Efficient Gröbner tracing in Groebner.jl

Alexander Demin
École polytechnique, CNRS

International Congress on Mathematical Software
High-Performance Computer Algebra

Waterloo, Ontario, Canada
20-23 July 2026

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Funded by the European Union

European Research Council

Motivation

Earlier (or later) at ICMS 2026:

- Identifiability of Linear Compartmental Models, *Cash Bortner*
- The Euclidean Distance Degree package for Macaulay2, *Will Huang*
- Trajectory Planning and Certification for 3-DOF Robot Manipulators Using Real Quantifier Elimination Based on Comprehensive Gröbner Systems, *Akira Terui et al.*
- Uncertainty Quantification in Parameter Estimation with Noisy Data, *Alexey Ovchinnikov*

Motivation

Earlier (or later) at ICMS 2026:

- Identifiability of Linear Compartmental Models, *Cash Bortner*
- The Euclidean Distance Degree package for Macaulay2, *Will Huang*
- Trajectory Planning and Certification for 3-DOF Robot Manipulators Using Real Quantifier Elimination Based on Comprehensive Gröbner Systems, *Akira Terui et al.*
- Uncertainty Quantification in Parameter Estimation with Noisy Data, *Alexey Ovchinnikov*

A common computational task..

These at some point mention or rely on **repeatedly** solving systems of nonlinear equations.

Motivation

Earlier (or later) at ICMS 2026:

- Identifiability of Linear Compartmental Models, *Cash Bortner*
- The Euclidean Distance Degree package for Macaulay2, *Will Huang*
- Trajectory Planning and Certification for 3-DOF Robot Manipulators Using Real Quantifier Elimination Based on Comprehensive Gröbner Systems, *Akira Terui et al.*
- Uncertainty Quantification in Parameter Estimation with Noisy Data, *Alexey Ovchinnikov*

A common computational task..

These at some point mention or rely on **repeatedly** solving systems of nonlinear equations.

In this talk, Groebner.jl

How to make repeated Gröbner basis computations faster and available?

One standard example from computer algebra

Example, structural identifiability of ODEs [Hong et al., 2019]

Consider the **CRN** example, a system of polynomials in 47 indeterminates:

$$
\begin{aligned}
x_{21} - x_{20}x_{10}k_{10} - x_{40}k_{20} - x_{40}k_{30} &= 0 \\
x_{20} - 544397587489487158925 &= 0 \\
&\dots \\
-6x_{52}x_{32}k_{60} - 4x_{33}x_{51}k_{60} - 4x_{53}x_{31}k_{60} - x_{34}x_{50}k_{60} - \dots &= 0.
\end{aligned}
$$

One standard example from computer algebra

Example, structural identifiability of ODEs [Hong et al., 2019]

Consider the **CRN** example, a system of polynomials in 47 indeterminates:

$$x_{21} - x_{20}x_{10}k_{10} - x_{40}k_{20} - x_{40}k_{30} = 0$$

$$x_{20} - 544397587489487158925 = 0$$

$$\dots$$

$$-6x_{52}x_{32}k_{60} - 4x_{33}x_{51}k_{60} - 4x_{53}x_{31}k_{60} - x_{34}x_{50}k_{60} - \dots = 0.$$

Solving the system

We can find the solution from the Gröbner basis of the system:

$$k_{60} = 850293348699016668 38, \quad k_{50} = 5635268359634276894 29, \quad \dots$$

One standard example from computer algebra

Example, structural identifiability of ODEs [Hong et al., 2019]

Consider the **CRN** example, a system of polynomials in 47 indeterminates:

$$x_{21} - x_{20}x_{10}k_{10} - x_{40}k_{20} - x_{40}k_{30} = 0$$

$$x_{20} - 544397587489487158925 = 0$$

$$\ldots$$

$$-6x_{52}x_{32}k_{60} - 4x_{33}x_{51}k_{60} - 4x_{53}x_{31}k_{60} - x_{34}x_{50}k_{60} - \ldots = 0.$$

Solving the system

We can find the solution from the Gröbner basis of the system:

$$k_{60} = 850293348699016668 38, \quad k_{50} = 563526835963427689429, \quad \ldots$$

