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

**GAP4 Language**
- **GAP Groups**
  - Discrete Mathematics
- **SINGULAR**
  - Algebraic Geometry
  - Commutative Algebra
  - **JSingular**
    - Just-in-time compiler for SINGULAR in JULIA

**POLYMAKE**
- Convex Geometry

**ANTS**
- **ANTIC**
  - Number Theory

**Flint**
- Arithmetic for Number Theory

**Nemo.jl**
- Generic Arithmetic for Recursive Data Structures

**Plural**
- Letterplace Non-commutative Algebra

**GBLA**
- F4/F5 Gröbner Basis Algorithms
- Fast Linear Algebra

**FACTORY**
- Polynomial Factorization

**Normaliz**
- Affine Semigroups

**Gfan**
- Tropical Geometry

**GBLA**
- F4/F5 Gröbner Basis Algorithms
- Fast Linear Algebra

**primdec.lib**
- Primary Decomposition

**Library Connections**
- **GAP libraries, e.g.** homalg Homological Algebra
- **CHEVIE**
  - Generic Character Tables
- **POLYMAKE extensions, e.g.**
  - a-tint Tropical Intersection Theory
- **Nemo.jl**
  - Generic Arithmetic for Recursive Data Structures
- **ANTIC libraries, e.g.**
- **JULIA libraries, e.g.**

**OSCAR: A visionary, new computer algebra system**
OSCAR: A visionary, new computer algebra system.
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.

Decker, Gutsche, Hart, Joswig

OSCAR: A visionary, new computer algebra system
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)

- 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.
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
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;
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
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.
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:

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

- Take mathematical problems within TRR195 and international community into account.
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).
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).
Julia access to Singular KERNEL functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\text{GF}(p)$, etc.
Example for Immediate Benefits: Singular.jl

Julia access to Singular KERNEL functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\text{GF}(p)$, etc.
- Polynomials, ideals, modules, matrices, etc.
Julia access to Singular KERNEL functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\text{GF}(p)$, etc.
- Polynomials, ideals, modules, matrices, etc.
- Gröbner bases, syzygies, free resolutions
Julia access to Singular KERNEL functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\text{GF}(p)$, 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 $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, 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
Julia access to Singular KERNEL functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\text{GF}(p)$, 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 $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\text{GF}(p)$, 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
- Clara Petroll: Bachelor thesis, 2017
Some History

- David Eisenbud and Bernd Sturmfels: *Binomial Ideals*, 1996
- 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 binomioal ideals)

```julia
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
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.

Decker, Gutsche, Hart, Joswig

OSCAR: A visionary, new computer algebra system
- 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: Flints: 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

Decker, Gutsche, Hart, Joswig

OSCAR: A visionary, new computer algebra system
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 $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\text{GF}(p)$, etc.
Access to Singular kernel functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\text{GF}(p)$, etc.
- Polynomials, ideals, modules, matrices, etc.
Access to Singular kernel functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\text{GF}(p)$, etc.
- Polynomials, ideals, modules, matrices, etc.
- Groebner basis, resolutions, syzygies
Access to Singular kernel functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\text{GF}(p)$, 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
Access to Singular kernel functions and data types:

- Coefficient rings $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, $\text{GF}(p)$, 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
- 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.
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 $\mathbb{Z}$
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
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
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 $\text{LCM}(m_1, m_2, \ldots, m_n)$, where $m_i$ is the minimal polynomial of $M$ restricted to $W_i$. 

Decker, Gutsche, Hart, Joswig

OSCAR: A visionary, new computer algebra system
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 $m(M)v_i = 0$

Can be checked using Matrix-Vector products, which are cheap. Leads to worst case $O(n^4)$ algorithm, but generically $O(n^3)$.
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 \( O(n^3) \)
### Table: Charpoly and minpoly timings

<table>
<thead>
<tr>
<th>Op</th>
<th>Sage 6.9</th>
<th>Pari 2.7.4</th>
<th>Magma 2.21-4</th>
<th>Giac 1.2.2</th>
<th>Flint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charpoly</td>
<td>0.2s</td>
<td>0.6s</td>
<td>0.06s</td>
<td>0.06s</td>
<td>0.04s</td>
</tr>
<tr>
<td>Minpoly</td>
<td>0.07s</td>
<td>&gt;160 hrs</td>
<td>0.05s</td>
<td>0.06s</td>
<td>0.04s</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Op</th>
<th>Sage 6.9</th>
<th>Pari 2.7.4</th>
<th>Magma 2.21-4</th>
<th>Nemo-0.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minpoly</td>
<td>—</td>
<td>$&gt; 160$ hrs</td>
<td>—</td>
<td>0.04s</td>
</tr>
</tbody>
</table>

**Table:** Minpoly timings

OSCAR: A visionary, new computer algebra system
GAP package JuliaInterface and Julia module 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

GAP $\leftrightarrow$ Julia
Julia Interface 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 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
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
### JuliaInterface and GAP.jl

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

![Diagram](image)

**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 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
gap> b := JuliaBox( 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 := 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
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
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
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 contains GAP data structures that can hold pointers to Julia objects:

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

