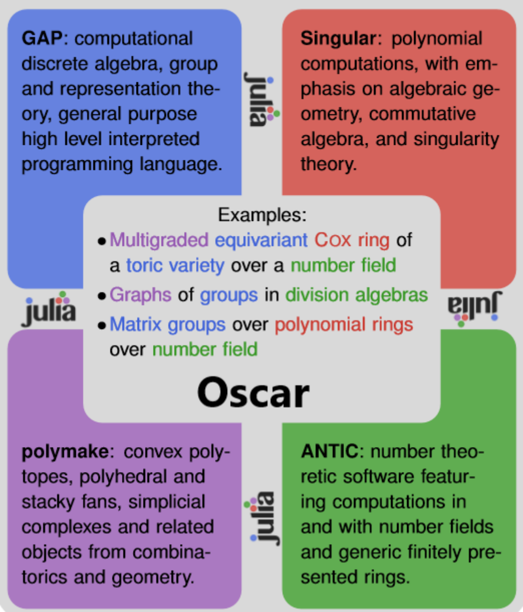
Oscar/Nemo.jl: A Julia package for computer algebra

Claus Fieker, William Hart,
Tommy Hofmann, Fredrik Johansson,
Marek Kaluba

March 6, 2018


GAP: computational discrete algebra, group and representation theory, general purpose high level interpreted programming language.
Singular: polynomial computations, with emphasis on algebraic geometry, commutative algebra, and singularity theory.
Examples:
• Multigraded equivariant Cox ring of a toric variety over a number field
• Graphs of groups in division algebras
• Matrix groups over polynomial rings over number field
Oscar
polymake: convex polytopes, polyhedral and stacky fans, simplicial complexes and related objects from combinatorics and geometry.
ANTIC: number theoretic software featuring computations in and with number fields and generic finitely presented rings.
julia
julia
julia
julia

Oscar components : cornerstone systems

- Gap : Group theory (discrete algebra)
- Singular : Polynomials, algebra, geometry
- Polymake : Polyhedreal geometry
- Antic : Algebraic number theory

Oscar components : cornerstone systems

- Gap : Group theory (discrete algebra)
- Singular : Polynomials, algebra, geometry
- Polymake : Polyhedreal geometry
- Antic : Algebraic number theory

Oscar components : Julia packages

- AbstractAlgebra.jl : generic algorithms

Oscar components : Julia packages

- AbstractAlgebra.jl : generic algorithms
- Nemo.jl : wrappers of Flint, Arb and Antic C libraries

Oscar components : Julia packages

- AbstractAlgebra.jl : generic algorithms
- Nemo.jl : wrappers of Flint, Arb and Antic C libraries
- Hecke.jl : Algebraic number theory, class field theory

Oscar components : Julia packages

- AbstractAlgebra.jl : generic algorithms
- Nemo.jl : wrappers of Flint, Arb and Antic C libraries
- Hecke.jl : Algebraic number theory, class field theory
- Singular.jl : wrapper of the Singular kernel

Oscar components : Julia packages

- AbstractAlgebra.jl : generic algorithms
- Nemo.jl : wrappers of Flint, Arb and Antic C libraries
- Hecke.jl : Algebraic number theory, class field theory
- Singular.jl : wrapper of the Singular kernel
- libGap.jl : interface to Gap from Julia

Oscar components : Julia packages

- AbstractAlgebra.jl : generic algorithms
- Nemo.jl : wrappers of Flint, Arb and Antic C libraries
- Hecke.jl : Algebraic number theory, class field theory
- Singular.jl : wrapper of the Singular kernel
- libGap.jl : interface to Gap from Julia
- Polymake.jl : interface to Polymake from Julia

Oscar components : Julia packages

- AbstractAlgebra.jl : generic algorithms
- Nemo.jl : wrappers of Flint, Arb and Antic C libraries
- Hecke.jl : Algebraic number theory, class field theory
- Singular.jl : wrapper of the Singular kernel
- libGap.jl : interface to Gap from Julia
- Polymake.jl : interface to Polymake from Julia

- JuliaInterface : Access to Julia from Gap

Antic cornerstone

C libraries:

- Flint - polynomials and linear algebra

Antic cornerstone

C libraries:

- Flint - polynomials and linear algebra
- Antic - number field arithmetic

Antic cornerstone

C libraries:

- Flint - polynomials and linear algebra
- Antic - number field arithmetic
- Arb - real and complex ball arithmetic

Antic cornerstone

C libraries:

- Flint - polynomials and linear algebra
- Antic - number field arithmetic
- Arb - real and complex ball arithmetic

Julia libraries:

- Nemo.jl - wrappers of the C libraries

Antic cornerstone

C libraries:

- Flint - polynomials and linear algebra
- Antic - number field arithmetic
- Arb - real and complex ball arithmetic

Julia libraries:

- Nemo.jl - wrappers of the C libraries
- AbstractAlgebra.jl - generic rings and fields

Antic cornerstone

C libraries:

- Flint - polynomials and linear algebra
- Antic - number field arithmetic
- Arb - real and complex ball arithmetic

Julia libraries:

- Nemo.jl - wrappers of the C libraries
- AbstractAlgebra.jl - generic rings and fields
- Hecke.jl - number fields, class field theory, algebraic number theory

New features in Flint

- Quadratic sieve integer factorisation

New features in Flint

- Quadratic sieve integer factorisation
- Elliptic curve integer factorisation

New features in Flint

- Quadratic sieve integer factorisation
- Elliptic curve integer factorisation
- APRCL primality test

New features in Flint

- Quadratic sieve integer factorisation
- Elliptic curve integer factorisation
- APRCL primality test
- Parallelised FFT

New features in Flint

- Quadratic sieve integer factorisation
- Elliptic curve integer factorisation
- APRCL primality test
- Parallelised FFT
- Howell form

New features in Flint

- Quadratic sieve integer factorisation
- Elliptic curve integer factorisation
- APRCL primality test
- Parallelised FFT
- Howell form
- Characteristic and minimal polynomial

New features in Flint

- Quadratic sieve integer factorisation
- Elliptic curve integer factorisation
- APRCL primality test
- Parallelised FFT
- Howell form
- Characteristic and minimal polynomial
- van Hoeij factorisation for $\mathbb{Z}[x]$

New features in Flint

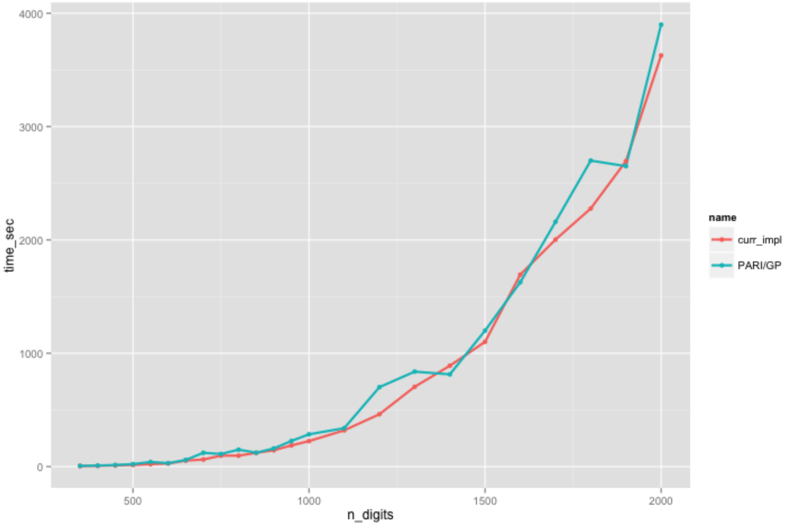
- Quadratic sieve integer factorisation
- Elliptic curve integer factorisation
- APRCL primality test
- Parallelised FFT
- Howell form
- Characteristic and minimal polynomial
- van Hoeij factorisation for $\mathbb{Z}[x]$
- Multivariate polynomial arithmetic over $\mathbb{Z}$, $\mathbb{Z}/n\mathbb{Z}$, $\mathbb{Q}$

Integer factorisation : Quadratic sieve

Table: Quadratic sieve timings

Digits	Pari/GP	Flint (1 core)	Flint (4 cores)
50	0.43	0.55	0.39
59	3.8	3.0	1.7
68	38	21	14
77	257	140	52
83	2200	1500	540

APRCL primality test timings



FFT: Integer and polynomial multiplication

Table: FFT timings

Words	1 core	4 cores	8 cores
110k	0.07s	0.05s	0.05s
360k	0.3s	0.1	0.1s
1.3m	1.1s	0.4s	0.3s
4.6m	4.5s	1.5s	1.0s
26m	28s	9s	6s
120m	140s	48s	33s
500m	800s	240s	150s

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.

Multivariate multiplication

Table: "Dense" Fateman multiply bench

n	Sage	Singular	Magma	Giac	Piranha	Trip	Flint
5	0.0063s	0.0048s	0.0018s	0.00023s	0.0011s	0.00057s	0.00023s
10	0.51s	0.11s	0.12s	0.0056s	0.029s	0.023s	0.0043s
15	9.1s	1.4s	1.9s	0.11s	0.39s	0.21s	0.045s
20	75s	21s	16s	0.62s	2.9s	2.3s	0.48s
25	474s	156s	98s	2.8s	14s	12s	2.3s
30	1667s	561s	440s	14s	56s	41s	10s

4 variables

Multivariate multiplication

Table: Sparse multiply benchmark

n	Sage	Singular	Magma	Giac	Piranha	Trip	Flint
4	0.0066s	0.0050s	0.0062s	0.0046s	0.0033s	0.0015s	0.0014s
6	0.15s	0.11s	0.080s	0.030s	0.025s	0.016s	0.016s
8	1.6s	0.79s	0.68s	0.28s	0.15s	0.10s	0.10s
10	8s	3.6s	3.0s	1.5s	0.62s	0.40s	0.48s
12	43s	14s	11s	4.8s	2.2s	2.2s	2.0s
14	173s	63s	37s	14s	6.7s	12s	7.2s
16	605s	201s	94s	39s	20s	39s	19s