Intermediate expression swell

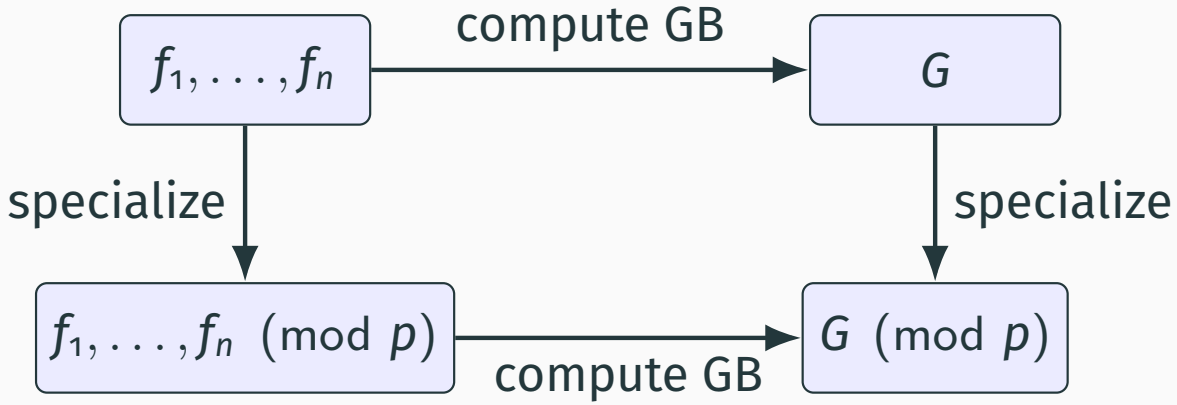
In Gröbner basis in computations there appears a term with coefficient:

389802997212380535243910008992734398822789281692823413123211432594829 24

It contains roughly **135,000 digits**.

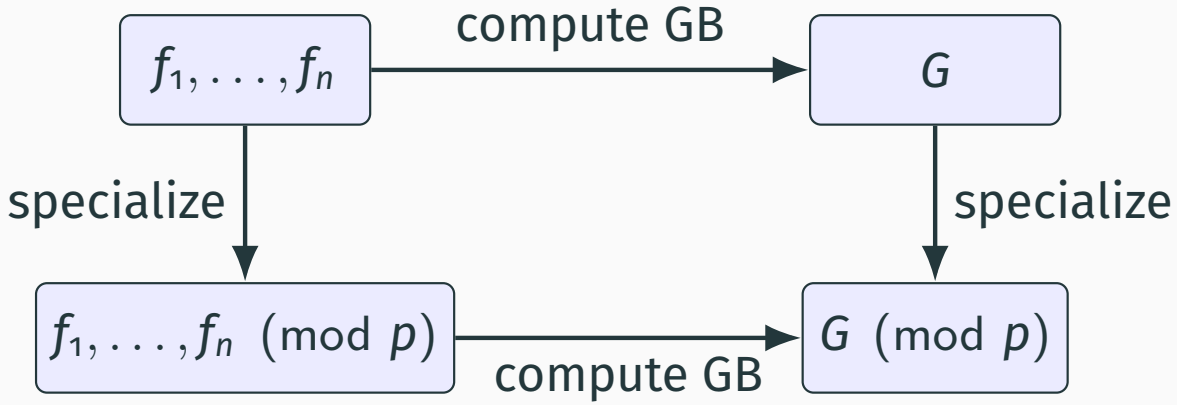
A standard solution: multi-modular methods

Gröbner bases and specialization "almost" commute:



A standard solution: multi-modular methods

Gröbner bases and specialization "almost" commute:



Avoiding intermediate expression swell:

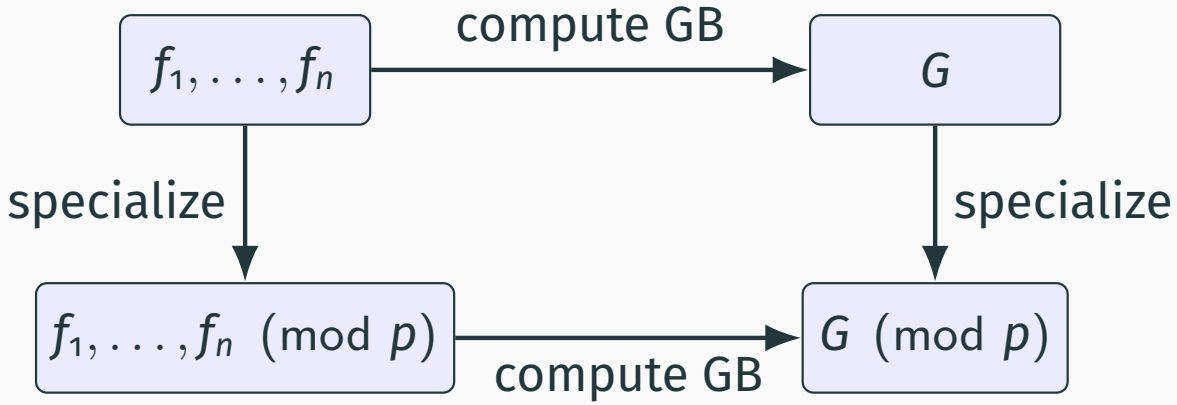
1. Gröbner basis of CRN system

Compute modulo prime $\mathbf{p_1 = 65551}$:

$$k_{60} = 2196, \quad k_{50} = 36593 \pmod{p_1}$$

A standard solution: multi-modular methods

Gröbner bases and specialization "almost" commute:



Avoiding intermediate expression swell:

1. Gröbner basis of CRN system

Compute modulo prime $\mathbf{p_1 = 65551}$:

$$k_{60} = 2196, \quad k_{50} = 36593 \pmod{p_1}$$

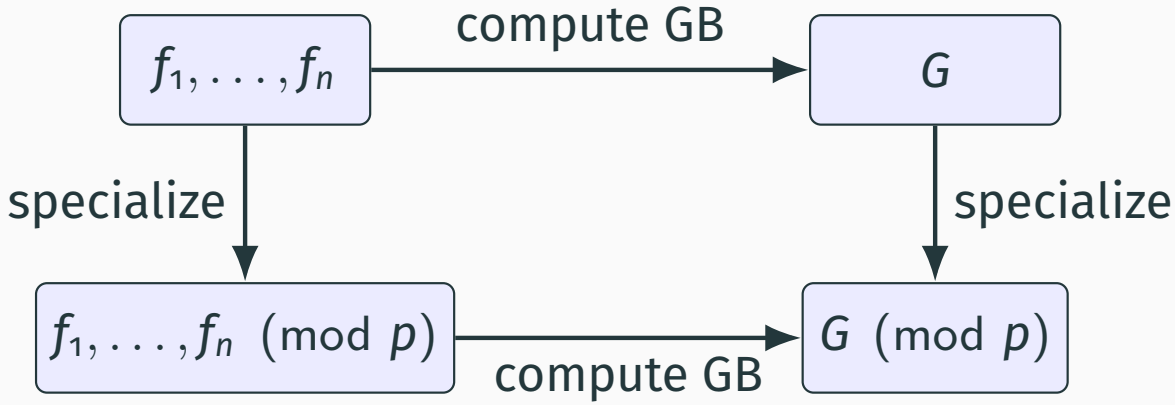
2. Gröbner basis of CRN system

Compute modulo prime $\mathbf{p_2 = 65557}$:

$$k_{60} = 60731, \quad k_{50} = 11098 \pmod{p_2}$$

A standard solution: multi-modular methods

Gröbner bases and specialization "almost" commute:



Avoiding intermediate expression swell:

1. Gröbner basis of CRN system

Compute modulo prime $\mathbf{p_1} = \mathbf{65551}$:

$$k_{60} = 2196, \quad k_{50} = 36593 \pmod{p_1}$$

2. Gröbner basis of CRN system

Compute modulo prime $\mathbf{p_2} = \mathbf{65557}$:

$$k_{60} = 60731, \quad k_{50} = 11098 \pmod{p_2}$$

And then reconstructing the result:

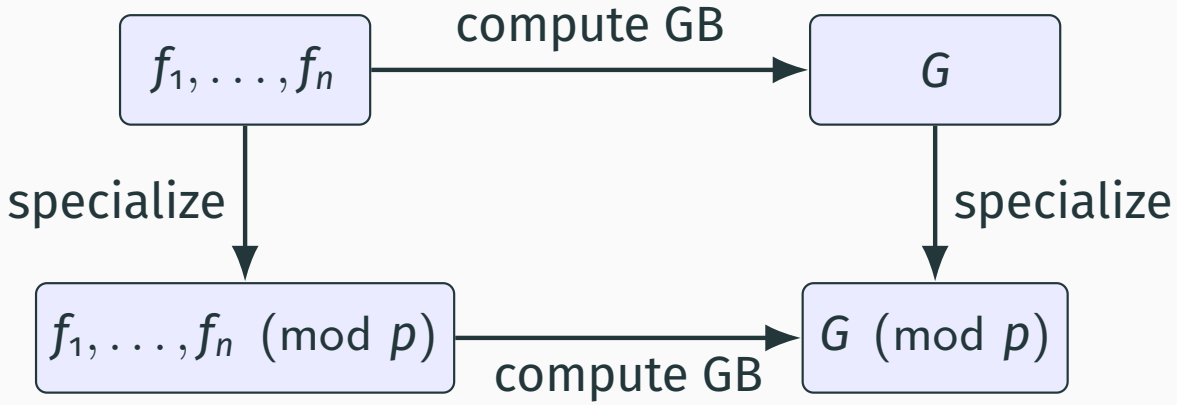
Gröbner basis of CRN system mod $p_1 p_2$

Interpolate via Chinese Remainder Theorem:

$$k_{60} = 2941603321, \quad k_{50} = 3859679473 \mod \mathbf{p_1 p_2}, \quad \ldots$$

A standard solution: multi-modular methods

Gröbner bases and specialization "almost" commute:



Avoiding intermediate expression swell:

1. Gröbner basis of CRN system

Compute modulo prime $\mathbf{p_1} = \mathbf{65551}$:

$k_{60} = 2196, \quad k_{50} = 36593 \pmod{p_1}$

2. Gröbner basis of CRN system

Compute modulo prime $\mathbf{p_2} = \mathbf{65557}$:

$k_{60} = 60731, \quad k_{50} = 11098 \pmod{p_2}$

In general, $\pmod I$ where I is an ideal

And then reconstructing the result:

Gröbner basis of CRN system mod $p_1 p_2$

Hensel lifting, rational function interpolation, etc.

Interpolate via Chinese Remainder Theorem :

$$k_{60} = 2941603321, \quad k_{50} = 3859679473 \mod \mathbf{p_1 p_2}, \quad \ldots$$

Gröbner tracing

Multi-modular method

Input: Polynomials $f_1, \ldots, f_n \in \mathbb{K}[x_1, \ldots, x_n]$.

Gröbner tracing

Multi-modular method

Input: Polynomials $f_1, \ldots, f_n \in \mathbb{K}[x_1, \ldots, x_n]$.

1. For $i = 1, 2, 3, \ldots$
    1.1 Let p_i be a prime number.

Gröbner tracing

Multi-modular method

Input: Polynomials $f_1, \ldots, f_n \in \mathbb{K}[x_1, \ldots, x_n]$.

1. For $i = 1, 2, 3, \ldots$
   1.1 Let p_i be a prime number.
   1.2 Compute G_i, a Gröbner basis of $\langle f_1, \ldots, f_n \rangle \bmod p_i$.

Gröbner tracing

Multi-modular method

Input: Polynomials $f_1, \ldots, f_n \in \mathbb{K}[x_1, \ldots, x_n]$.

1. For $i = 1, 2, 3, \ldots$
   1.1 Let p_i be a prime number.
   1.2 Compute G_i, a Gröbner basis of $\langle f_1, \ldots, f_n \rangle \bmod p_i$.
2. Reconstruct and return basis G from $G_1, G_2, \ldots$.

Gröbner tracing

Multi-modular method with tracing [Traverso, 1989]

Input: Polynomials $f_1, \ldots, f_n \in \mathbb{K}[x_1, \ldots, x_n]$.

1. Precompute a trace T using $f_1, \ldots, f_n$.
2. For $i = 1, 2, 3, \ldots$
   - 2.1 Let p_i be a prime number.
   - 2.2 Compute G_i, a Gröbner basis of $\langle f_1, \ldots, f_n \rangle \bmod p_i$ using the trace T.
3. Reconstruct and return basis G from $G_1, G_2, \ldots$.