```gap
jl_sqrt := JuliaFunction( "sqrt" );
<Julia function: sqrt>
```
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 provides the possibility to call Julia functions by converting GAP objects:

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

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

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

\begin{verbatim}
gap> jl_sqrt := JuliaFunction( "sqrt" );
<Julia function: sqrt>

gap> jl_sqrt( 4 );
2.
\end{verbatim}

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

```gap
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:

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

```plaintext
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
```

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 );
```
JuliaInterface: Julia functions as kernel modules

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
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 ) );;
```
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( 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
```
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( 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**

- Find time critical parts of algorithms, rewrite them in C.
- Find time critical parts of algorithms, rewrite them in Julia.
- **Benefits:**
  - Julia is more flexible than C.
  - Julia has more functionality available in its standard library than C.
  - Julia may be easier to use 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.
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 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
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)?
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

OSCAR: A visionary, new computer algebra system
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)?

### 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
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

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)

Decker, Gutsche, Hart, Joswig

OSCAR: A visionary, new computer algebra system
Category theory as programming language

Category theory

- abstracts mathematical structures
Category theory abstracts mathematical structures and 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

Decker, Gutsche, Hart, Joswig

OSCAR: A visionary, new computer algebra system
Category theory abstracts mathematical structures and 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:

<table>
<thead>
<tr>
<th>$\text{Obj} \mathcal{A}$</th>
<th>$\text{Hom} \mathcal{A}(A, B) \times \text{Hom} \mathcal{A}(A, B)$</th>
<th>$\text{Hom} \mathcal{A}(A, C)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{id}_A \in \text{Hom} \mathcal{A}(A, A)$</td>
<td>$\text{assoc.}$</td>
<td>$\text{Hom} \mathcal{A}(A, A)$</td>
</tr>
</tbody>
</table>
A category $\mathcal{A}$ contains the following data:

- $\text{Obj}_{\mathcal{A}}$
Computable categories

Definition

A category \( \mathcal{A} \) contains the following data:

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

\[ \circ : \text{Hom}_{\mathcal{A}}(B, C) \times \text{Hom}_{\mathcal{A}}(A, B) \to \text{Hom}_{\mathcal{A}}(A, C) \]

Neutral elements: \( \text{id}_A \in \text{Hom}_{\mathcal{A}}(A, A) \)

A & B & C

Decker, Gutsche, Hart, Joswig

OSCAR: A visionary, new computer algebra system
Computable categories

Definition

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

- $\text{Obj}_\mathcal{A}$
- $\text{Hom}_\mathcal{A}(A, B)$
A category $\mathcal{A}$ contains the following data:

- $\text{Obj}_{\mathcal{A}}$
- $\text{Hom}_{\mathcal{A}}(A, B)$
A category $\mathcal{A}$ contains the following data:

- $\text{Obj}_{\mathcal{A}}$
