Software Development within the SPP1489: Number Theory

Claus Fieker
and
Bill Hart, Tommy Hofmann
and
others

September 29, 2015

Computer algebra is incomplete without number theory.
Applications: (small selection)

- absolute factorisation
- varieties over $\mathbb{C}$
- group representations and character theory
- residue class fields

Number fields are fundamental base rings for an enormous range of applications!

Computer algebra is incomplete without number theory.
Applications: (small selection)

- absolute factorisation
- varieties over $\mathbb{C}$
- group representations and character theory
- residue class fields

Number fields are fundamental base rings for an enormous range of applications!

Computer algebra is incomplete without number theory.
Number theory ala Zassenhaus:

- ring of integers (normalisation)
- unit group
- class group (ideal structure)
- Galois group (automorphisms)

Number theory is also veryactive research in its own right.

- class field theory, Langlands
- modular forms
- arithmetic geometry

Computer algebra is incomplete without number theory.
Number theory ala Zassenhaus:

- ring of integers (normalisation)
- unit group
- class group (ideal structure)
- Galois group (automorphisms)

Number theory is also veryactive research in its own right.

- class field theory, Langlands
- modular forms
- arithmetic geometry

Fact

We cannot do everything in one project

$\Longrightarrow$

Goal

Class groups

Automatically including maximal order and unit group as well and element arithmetic, ideal arithmetic, residue class fields, conjugates, logarithms, ...
All of Zassenhaus but Galois groups.

Fact

We cannot do everything in one project

$$\Longrightarrow$$

Goal

Class groups

Automatically including maximal order and unit group as well and element arithmetic, ideal arithmetic, residue class fields, conjugates, logarithms, . . .
All of Zassenhaus but Galois groups.

Fact

We cannot do everything in one project

$$\Longrightarrow$$

Goal

Class groups

Automatically including maximal order and unit group as well and element arithmetic, ideal arithmetic, residue class fields, conjugates, logarithms, ...
All of Zassenhaus but Galois groups.

Fact

We cannot do everything in one project

$$\Longrightarrow$$

Goal

Class groups

Automatically including maximal order and unit group as well and element arithmetic, ideal arithmetic, residue class fields, conjugates, logarithms, ...

All of Zassenhaus but Galois groups.

Fact

We cannot do everything in one project

$$\Longrightarrow$$

Goal

Class groups

Automatically including maximal order and unit group as well and element arithmetic, ideal arithmetic, residue class fields, conjugates, logarithms, . . .
All of Zassenhaus but Galois groups.

Why?

- need generic rings that are designed for speed (see benchmarks on `www.nemocas.org`)
- programming in C/ C++ is cumbersome and extremely error prone
- want an interactive environment that can be supplemented easily in C
- establish a mid-level programming language for computer algebra, somewhere between Gap and C
- need stable, fast number theory (library) for a number of other projects

Why?

- need generic rings that are designed for speed (see benchmarks on `www.nemocas.org`)
- programming in C/ C++ is cumbersome and extremely error prone
- want an interactive environment that can be supplemented easily in C
- establish a mid-level programming language for computer algebra, somewhere between Gap and C
- need stable, fast number theory (library) for a number of other projects

Why?

- need generic rings that are designed for speed (see benchmarks on www.nemocas.org)
- programming in C/ C++ is cumbersome and extremely error prone
- want an interactive environment that can be supplemented easily in C
- establish a mid-level programming language for computer algebra, somewhere between Gap and C
- need stable, fast number theory (library) for a number of other projects

Why?

- need generic rings that are designed for speed (see benchmarks on `www.nemocas.org`)
- programming in C/ C++ is cumbersome and extremely error prone
- want an interactive environment that can be supplemented easily in C
- establish a mid-level programming language for computer algebra, somewhere between Gap and C
- need stable, fast number theory (library) for a number of other projects

Why?

- need generic rings that are designed for speed (see benchmarks on www.nemocas.org)
- programming in C/ C++ is cumbersome and extremely error prone
- want an interactive environment that can be supplemented easily in C
- establish a mid-level programming language for computer algebra, somewhere between Gap and C
- need stable, fast number theory (library) for a number of other projects

Why?

- need generic rings that are designed for speed (see benchmarks on www.nemocas.org)
- programming in C/ C++ is cumbersome and extremely error prone
- want an interactive environment that can be supplemented easily in C
- establish a mid-level programming language for computer algebra, somewhere between Gap and C
- need stable, fast number theory (library) for a number of other projects

Goal

By next summer competitive, fast class groups in large (degree) fields

Have

So far: fastest available number fields and ideals, skeleton class group and Euler-product with provable errors, maximal order, sparse linear algebra, lattice enumeration and reduction, . . .

Goal

By next summer competitive, fast class groups in large (degree) fields

Have

So far: fastest available number fields and ideals, skeleton class group and Euler-product with provable errors, maximal order, sparse linear algebra, lattice enumeration and reduction, . . .

Goal

By next summer competitive, fast class groups in large (degree) fields

Have

So far: fastest available number fields and ideals, skeleton class group and Euler-product with provable errors, maximal order, sparse linear algebra, lattice enumeration and reduction, ...

The Team:

Bill Hart: $\Rightarrow$ clean, high-performance code (see Flint), reusable for as many projects as possible

Me: $\Rightarrow$ do *not* want to use (pure) C/C++, want interactive shell, not library, can tolerate small portions of C for speed.

Tommy Hofmann: $\Rightarrow$ clean code for a shortest path to class groups

The Team:

Bill Hart: ⇒ clean, high-performance code (see Flint), reusable for as many projects as possible

Me: ⇒ do *not* want to use (pure) C/C++, want interactive shell, not library, can tolerate small portions of C for speed.