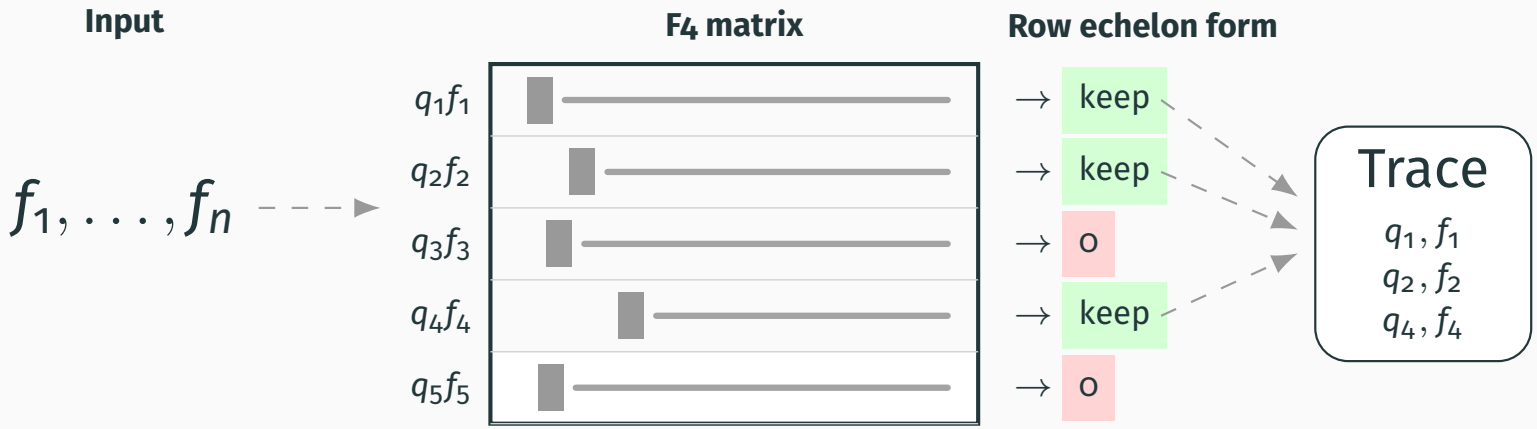
Gröbner tracing

Multi-modular method with tracing [Traverso, 1989]

Input: Polynomials $f_1, \ldots, f_n \in \mathbb{K}[x_1, \ldots, x_n]$.

1. Precompute a trace T using $f_1, \ldots, f_n$.
2. For $i = 1, 2, 3, \ldots$
   - 2.1 Let p_i be a prime number.
   - 2.2 Compute G_i, a Gröbner basis of $\langle f_1, \ldots, f_n \rangle \bmod p_i$ using the trace T.
3. Reconstruct and return basis G from $G_1, G_2, \ldots$.

When using the F4 algorithm for computing Gröbner bases:



Adoption of Gröbner tracing

Existing software uses tracing internally to compute Gröbner bases over $\mathbb{Q}$: Giac/Xcas, msolve, Groebner.jl, etc.

3. Reconstruct and return Gröbner basis G from $G_1, G_2, \ldots$.

Adoption of Gröbner tracing

Existing software uses tracing internally to compute Gröbner bases over $\mathbb{Q}$: Giac/Xcas, msolve, Groebner.jl, etc.

3. Reconstruct and return Gröbner basis G from $G_1, G_2, \ldots$.

In some applications, Gröbner basis is not the final result:

Adoption of Gröbner tracing

Existing software uses tracing internally to compute Gröbner bases over $\mathbb{Q}$: Giac/Xcas, msolve, Groebner.jl, etc.

3. Reconstruct and return Gröbner basis G from $G_1, G_2, \ldots$.

In some applications, Gröbner basis is not the final result:

- Structural identifiability – needs only a subset of the coefficients of G.

Adoption of Gröbner tracing

Existing software uses tracing internally to compute Gröbner bases over $\mathbb{Q}$: Giac/Xcas, msolve, Groebner.jl, etc.

3. Reconstruct and return Gröbner basis G from $G_1, G_2, \ldots$.

In some applications, Gröbner basis is not the final result:

- Structural identifiability – needs only a subset of the coefficients of G.
- Cryptography [Joux et al.] – does not reconstruct G at all.

Adoption of Gröbner tracing

Existing software uses tracing internally to compute Gröbner bases over $\mathbb{Q}$: Giac/Xcas, msolve, Groebner.jl, etc.