5 variables

Language for generic programming

- Support for Windows, Linux, Mac

Language for generic programming

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats

Language for generic programming

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode

Language for generic programming

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading

Language for generic programming

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading
- Fast generics and metaprogramming

Language for generic programming

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading
- Fast generics and metaprogramming
- Maintained and popular

Language for generic programming

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading
- Fast generics and metaprogramming
- Maintained and popular
- Open source

Language for generic programming

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading
- Fast generics and metaprogramming
- Maintained and popular
- Open source
- Imperative syntax

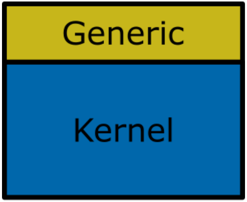
Language for generic programming

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading
- Fast generics and metaprogramming
- Maintained and popular
- Open source
- Imperative syntax
- Garbage collected

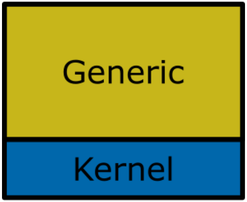
Language for generic programming

- Support for Windows, Linux, Mac
- 64 bit integers and double precision floats
- Console/REPL mode
- Operator overloading
- Fast generics and metaprogramming
- Maintained and popular
- Open source
- Imperative syntax
- Garbage collected
- Easy/efficient C interop

Efficient generics

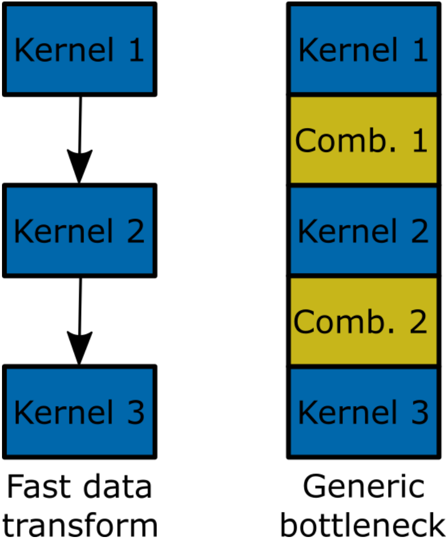


Fast generics



Slow generics

Efficient generics



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.

AbstractAlgebra/Nemo capabilities:

- residue rings

AbstractAlgebra/Nemo capabilities:

- residue rings
- fraction fields

AbstractAlgebra/Nemo capabilities:

- residue rings
- fraction fields
- dense univariate polynomials

AbstractAlgebra/Nemo capabilities:

- residue rings
- fraction fields
- dense univariate polynomials
- dense linear algebra

AbstractAlgebra/Nemo capabilities:

- residue rings
- fraction fields
- dense univariate polynomials
- dense linear algebra
- power series

New features of AbstractAlgebra/Nemo:

- Residue fields

New features of AbstractAlgebra/Nemo:

- Residue fields
- Laurent series

New features of AbstractAlgebra/Nemo:

- Residue fields
- Laurent series
- Generic HNF, SNF, Popov form

New features of AbstractAlgebra/Nemo:

- Residue fields
- Laurent series
- Generic HNF, SNF, Popov form
- Generic multivariate polynomials

New features of AbstractAlgebra/Nemo:

- Residue fields
- Laurent series
- Generic HNF, SNF, Popov form
- Generic multivariate polynomials
- Capped absolute power series

New features of AbstractAlgebra/Nemo:

- Residue fields
- Laurent series
- Generic HNF, SNF, Popov form
- Generic multivariate polynomials
- Capped absolute power series
- Permutation groups, Young tableaux, characters

New features of AbstractAlgebra/Nemo:

- Residue fields
- Laurent series
- Generic HNF, SNF, Popov form
- Generic multivariate polynomials
- Capped absolute power series
- Permutation groups, Young tableaux, characters
- Ducos' algorithm for resultants

New features of AbstractAlgebra/Nemo:

- Residue fields
- Laurent series
- Generic HNF, SNF, Popov form
- Generic multivariate polynomials
- Capped absolute power series
- Permutation groups, Young tableaux, characters
- Ducos' algorithm for resultants
- Rewritten documentation

New features of AbstractAlgebra/Nemo:

- Residue fields
- Laurent series
- Generic HNF, SNF, Popov form
- Generic multivariate polynomials
- Capped absolute power series
- Permutation groups, Young tableaux, characters
- Ducos' algorithm for resultants
- Rewritten documentation
- Integration with Singular.jl

Appplication: modular equations

- My thesis was on computing abelian extension of number fields via modular equations:

Appplication: modular equations

- My thesis was on computing abelian extension of number fields via modular equations:
- Relation $P_n(A(\tau), A(n\tau)) = 0$ for modular function $A(\tau)$ and polynomial $P_n(X, Y)$, for all τ in complex upper half plane