Tommy Hofmann: ⇒ clean code for a shortest path to class groups

The Team:

Bill Hart: ⇒ clean, high-performance code (see Flint), reusable for as many projects as possible

Me: ⇒ do *not* want to use (pure) C/C++, want interactive shell, not library, can tolerate small portions of C for speed.

Tommy Hofmann: ⇒ clean code for a shortest path to class groups

The Team:

Bill Hart: $\Rightarrow$ clean, high-performance code (see Flint), reusable for as many projects as possible

Me: $\Rightarrow$ do *not* want to use (pure) C/C++, want interactive shell, not library, can tolerate small portions of C for speed.

Tommy Hofmann: $\Rightarrow$ clean code for a shortest path to class groups

The Team:

Bill Hart: ⇒ clean, high-performance code (see Flint), reusable for as many projects as possible

Me: ⇒ do *not* want to use (pure) C/C++, want interactive shell, not library, can tolerate small portions of C for speed.

Tommy Hofmann: ⇒ clean code for a shortest path to class groups

The Team:

Bill Hart: ⇒ clean, high-performance code (see Flint), reusable for as many projects as possible

Me: ⇒ do *not* want to use (pure) C/C++, want interactive shell, not library, can tolerate small portions of C for speed.

Tommy Hofmann: ⇒ clean code for a shortest path to class groups

The Team:

Bill Hart: ⇒ clean, high-performance code (see Flint), reusable for as many projects as possible

Me: ⇒ do *not* want to use (pure) C/C++, want interactive shell, not library, can tolerate small portions of C for speed.

Tommy Hofmann: ⇒ clean code for a shortest path to class groups

Results I:

- ANTIC: a small, extremely fast C-library for basic arithmetic in number fields (7kloc)
- NEMO: a Julia-package wrapping Flint and antic into the Julia-ecosystem, recursive rings, matrices (20kloc)
- HECKE: a Julia-package for the "missing" number theory (13kloc)

growing

Results I:

- ANTIC: a small, extremely fast C-library for basic arithmetic in number fields (7kloc)
- NEMO: a Julia-package wrapping Flint and antic into the Julia-ecosystem, recursive rings, matrices (20kloc)
- HECKE: a Julia-package for the “missing” number theory (13kloc)

growing

Results I:

- ANTIC: a small, extremely fast C-library for basic arithmetic in number fields (7kloc)
- NEMO: a Julia-package wrapping Flint and antic into the Julia-ecosystem, recursive rings, matrices (20kloc)
- HECKE: a Julia-package for the “missing” number theory (13kloc)

growing

Results I:

- ANTIC: a small, extremely fast C-library for basic arithmetic in number fields (7kloc)
- NEMO: a Julia-package wrapping Flint and antic into the Julia-ecosystem, recursive rings, matrices (20kloc)
- HECKE: a Julia-package for the "missing" number theory (13kloc)

growing

Results I:

- ANTIC: a small, extremely fast C-library for basic arithmetic in number fields (7kloc)
- NEMO: a Julia-package wrapping Flint and antic into the Julia-ecosystem, recursive rings, matrices (20kloc)
- HECKE: a Julia-package for the "missing" number theory (13kloc)

growing

Results II:

Have

ANTIC *is the fastest library for arithmetic in number fields available - faster than Magma and pari.*

Have

Generic, recursive, rings in NEMO *are faster than everything else (Sage, Magma, pari)*

Have

Ideal arithmetic in HECKE *is orders of magnitude faster than Magma and pari*

Results II:

Have

ANTIC *is the fastest library for arithmetic in number fields available - faster than Magma and pari.*

Have

Generic, recursive, rings in NEMO *are faster than everything else (Sage, Magma, pari)*

Have

Ideal arithmetic in HECKE *is orders of magnitude faster than Magma and pari*

Results II:

Have

ANTIC *is the fastest library for arithmetic in number fields available - faster than Magma and pari.*

Have

Generic, recursive, rings in NEMO *are faster than everything else (Sage, Magma, pari)*

Have

Ideal arithmetic in HECKE *is orders of magnitude faster than Magma and pari*

Julia:

- JIT (Just-In-Time) compiled language
- interactive shell
- modern language features
- fast on trivial operations
- easy integration of existing C-libraries
- (easy) access to C++-libraries
- garbage collected
- (strongly) typed

What is actually there?

- git repositories for Nemo, hecke and antic
- basic rings: $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{F}_q$, $\mathbb{Q}_p$
- polynomial rings and residue class rings
- linear algebra over basic rings
- number fields
- orders and ideals
- maximal orders and ideals
- factor base and relations for the class group
- Euler product with error guarantees
- processing of unit group
- sparse linear algebra
- lattice enumeration

What is missing (for now)?

- top-level class group
- fast(er) maximal order
- fast(er) factorisation
- quality control: documentation, profiling and testing
- use of subfields, Brauer characters
- p-adic techniques
- parallelisation

Dreaming:

- (good) linear algebra over number fields
- non-commutative orders
- function fields
- class field theory
- strong integration of (Gap, Polymake, Singular) into Julia
- strong integration of Julia into Gap, Polymake and Singular
- Galois groups and automorphisms
- Galois cohomology
- representation theory
- the rest

- Julia: `http://www.julialang.org`
- Nemo: `http://www.nemocas.org`
- hecke `http://github.com/thofma/hecke`

Julia:

```
julia> using hecke
julia> K, a = MaximalRealSubfield(43, "a")
julia> O = hecke.NfMaximalOrder(K,
         hecke.FakeFmpqMat(MatrixSpace(ZZ, 21, 21)(1),
              fmpz(1)));
julia> set_verbose_level(:ClassGroup, 2)
julia> L = lll(O)
julia> @time c = hecke.class_group(L, bound = 200,
                                         method=1);
```