- $\text{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \text{Hom}_{\mathcal{A}}(B, C) \times \text{Hom}_{\mathcal{A}}(A, B) \to \text{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
A category $\mathcal{A}$ contains the following data:

- $\text{Obj}_\mathcal{A}$
- $\text{Hom}_\mathcal{A}(A, B)$
- $\circ : \text{Hom}_\mathcal{A}(B, C) \times \text{Hom}_\mathcal{A}(A, B) \rightarrow \text{Hom}_\mathcal{A}(A, C)$ (assoc.)
A category $\mathcal{A}$ contains the following data:

- $\text{Obj}_{\mathcal{A}}$
- $\text{Hom}_{\mathcal{A}}(A, B)$
- $\circ : \text{Hom}_{\mathcal{A}}(B, C) \times \text{Hom}_{\mathcal{A}}(A, B) \to \text{Hom}_{\mathcal{A}}(A, C)$ (assoc.)
- Neutral elements: $\text{id}_A \in \text{Hom}_{\mathcal{A}}(A, A)$
A category $\mathcal{A}$ contains the following data:

- $\text{Obj}_\mathcal{A}$
- $\text{Hom}_\mathcal{A}(A, B)$
- $\circ : \text{Hom}_\mathcal{A}(B, C) \times \text{Hom}_\mathcal{A}(A, B) \rightarrow \text{Hom}_\mathcal{A}(A, C)$ (assoc.)
- Neutral elements: $\text{id}_A \in \text{Hom}_\mathcal{A}(A, A)$
A category $\mathcal{A}$ contains the following data:

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

**Definition**

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

- $\text{Obj}_\mathcal{A}$
- $\text{Hom}_\mathcal{A}(A, B)$
- $\circ : \text{Hom}_\mathcal{A}(B, C) \times \text{Hom}_\mathcal{A}(A, B) \to \text{Hom}_\mathcal{A}(A, C)$ (assoc.)
- Neutral elements: $\text{id}_A \in \text{Hom}_\mathcal{A}(A, A)$
A category $\mathcal{A}$ contains the following data:

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

**Definition**

A category \( \mathcal{A} \) contains the following data:

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

**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:

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

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:

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

**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:

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

**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:

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

**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:

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

### 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:

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

**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:

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

**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
Some categorical operations in abelian categories
Some categorical operations in abelian categories

- Zero morphisms
Some categorical operations in abelian categories

- Zero morphisms
- Addition and subtraction of morphisms
Some categorical operations in abelian categories

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

Decker, Gutsche, Hart, Joswig
OSCAR: A visionary, new computer algebra system
Implementation of the kernel

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

\[ \ldots \text{one needs an object } \ker \varphi, \]

\[ \begin{align*}
A \xrightarrow{\varphi} B
\end{align*} \]
Let \( \varphi \in \text{Hom}(A, B) \). To fully describe the kernel of \( \varphi \) ... 

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

\[
\begin{array}{c}
\text{ker } \varphi \\
\kappa
\end{array}
\quad \begin{array}{c}
A \\
\varphi
\end{array}
\quad \begin{array}{c}
B
\end{array}
\]
Implementation of the kernel

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

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

\[
\begin{array}{ccc}
    \ker \varphi & \xrightarrow{\kappa} & 0 \\
    ^{\kappa} & \downarrow & \downarrow \varphi \\
    A & \xrightarrow{\tau} & B \\