Appplication: modular equations

- My thesis was on computing abelian extension of number fields via modular equations:
- Relation $P_n(A(\tau), A(n\tau)) = 0$ for modular function $A(\tau)$ and polynomial $P_n(X, Y)$, for all τ in complex upper half plane
- Can specialise at certain values of τ to give "small" generating polynomials for extensions of number fields

Appplication: modular equations

- My thesis was on computing abelian extension of number fields via modular equations:
- Relation $P_n(A(\tau), A(n\tau)) = 0$ for modular function $A(\tau)$ and polynomial $P_n(X, Y)$, for all τ in complex upper half plane
- Can specialise at certain values of τ to give "small" generating polynomials for extensions of number fields
- Example: Klein j-function is a modular function

Appplication: modular equations

- My thesis was on computing abelian extension of number fields via modular equations:
- Relation $P_n(A(\tau), A(n\tau)) = 0$ for modular function $A(\tau)$ and polynomial $P_n(X, Y)$, for all τ in complex upper half plane
- Can specialise at certain values of τ to give "small" generating polynomials for extensions of number fields
- Example: Klein j-function is a modular function
- Periodic with period 1, so has Fourier expansion, called a q-expansion:

Appplication: modular equations

- My thesis was on computing abelian extension of number fields via modular equations:
- Relation $P_n(A(\tau), A(n\tau)) = 0$ for modular function $A(\tau)$ and polynomial $P_n(X, Y)$, for all τ in complex upper half plane
- Can specialise at certain values of τ to give "small" generating polynomials for extensions of number fields
- Example: Klein j-function is a modular function
- Periodic with period 1, so has Fourier expansion, called a q-expansion:
- $j(\tau) = q^{-1} + 744 + 196884q + 21493760q^2 + \cdots$, where $q = exp(2\pi i\tau)$

Modular equation example

- For $n = 2$: $P_2(j(\tau), j(2\tau)) = 0$ for

$$P_2(X,Y) = X^3 - X^2Y2 + 1488X^2Y^2 - 162000X^2 + 1488XY^2$$
$$40773374XY + 8748000000X + Y^2 - 162000Y^2 + 8748000000Y$$
$$- 157464000000000.$$

Dedekind eta function

- $\eta(\tau) = q^{1/24} \prod_{n=1}^{\infty} (1 - q^n)$.

Dedekind eta function

- $\eta(\tau) = q^{1/24} \prod_{n=1}^{\infty}(1 - q^n)$.
- Not a modular function, but quotients of them are

Dedekind eta function

- $\eta(\tau) = q^{1/24} \prod_{n=1}^{\infty} (1 - q^n)$.
- Not a modular function, but quotients of them are
- Enter the Weber functions:

$$\mathfrak{f}(\tau) = \frac{\eta^2(\tau)}{\eta(\tau/2)\eta(2\tau)},$$

$$\mathfrak{f}_1(\tau) = \frac{\eta(\tau/2)}{\eta(\tau)},$$

$$\mathfrak{f}_2(\tau) = \frac{\sqrt{2}\eta(2\tau)}{\eta(\tau)}.$$

Modular equations for Weber functions

- Classically modular equations are computed for primes $n > 2$

Modular equations for Weber functions

- Classically modular equations are computed for primes $n > 2$
- For example, for $n = 13$ we have $\Phi_{13}(\mathfrak{f}(\tau), \mathfrak{f}(13\tau)) = 0$ where

$$\Phi_{13}(X, Y) = X^{14} - X^{13}Y^{13} + 13X^{12}Y^2 + 52X^{10}Y^4$$
$$+ 78X^8Y^6 + 78X^6Y^8 + 52X^4Y^{10} + 13X^2Y^{12} + 64XY + Y^{14}.$$

Even degree modular equations

- Classically the theory depends on the "degree" n being coprime to 2

Even degree modular equations

- Classically the theory depends on the "degree" n being coprime to 2
- Modular equations of every degree exist, they are just hard to find

Even degree modular equations

- Classically the theory depends on the "degree" n being coprime to 2
- Modular equations of every degree exist, they are just hard to find
- We would like to find modular equations of even degree n

$n = 2$

- Naive strategy:

$n = 2$

- Naive strategy:
- compute $A = \mathfrak{f}_1(\tau)$ and $B = \mathfrak{f}_1(2\tau)$ for a random τ in the upper half plane

$n = 2$

- Naive strategy:
- compute $A = \mathfrak{f}_1(\tau)$ and $B = \mathfrak{f}_1(2\tau)$ for a random τ in the upper half plane
- Find $\mathbb{Z}$-linear combination of terms $A^i B^j$ equal to zero

Finding linear combinations

- The function modeta in Nemo/Arb computes the eta function in the upper half plane

Finding linear combinations

- The function modeta in Nemo/Arb computes the eta function in the upper half plane
- We add Weber functions to Nemo, e.g.

```
function f1(x::acb)
   return divexact(modeta(x/2), modeta(x))
end
```