3. Reconstruct and return Gröbner basis G from $G_1, G_2, \ldots$.

In some applications, Gröbner basis is not the final result:

- Structural identifiability – needs only a subset of the coefficients of G.
- Cryptography [Joux et al.] – does not reconstruct G at all.
- Solving – $G_1, G_2, \ldots$ are enough, reconstructing G might be wasteful.

Adoption of Gröbner tracing

Existing software uses tracing internally to compute Gröbner bases over $\mathbb{Q}$: Giac/Xcas, msolve, Groebner.jl, etc.

3. Reconstruct and return Gröbner basis G from $G_1, G_2, \ldots$.

In some applications, Gröbner basis is not the final result:

- Structural identifiability – needs only a subset of the coefficients of G.
- Cryptography [Joux et al.] – does not reconstruct G at all.
- Solving – $G_1, G_2, \ldots$ are enough, reconstructing G might be wasteful.

$\rightsquigarrow$

Interface

In Gröbner bases libraries, the following interface could be useful:

- `precompute` $(f_1, \ldots, f_n) \rightarrow$ a trace T.

Adoption of Gröbner tracing

Existing software uses tracing internally to compute Gröbner bases over $\mathbb{Q}$: Giac/Xcas, msolve, Groebner.jl, etc.

3. Reconstruct and return Gröbner basis G from $G_1, G_2, \ldots$.

In some applications, Gröbner basis is not the final result:

- Structural identifiability – needs only a subset of the coefficients of G.
- Cryptography [Joux et al.] – does not reconstruct G at all.
- Solving – $G_1, G_2, \ldots$ are enough, reconstructing G might be wasteful.

$\rightsquigarrow$

Interface

In Gröbner bases libraries, the following interface could be useful:

- `precompute` $(f_1, \ldots, f_n) \rightarrow$ a trace T.
- `apply` $(f_1, \ldots, f_n \bmod p,$ trace $T) \rightarrow$ a Gröbner basis $\bmod p$.

Our contribution

- We provide such tracing interface in Groebner.jl.

```
sys, sys_1, sys_2, sys_3 = ... # define input systems appropriately

trace, _ = groebner_learn(sys)

for i = 1 : 1000
    success, gb = groebner_apply!(trace, sys_i)
end
```

Our contribution

- We provide such tracing interface in Groebner.jl.

```
sys, sys_1, sys_2, sys_3 = ... # define input systems appropriately

trace, _ = groebner_learn(sys)

for i = 1 : 1000
    success, gb = groebner_apply!(trace, sys_i)
end
```

- Our implementation is efficient

```
@btime groebner(sys);   # Katsura-10,  1024 solutions
#  731.000 ms (84551 allocations: 139.38 MiB)

@btime groebner_apply!(trace, sys);
#  263.576 ms (44002 allocations: 87.02 MiB)
```

Our contribution

- We provide such tracing interface in Groebner.jl.

```
sys, sys_1, sys_2, sys_3 = ... # define input systems appropriately

trace, _ = groebner_learn(sys)

for i = 1 : 1000
    success, gb = groebner_apply!(trace, sys_i)
end
```

- Our implementation is efficient

```
@btime groebner(sys);   # Katsura-10,  1024 solutions
#  731.000 ms (84551 allocations: 139.38 MiB)

@btime groebner_apply!(trace, sys);
#  263.576 ms (44002 allocations: 87.02 MiB)
```

- We make tracing faster.

Making Gröbner tracing faster

Recall components of the classic F4 algorithm mod p:

Making Gröbner tracing faster

Recall components of the classic F4 algorithm mod p:

- Constructing S-polynomials
- Symbolic preprocessing
- Constructing Macaulay matrix
- Updating critical pairs

Making Gröbner tracing faster

Recall components of the classic F4 algorithm mod p:

- Constructing S-polynomials
- Symbolic preprocessing
- Constructing Macaulay matrix
- Updating critical pairs
- Computing row echelon form mod p

Making Gröbner tracing faster

Recall components of the classic F4 algorithm mod p:

- Constructing S-polynomials
- Symbolic preprocessing
- Constructing Macaulay matrix
- Updating critical pairs

$\rightsquigarrow$ depend only on **monomials**

- Computing row echelon form mod p

$\rightsquigarrow$ depends on **coefficients** and p

Making Gröbner tracing faster

Recall components of the classic F4 algorithm mod $(p_1, \ldots, p_N)$:

- Constructing S-polynomials
- Symbolic preprocessing
- Constructing Macaulay matrix
- Updating critical pairs

$\rightsquigarrow$ depend only on **monomials**

- Computing row echelon form mod $(p_1, \ldots, p_N)$

$\rightsquigarrow$ depends on **coefficients** and p

$\rightsquigarrow$ compute mod $(p_1, \ldots, p_N)$, with arithmetic done component-wise.

