \mathrm{KernelEmbedding}\left(\varphi\right)$

```
kappa := KernelEmbedding( phi );
```

$\gamma := \iota_1 \circ \pi_1 \circ \kappa$

Translation to CAP

$\pi_i := \mathrm{ProjectionInFactorOfDirectSum}\left(\left(M_1, M_2\right), i\right),\ i = 1, 2$

```
pi1 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 1 );
pi2 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 2 );
```

$\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2$

```
lambda := PostCompose( iota1, pi1 );
phi := lambda - PostCompose( iota2, pi2 );
```

$\kappa := \mathrm{KernelEmbedding}\left(\varphi\right)$

```
kappa := KernelEmbedding( phi );
```

$\gamma := \iota_1 \circ \pi_1 \circ \kappa$

```
gamma := PostCompose( lambda, kappa );
```

Translation to CAP

```
pi1 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 1 );
pi2 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 2 );
```

```
lambda := PostCompose( iota1, pi1 );
phi := lambda - PostCompose( iota2, pi2 );
```

```
kappa := KernelEmbedding( phi );
```

```
gamma := PostCompose( lambda, kappa );
```

Translation to CAP

```
pi1 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 1 );
pi2 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 2 );
lambda := PostCompose( iota1, pi1 );
phi := lambda - PostCompose( iota2, pi2 );
kappa := KernelEmbedding( phi );
gamma := PostCompose( lambda, kappa );
```

Translation to CAP

```
IntersectionOfObjects := function( iota1, iota2 )

  pi1 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 1 );
  pi2 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 2 );
  lambda := PostCompose( iota1, pi1 );
  phi := lambda - PostCompose( iota2, pi2 );
  kappa := KernelEmbedding( phi );
  gamma := PostCompose( lambda, kappa );
```

Translation to CAP

```
IntersectionOfObjects := function( iota1, iota2 )

  M1 := Source( iota1 );
  M2 := Source( iota2 );

  pi1 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 1 );
  pi2 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 2 );

  lambda := PostCompose( iota1, pi1 );
  phi := lambda - PostCompose( iota2, pi2 );

  kappa := KernelEmbedding( phi );

  gamma := PostCompose( lambda, kappa );
```

Translation to CAP

```
IntersectionOfObjects := function( iota1, iota2 )

  M1 := Source( iota1 );
  M2 := Source( iota2 );
  pi1 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 1 );
  pi2 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 2 );
  lambda := PostCompose( iota1, pi1 );
  phi := lambda - PostCompose( iota2, pi2 );
  kappa := KernelEmbedding( phi );
  gamma := PostCompose( lambda, kappa );
  return gamma;
end;
```

Translation to CAP

```
IntersectionOfObjects := function( iota1, iota2 )
  local M1, M2, pi1, pi2, lambda, phi, kappa, gamma;
  M1 := Source( iota1 );
  M2 := Source( iota2 );
  pi1 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 1 );
  pi2 := ProjectionInFactorOfDirectSum( [ M1, M2 ], 2 );
  lambda := PostCompose( iota1, pi1 );
  phi := lambda - PostCompose( iota2, pi2 );
  kappa := KernelEmbedding( phi );
  gamma := PostCompose( lambda, kappa );
  return gamma;
end;
```

Mathematica: Integration

```
In[1]:= Integrate[ArcTan[x] - ArcCot[1/x], {x,0,1}]

Out[1]= 0

In[2]:= Integrate[ArcTan[x] - ArcCot[1/x], {x,0,1.0}]

                        -15
Out[2]= -7.88258 10
```

Mathematica: Integration

```
In[1]:= Integrate[ArcTan[x] - ArcCot[1/x], {x,0,1}]

Out[1]= 0

In[2]:= Integrate[ArcTan[x] - ArcCot[1/x], {x,0,1.0}]

                          -15
Out[2]= -7.88258 10

In[3]:= FullSimplify[ArcTan[x] - ArcCot[1/x]]

Out[3]= 0
```

Sage: Gröbner Bases

```
a = 1
b = 2
c = -3

x,y = QQ['x,y'].gens()
f = a*x^3*y^2+b*x+y^2+1
g = c*x*y^4+x^3+y
I = ideal(f,g)
B = I.groebner_basis(); B

[y^6 + 1/3*x^2*y^3 - 1/3*x^2*y^2 + y^4
     - 1/3*x^2 + 2/3*y,
 x^5 + 3*y^4 + x^2*y + 6*x*y^2 + 3*y^2,
 x^3*y^2 + y^2 + 2*x + 1,
 x*y^4 - 1/3*x^3 - 1/3*y]
```

Sage: Plotting Curves

```
var('x,y')
f = a*x^3*y^2+b*x+y^2+1
g = c*x*y^4+x^3+y

C = implicit_plot(f, (x,-2,2), (y,-2,2),
    cmap=['red'], plot_points=150, fill=False)
D = implicit_plot(g, (x,-2,2), (y,-2,2),
    cmap=['blue'], plot_points=150, fill=False)
C+D
```