Finding linear combinations

- The function modeta in Nemo/Arb computes the eta function in the upper half plane
- We add Weber functions to Nemo, e.g.

```
function f1(x::acb)
  return divexact(modeta(x/2), modeta(x))
end
```

- We can find small linear combinations of terms $x_{i,j} = \mathfrak{f}_1(\tau)^i \mathfrak{f}_1(2\tau)^j$ using LLL

Finding linear combinations

- The function modeta in Nemo/Arb computes the eta function in the upper half plane
- We add Weber functions to Nemo, e.g.

```
function f1(x::acb)
   return divexact(modeta(x/2), modeta(x))
end
```

- We can find small linear combinations of terms $x_{i,j} = \mathfrak{f}_1(\tau)^i \mathfrak{f}_1(2\tau)^j$ using LLL
- We LLL reduce the matrix

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 & \mathcal{R}(x_{0,0}) & \mathcal{I}(x_{0,0}) \\ 0 & 1 & 0 & \dots & 0 & \mathcal{R}(x_{0,1}) & \mathcal{I}(x_{0,1}) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & \dots & 1 & \mathcal{R}(x_{m,n}) & \mathcal{I}(x_{m,n}) \end{pmatrix}$$

LLL reduction

```
function lindep(V::Array{acb}, bits::Int)
  n = length(V)
  W = [ldexp(s, bits) for s in V]
  M = zero_matrix(ZZ, n, n + 2)
  for i = 1:n
    M[i, i] = ZZ(1)
    M[i, n + 1] = unique_integer(floor(real(W[i]) + 0.5))
    M[i, n + 2] = unique_integer(floor(imag(W[i]) + 0.5))
  end
  L = lll(M)
  return [L[1, i] for i = 1:n]
end
```

Modular equation of degree 2

- Fails miserably: simply too many terms

Modular equation of degree 2

- Fails miserably: simply too many terms
- Works if we take $(A^8)^i(B^8)^j$ instead

Modular equation of degree 2

- Fails miserably: simply too many terms
- Works if we take $(A^8)^i(B^8)^j$ instead
- $$\Phi_2(X, Y) = X^2Y + 16X - Y^2$$

Computing modular equations

```
CC = ComplexField(128)
tau = CC(rand(), abs(rand()))
A = modweber_f1(tau)^8; B = modweber_f1(2*tau)^8

vals = [A^i*B^j for i in 0:2 for j in 0:2];
C = lindep(vals, 100)

R, (x, y) = PolynomialRing(ZZ, ["x", "y"])
Phi = sum([C[3*i+j+1]*x^i*y^j for i in 0:2 for j in 0:2])
```

Modular equation of degree 4

- Same polynomial $\Phi_2(X, Y)$ relates $\mathfrak{f}_1(\tau)^8$ and $\mathfrak{f}_1(2\tau)^8$ as relates $\mathfrak{f}_1(2\tau)^8$ and $\mathfrak{f}_1(4\tau)^8$.

Modular equation of degree 4

- Same polynomial $\Phi_2(X, Y)$ relates $\mathfrak{f}_1(\tau)^8$ and $\mathfrak{f}_1(2\tau)^8$ as relates $\mathfrak{f}_1(2\tau)^8$ and $\mathfrak{f}_1(4\tau)^8$.
- Take the resultant of $yz^2 + 16z - y^2$ and $zx^2 + 16x - z^2$ and eliminate z:

Modular equation of degree 4

- Same polynomial $\Phi_2(X, Y)$ relates $\mathfrak{f}_1(\tau)^8$ and $\mathfrak{f}_1(2\tau)^8$ as relates $\mathfrak{f}_1(2\tau)^8$ and $\mathfrak{f}_1(4\tau)^8$.
- Take the resultant of $yz^2 + 16z - y^2$ and $zx^2 + 16x - z^2$ and eliminate z:
- $\Phi_4(X, Y) =$
  $Y^4 - X^4Y^3 - 32XY^3 + 240X^2Y^2 - 256X^3Y - 4096X$

Modular equation of degree 4

- Same polynomial $\Phi_2(X, Y)$ relates $\mathfrak{f}_1(\tau)^8$ and $\mathfrak{f}_1(2\tau)^8$ as relates $\mathfrak{f}_1(2\tau)^8$ and $\mathfrak{f}_1(4\tau)^8$.
- Take the resultant of $yz^2 + 16z - y^2$ and $zx^2 + 16x - z^2$ and eliminate z:
- $\Phi_4(X, Y) =$
  $Y^4 - X^4Y^3 - 32XY^3 + 240X^2Y^2 - 256X^3Y - 4096X$
- Is there a relation between lower powers of $\mathfrak{f}_1(\tau)$ and $\mathfrak{f}_1(4\tau)$?