    ^{T} & \downarrow & \downarrow \\
    0 & = & 0
\end{array}
\]
Let $\phi \in \text{Hom}(A, B)$. To fully describe the kernel of $\phi$ . . .

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

\[\begin{array}{c}
\text{ker } \phi \\
\uparrow \lambda \\
\downarrow T \\
0 \\
\end{array} \xleftarrow{\kappa} \xrightarrow{\tau} A \xrightarrow{\phi} B \xrightarrow{\kappa} 0 \xrightarrow{\lambda} \text{ker } \phi\]
Let $\varphi \in \text{Hom}(A, B)$. To fully describe the kernel of $\varphi$ . . .

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

\[
\begin{array}{c}
\ker \varphi & \xrightarrow{\kappa} & 0 \\
\downarrow{\lambda} & {\kappa} & \downarrow{\tau} \\
T & \xrightarrow{\varphi} & A & \xrightarrow{\varphi} & B \\
\downarrow{\tau} & {\lambda} & \downarrow{\tau} & {\lambda} & \downarrow{\tau} \\
0 & \xleftarrow{T} & 0 & \xleftarrow{T} & 0
\end{array}
\]
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, and a categorical programming language having categorical operations as syntax elements.
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

Decker, Gutsche, Hart, Joswig

OSCAR: A visionary, new computer algebra system
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 $\gamma : M_1 \cap M_2 \hookrightarrow N$. 

\[ \begin{array}{c}
M_1 \\
\downarrow l_1 \\
N \\
\downarrow l_2 \\
M_2 
\end{array} \]
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 $\gamma : M_1 \cap M_2 \hookrightarrow N$. 

\[
\begin{array}{c}
M_1 \\
\pi_1 \\
M_1 \oplus M_2 \\
\end{array} \\
\begin{array}{c}
M_1 \\
\iota_1 \\
N \\
\end{array} \\
\begin{array}{c}
M_2 \\
\iota_2 \\
N \\
\end{array}
\]
Computing the intersection

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

\[
\begin{align*}
M_1 \oplus M_2 & \quad M_1 & \quad N \\
\pi_1 & \quad \pi_2 & \quad \pi_1 \circ \kappa & \quad \pi_2 \circ \kappa
\end{align*}
\]
Computing the intersection

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

\[ \pi_i := \text{ProjectionInFactorOfDirectSum}((M_1, M_2), i), \ i = 1, 2 \]
Computing the intersection

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

\[ \pi_i := \text{ProjectionInFactorOfDirectSum} \left( (M_1, M_2), i \right), \ i = 1, 2 \]
Computing the intersection

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

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

Decker, Gutsche, Hart, Joswig
OSCAR: A visionary, new computer algebra system
Computing the intersection

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

- $\pi_i := \text{ProjectionInFactorOfDirectSum}((M_1, M_2), i), \ i = 1, 2$
- $\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2$
Computing the intersection

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

- $\pi_i := \text{ProjectionInFactorOfDirectSum}((M_1, M_2), i), \ i = 1, 2$
- $\varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2$
- $\kappa := \text{KernelEmbedding}(\varphi)$
Computing the intersection

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

\[ \pi_i := \text{ProjectionInFactorOfDirectSum}((M_1, M_2), i), \ i = 1, 2 \]
\[ \varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2 \]
\[ \kappa := \text{KernelEmbedding}(\varphi) \]
Computing the intersection

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

\[ \begin{array}{c}
M_1 \cap M_2 \xrightarrow{\kappa} M_1 \oplus M_2 \\
\phantom{M_1 \cap M_2} \xrightarrow{\pi_1} \phantom{M_1 \oplus M_2} M_1 \\
\phantom{M_1 \cap M_2} \xrightarrow{\pi_2} \phantom{M_1 \oplus M_2} M_2 \\
\phantom{M_1 \cap M_2} \xrightarrow{\varphi} \phantom{M_1 \oplus M_2} N \\
\phantom{M_1 \cap M_2} \xrightarrow{\iota_1 \circ \pi_1 - \iota_2 \circ \pi_2} \phantom{M_1 \oplus M_2} N \\
\end{array} \]

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