Modular equation of degree 4

- Same polynomial $\Phi_2(X, Y)$ relates $\mathfrak{f}_1(\tau)^8$ and $\mathfrak{f}_1(2\tau)^8$ as relates $\mathfrak{f}_1(2\tau)^8$ and $\mathfrak{f}_1(4\tau)^8$.
- Take the resultant of $yz^2 + 16z - y^2$ and $zx^2 + 16x - z^2$ and eliminate z:
- $\Phi_4(X, Y) =$
  $Y^4 - X^4Y^3 - 32XY^3 + 240X^2Y^2 - 256X^3Y - 4096X$
- Is there a relation between lower powers of $\mathfrak{f}_1(\tau)$ and $\mathfrak{f}_1(4\tau)$?
- Can proceed in the same way for degree $2n$, but the equations are enormous!

Weber modular equations

- Another method is via Weber modular equations and some identities

Weber modular equations

- Another method is via Weber modular equations and some identities
- For example, for $n = 3$ Weber has:

$$\mathfrak{f}(\tau)^2\mathfrak{f}(3\tau)^2 = \mathfrak{f}_2(\tau)^2\mathfrak{f}_2(3\tau)^2 + \mathfrak{f}_1(\tau)^2\mathfrak{f}_1(3\tau)^2$$

Weber modular equations

- Another method is via Weber modular equations and some identities
- For example, for $n = 3$ Weber has:

$$\mathfrak{f}(\tau)^2\mathfrak{f}(3\tau)^2 = \mathfrak{f}_2(\tau)^2\mathfrak{f}_2(3\tau)^2 + \mathfrak{f}_1(\tau)^2\mathfrak{f}_1(3\tau)^2$$

- Can use the identities

$$\mathfrak{f}(\tau)^2\mathfrak{f}_1(\tau)^2\mathfrak{f}_2(\tau)^2 = 2$$

and

$$\mathfrak{f}_2(\tau)^2 = 2/\mathfrak{f}_1(2\tau)^2$$

to yield:

Weber modular equations

- Another method is via Weber modular equations and some identities
- For example, for $n = 3$ Weber has:

$$\mathfrak{f}(\tau)^2\mathfrak{f}(3\tau)^2 = \mathfrak{f}_2(\tau)^2\mathfrak{f}_2(3\tau)^2 + \mathfrak{f}_1(\tau)^2\mathfrak{f}_1(3\tau)^2$$

- Can use the identities

$$\mathfrak{f}(\tau)^2\mathfrak{f}_1(\tau)^2\mathfrak{f}_2(\tau)^2 = 2$$

  and

$$\mathfrak{f}_2(\tau)^2 = 2/\mathfrak{f}_1(2\tau)^2$$

  to yield:

- $$\frac{\mathfrak{f}_1(2\tau)^2\mathfrak{f}_1(6\tau)^2}{\mathfrak{f}_1(\tau)^2\mathfrak{f}_1(3\tau)^2} = \frac{4}{\mathfrak{f}_1(2\tau)^2\mathfrak{f}_1(6\tau)^2} + \mathfrak{f}_1(\tau)^2\mathfrak{f}_1(3\tau)^2.$$

Modular equations of degree $2n$

- Can use a similar method to generate modular equations for degree $2n$ for all odd n

Modular equations of degree $2n$

- Can use a similar method to generate modular equations for degree $2n$ for all odd n
- Can even generate a modular equation of degree 4:

Modular equations of degree $2n$

- Can use a similar method to generate modular equations for degree $2n$ for all odd n
- Can even generate a modular equation of degree 4:

$$\mathfrak{f}_1(\tau)^8\mathfrak{f}_1(2\tau)^2\mathfrak{f}_1(4\tau)^4 + 8\mathfrak{f}_1(\tau)^4 - \mathfrak{f}_1(2\tau)^6\mathfrak{f}_1(4\tau)^4 = 0.$$

Modular equations of degree $2n$

- Can use a similar method to generate modular equations for degree $2n$ for all odd n
- Can even generate a modular equation of degree 4:

$$\mathfrak{f}_1(\tau)^8\mathfrak{f}_1(2\tau)^2\mathfrak{f}_1(4\tau)^4 + 8\mathfrak{f}_1(\tau)^4 - \mathfrak{f}_1(2\tau)^6\mathfrak{f}_1(4\tau)^4 = 0.$$

- Which can be rewritten:

$$\frac{\mathfrak{f}_1(\tau)^4}{\mathfrak{f}_1(2\tau)^2} + \frac{8}{\mathfrak{f}_1(2\tau)^4\mathfrak{f}_1(4\tau)^4} = \frac{\mathfrak{f}_1(2\tau)^2}{\mathfrak{f}_1(\tau)^4}.$$

Modular equation of degree 12

- Next logical case is degree 12 modular equation

Modular equation of degree 12

- Next logical case is degree 12 modular equation
- Let's define:

$$B = \frac{\mathfrak{f}_1(2\tau)^2}{\mathfrak{f}_1(6\tau)^2}, \quad A = \frac{\mathfrak{f}_1(\tau)^2}{\mathfrak{f}_1(12\tau)^2}.$$

Modular equation of degree 12

- Next logical case is degree 12 modular equation
- Let's define:

$$B = \frac{\mathfrak{f}_1(2\tau)^2}{\mathfrak{f}_1(6\tau)^2}, \quad A = \frac{\mathfrak{f}_1(\tau)^2}{\mathfrak{f}_1(12\tau)^2}.$$

- No luck with existing methods

Modular equation of degree 12

- Next logical case is degree 12 modular equation
- Let's define:

$$B = \frac{\mathfrak{f}_1(2\tau)^2}{\mathfrak{f}_1(6\tau)^2}, \quad A = \frac{\mathfrak{f}_1(\tau)^2}{\mathfrak{f}_1(12\tau)^2}.$$

- No luck with existing methods
- Idea: try to find modular equations between A, B^2 or A, B^4 or A^2, B or A^4, B or A^2, B^2, etc.

Modular equation of degree 12

- Next logical case is degree 12 modular equation
- Let's define:

$$B = \frac{\mathfrak{f}_1(2\tau)^2}{\mathfrak{f}_1(6\tau)^2}, \quad A = \frac{\mathfrak{f}_1(\tau)^2}{\mathfrak{f}_1(12\tau)^2}.$$

- No luck with existing methods
- Idea: try to find modular equations between A, B^2 or A, B^4 or A^2, B or A^4, B or A^2, B^2, etc.
- We get the modular equation

$$B^{12} + 14B^6 + 8B^3 + 1 = B^9 \left(\frac{B^2}{A^4} + 16\frac{A^4}{B^2} \right).$$

Modular equation of degree 12

- Next logical case is degree 12 modular equation
- Let's define:

$$B = \frac{\mathfrak{f}_1(2\tau)^2}{\mathfrak{f}_1(6\tau)^2}, \quad A = \frac{\mathfrak{f}_1(\tau)^2}{\mathfrak{f}_1(12\tau)^2}.$$

- No luck with existing methods
- Idea: try to find modular equations between A, B^2 or A, B^4 or A^2, B or A^4, B or A^2, B^2, etc.
- We get the modular equation

$$B^{12} + 14B^6 + 8B^3 + 1 = B^9\left(\frac{B^2}{A^4} + 16\frac{A^4}{B^2}\right).$$

- Now there's enough data to guess at a general pattern.

General pattern for degree $4n$

- Define
$$A = \frac{\mathfrak{f}_1(\tau)^2}{\mathfrak{f}_1(4n\tau)^2}, \quad B = \frac{\mathfrak{f}_1(2\tau)^2}{\mathfrak{f}_1(2n\tau)^2}.$$

General pattern for degree $4n$

- Define
$$A = \frac{\mathfrak{f}_1(\tau)^2}{\mathfrak{f}_1(4n\tau)^2}, \quad B = \frac{\mathfrak{f}_1(2\tau)^2}{\mathfrak{f}_1(2n\tau)^2}.$$
- Search for linear combinations of $A^k B^l$ where
$$(8n-2)k + (4n-4)l \equiv m \pmod{24},$$
for fixed m.

Degree 20 modular equation

- For degree 20, $n = 5$

Degree 20 modular equation

- For degree 20, $n = 5$
- Look for combinations $A^k B^l$ where

$$38k + 16l \equiv m \pmod{24}, \quad \text{for fixed} \quad m$$

Degree 20 modular equation

- For degree 20, $n = 5$
- Look for combinations $A^k B^l$ where

$$38k + 16l \equiv m \pmod{24}, \quad \text{for fixed} \quad m$$

- The smallest equation we find is

$$B^{18} + 2B^{15} + 255B^{12} - 580B^9 + 255B^6 - 30B^3 + 1 = 256A^8B^{11} - 256A^4B^7 - 16\frac{B^{11}}{A^4} + \frac{B^{19}}{A^8}$$

```
CC = ComplexField(1000)
tau = CC(rand(), abs(rand()))
A = modweber_f1(tau)^2/modweber_f1(20*tau)^2
B = modweber_f1(2*tau)^2/modweber_f1(10*tau)^2

pairs = [p for p in
   Iterators.filter(x->mod(38*x[1]+16*x[2], 24) == 0,
      (k, l) for k in 0:24 for l in 0:24)]

vals = [A^k*B^l for (k, l) in pairs];
C = lindep(vals, 400)

Phi = sum(C[i]*x^pairs[i][1]*y^pairs[i][2]
    for i in 1:length(pairs))
```

Checking the identities

- Check identities for many random τ

Checking the identities

- Check identities for many random τ
- Can check q-series identities

Checking the identities

- Check identities for many random τ
- Can check q-series identities
- Need Puiseux series

$$\eta(\tau) = q^{1/24} \prod_{n=1}^{\infty} (1 - q^n).$$

Checking the identities

- Check identities for many random τ
- Can check q-series identities
- Need Puiseux series

$$\eta(\tau) = q^{1/24} \prod_{n=1}^{\infty} (1 - q^n).$$

- Nemo has Laurent series

Checking the identities

- Check identities for many random τ
- Can check q-series identities
- Need Puiseux series

$$\eta(\tau) = q^{1/24} \prod_{n=1}^{\infty} (1 - q^n).$$

- Nemo has Laurent series
- Could just scale all exponents by factor of 24

Checking the identities

- Check identities for many random τ
- Can check q-series identities
- Need Puiseux series

$$\eta(\tau) = q^{1/24} \prod_{n=1}^{\infty} (1 - q^n).$$

- Nemo has Laurent series
- Could just scale all exponents by factor of 24
- Chose to implement simple generic Puiseux series