\[ \varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2 \]

\[ \kappa := \text{KernelEmbedding}(\varphi) \]

\[ \gamma := \iota_1 \circ \pi_1 \circ \kappa \]
\[ \pi_i := \text{ProjectionInFactorOfDirectSum}((M_1, M_2), i), \; i = 1, 2 \]

\[
\begin{align*}
\pi_1 & := \text{ProjectionInFactorOfDirectSum}( [ M_1, M_2 ], 1 ); \\
\pi_2 & := \text{ProjectionInFactorOfDirectSum}( [ M_1, M_2 ], 2 );
\end{align*}
\]

\[ \varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2 \]

\[ \kappa := \text{KernelEmbedding}(\varphi) \]

\[ \gamma := \iota_1 \circ \pi_1 \circ \kappa \]
\[ \pi_i := \text{ProjectionInFactorOfDirectSum}((M_1, M_2), i), \ i = 1, 2 \]

\[
\begin{align*}
\pi_1 &:= \text{ProjectionInFactorOfDirectSum}([M_1, M_2], 1); \\
\pi_2 &:= \text{ProjectionInFactorOfDirectSum}([M_1, M_2], 2);
\end{align*}
\]

\[ \varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2 \]

\[
\begin{align*}
\lambda &:= \text{PostCompose}(\iota_1, \pi_1) \\
\phi &:= \lambda - \text{PostCompose}(\iota_2, \pi_2);
\end{align*}
\]

\[ \kappa := \text{KernelEmbedding}(\varphi) \]

\[ \gamma := \iota_1 \circ \pi_1 \circ \kappa \]
\[ \pi_i := \text{ProjectionInFactorOfDirectSum}\left((M_1, M_2), i\right), \ i = 1, 2 \]

\[ \pi_1 := \text{ProjectionInFactorOfDirectSum}(\ [ M_1, M_2 \ ], 1 ); \]
\[ \pi_2 := \text{ProjectionInFactorOfDirectSum}(\ [ M_1, M_2 \ ], 2 ); \]

\[ \varphi := \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2 \]

\[ \lambda := \text{PostCompose}(\ \iota_1, \pi_1 \ ); \]
\[ \phi := \lambda - \text{PostCompose}(\ \iota_2, \pi_2 \ ); \]

\[ \kappa := \text{KernelEmbedding}\ (\varphi) \]

\[ \kappa := \text{KernelEmbedding}(\ \phi \ ); \]

\[ \gamma := \iota_1 \circ \pi_1 \circ \kappa \]
\( \pi_i := \text{ProjectionInFactorOfDirectSum}((M_1, M_2), i), i = 1, 2 \)

\[
\begin{align*}
\pi_1 &:= \text{ProjectionInFactorOfDirectSum}( [ M_1, M_2 ], 1 ); \\
\pi_2 &:= \text{ProjectionInFactorOfDirectSum}( [ M_1, M_2 ], 2 ); \\
\varphi &:= \iota_1 \circ \pi_1 - \iota_2 \circ \pi_2 \\
\lambda &:= \text{PostCompose}( \iota_1, \pi_1 ); \\
\phi &:= \lambda - \text{PostCompose}( \iota_2, \pi_2 ); \\
\kappa &:= \text{KernelEmbedding}( \varphi ) \\
\kappa &:= \text{KernelEmbedding}( \phi ); \\
\gamma &:= \iota_1 \circ \pi_1 \circ \kappa \\
\Gamma &:= \text{PostCompose}( \lambda, \kappa );
\end{align*}
\]
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

\[
\begin{align*}
\pi_1 & := \text{ProjectionInFactorOfDirectSum}( \mathcal{M}_1, \mathcal{M}_2, 1 ); \\
\pi_2 & := \text{ProjectionInFactorOfDirectSum}( \mathcal{M}_1, \mathcal{M}_2, 2 ); \\
\lambda & := \text{PostCompose}( \iota_1, \pi_1 ); \\
\phi & := \lambda - \text{PostCompose}( \iota_2, \pi_2 ); \\
\kappa & := \text{KernelEmbedding}( \phi ); \\
\gamma & := \text{PostCompose}( \lambda, \kappa );
\end{align*}
\]
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;
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 );
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

\textbf{In [1]} := \texttt{Integrate[ArcTan[x] - ArcCot[1/x], \{x, 0, 1\}]
\texttt{Out[1]} = 0

\textbf{In [2]} := \texttt{Integrate[ArcTan[x] - ArcCot[1/x], \{x, 0, 1.0\}]
\texttt{Out[2]} = -7.88258 \times 10^{-15}
Integrate $\arctan(x) - \arccot(1/x)$ from 0 to 1.

- Out[1] = 0

Integrate $\arctan(x) - \arccot(1/x)$ from 0 to 1.0.


FullSimplify $\arctan(x) - \arccot(1/x)$.

- Out[3] = 0
\texttt{a = 1}
\texttt{b = 2}
\texttt{c = -3}

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

\[
\begin{align*}
&y^6 + \frac{1}{3}x^2y^3 - \frac{1}{3}x^2y^2 + y^4 \\
&- \frac{1}{3}x^2 + \frac{2}{3}y, \\
x^5 + 3y^4 + x^2y + 6x*y^2 + 3y^2, \\
x^3y^2 + y^2 + 2x + 1, \\
x*y^4 - \frac{1}{3}x^3 - \frac{1}{3}y
\end{align*}
\]
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
```