The Puiseux series types

```
mutable struct PuiseuxSeriesRing{T <: RingElement} <: Ring
   laurent_ring::Ring
end

mutable struct PuiseuxSeries{T <: RingElement} <: RingElem
   data::LaurentSeries{T}
   scale::Rational{Int}
   parent::PuiseuxSeriesRing{T}

   function PuiseuxSeries{T}(d::LaurentSeries{T},
            scale::Rational{Int}) where T <: RingElement
      new{T}(d, scale)
   end
end
```

Ring interface

- Must implement the AbstractAlgebra.jl Ring interface

Ring interface

- Must implement the AbstractAlgebra.jl Ring interface
- parent_type, elem_type, base_ring, parent

Ring interface

- Must implement the AbstractAlgebra.jl Ring interface
- parent_type, elem_type, base_ring, parent
- isdomain_type, isexact_type

Ring interface

- Must implement the AbstractAlgebra.jl Ring interface
- parent_type, elem_type, base_ring, parent
- isdomain_type, isexact_type
- hash, deepcopy

Ring interface

- Must implement the AbstractAlgebra.jl Ring interface
- parent_type, elem_type, base_ring, parent
- isdomain_type, isexact_type
- hash, deepcopy
- Constructors, e.g. $R()$, $R(123)$

Ring interface

- Must implement the AbstractAlgebra.jl Ring interface
- parent_type, elem_type, base_ring, parent
- isdomain_type, isexact_type
- hash, deepcopy
- Constructors, e.g. $R()$, $R(123)$
- zero, one, iszero, isone

Ring interface

- Must implement the AbstractAlgebra.jl Ring interface
- parent_type, elem_type, base_ring, parent
- isdomain_type, isexact_type
- hash, deepcopy
- Constructors, e.g. $R()$, $R(123)$
- zero, one, iszero, isone
- canonical_unit

Ring interface

- Must implement the AbstractAlgebra.jl Ring interface
- parent_type, elem_type, base_ring, parent
- isdomain_type, isexact_type
- hash, deepcopy
- Constructors, e.g. $R()$, $R(123)$
- zero, one, iszero, isone
- canonical_unit
- show (for printing) + helpers

Ring interface

- Must implement the AbstractAlgebra.jl Ring interface
- parent_type, elem_type, base_ring, parent
- isdomain_type, isexact_type
- hash, deepcopy
- Constructors, e.g. $R()$, $R(123)$
- zero, one, iszero, isone
- canonical_unit
- show (for printing) + helpers
- arithmetic operations, comparison operators

Ring interface

- Must implement the AbstractAlgebra.jl Ring interface
- parent_type, elem_type, base_ring, parent
- isdomain_type, isexact_type
- hash, deepcopy
- Constructors, e.g. $R()$, $R(123)$
- zero, one, iszero, isone
- canonical_unit
- show (for printing) + helpers
- arithmetic operations, comparison operators
- powering, (exact) quotient

Ring interface

- Must implement the AbstractAlgebra.jl Ring interface
- parent_type, elem_type, base_ring, parent
- isdomain_type, isexact_type
- hash, deepcopy
- Constructors, e.g. $R()$, $R(123)$
- zero, one, iszero, isone
- canonical_unit
- show (for printing) + helpers
- arithmetic operations, comparison operators
- powering, (exact) quotient
- in-place operators

Verifying the q-series identities

```
R, q = PuiseuxSeriesRing(ZZ, 1000, "q")

eta_qexp(q) = prod(1 - q^n for n = 1:50)*q^(1//24)
f1(q) = divexact(eta_qexp(q^(1//2)), eta_qexp(q))

A = divexact(f1(q)^2, f1(q^12)^2)
B = divexact(f1(q^2)^2, f1(q^6)^2)

A^8*(B^18 + 2B^15 + 255B^12 - 580B^9 + 255B^6
- 30B^3 + 1 - 256A^8*B^11 + 256A^4*B^7)
+ 16A^4*B^11 - B^19 == 0
```

Improvements

- Scale Laurent series according to gaps between nonzero terms

Improvements

- Scale Laurent series according to gaps between nonzero terms
- Puiseux series store only denominator of exponents

Improvements

- Scale Laurent series according to gaps between nonzero terms
- Puiseux series store only denominator of exponents
- Use Flint's fast polynomial arithmetic for Laurent series over $\mathbb{Z}$

Improvements

- Scale Laurent series according to gaps between nonzero terms
- Puiseux series store only denominator of exponents
- Use Flint's fast polynomial arithmetic for Laurent series over $\mathbb{Z}$
- Faster eta function q-series

Thank You

http://nemocas.org/