Making Gröbner tracing faster

Recall components of the classic F4 algorithm mod $(p_1, \ldots, p_N)$:

- Constructing S-polynomials
- Symbolic preprocessing
- Constructing Macaulay matrix
- Updating critical pairs

$\rightsquigarrow$ depend only on **monomials**

- Computing row echelon form mod $(p_1, \ldots, p_N)$

$\rightsquigarrow$ depends on **coefficients** and p

$\rightsquigarrow$ compute mod $(p_1, \ldots, p_N)$, with arithmetic done component-wise.

In other words..

Compute over the product ring

$$\mathbb{Z}/p_1\mathbb{Z} \times \cdots \times \mathbb{Z}/p_N\mathbb{Z}$$

Making Gröbner tracing faster

Recall components of the classic F4 algorithm mod $(p_1, \ldots, p_N)$:

- Constructing S-polynomials
- Symbolic preprocessing
- Constructing Macaulay matrix
- Updating critical pairs

$\rightsquigarrow$ depend only on **monomials**

- Computing row echelon form mod $(p_1, \ldots, p_N)$

$\rightsquigarrow$ depends on **coefficients** and p

$\rightsquigarrow$ compute mod $(p_1, \ldots, p_N)$, with arithmetic done component-wise.

In other words..

Compute over the product ring

$$\mathbb{Z}/p_1\mathbb{Z} \times \cdots \times \mathbb{Z}/p_N\mathbb{Z}$$

Used by Gräbe, and de Kleine & Monagan already in **1993** and **2001**, resp.

Making Gröbner tracing faster

Arithmetic in $\mathbb{Z}/p_1\mathbb{Z} \times \cdots \times \mathbb{Z}/p_N\mathbb{Z}$ can be implemented with tuples:

```
julia> a1 = (1,2,3,4)::NTuple{4, Int64}
(1, 2, 3, 4)

julia> a2 = (5,6,7,8)::NTuple{4, Int64}
(5, 6, 7, 8)
```

```
julia> a1 .+ a2
(6, 8, 10, 12)
```

Typically uses SIMD automatically in Julia:

```
julia> @code_native a1 .+ a2
        push    rbp
        mov     rbp, rsp
        mov     rax, rdi
        vmovdqu ymm0, ymmword ptr [rdx]
        vpaddq  ymm0, ymm0, ymmword ptr [rsi]
        vmovdqu ymmword ptr [rdi], ymm0
        pop     rbp
        ret
```

Making Gröbner tracing faster

In the interface:

```
success, (gb1,gb2,gb3,gb4) = groebner_apply!(trace, (sys1,sys2,sys3,sys4))
```

Making Gröbner tracing faster

In the interface:

```
success, (gb1,gb2,gb3,gb4) = groebner_apply!(trace, (sys1,sys2,sys3,sys4))
```

Before:

2. For $i = 1, 2, 3, \ldots$
   2.1 Let p_i be a prime number.
   2.2 Compute G_i, a Gröbner basis of $\langle f_1, \ldots, f_n \rangle \bmod p_i$ using the trace T.

Making Gröbner tracing faster

In the interface:

```
success, (gb1,gb2,gb3,gb4) = groebner_apply!(trace, (sys1,sys2,sys3,sys4))
```

Before:

2. For $i = 1, 2, 3, \ldots$
   2.1 Let p_i be a prime number.
   2.2 Compute G_i, a Gröbner basis of $\langle f_1, \ldots, f_n \rangle \bmod p_i$ using the trace T.

After:

1. Let $p_1, \ldots, p_N$ be prime numbers.
2. Compute a Gröbner basis of $\langle f_1, \ldots, f_n \rangle \bmod (p_1, \ldots, p_N)$ using trace T.

Results

In `StructuralIdentifiability.jl`:

Table 1: The number of Gröbner bases and the total running times for computing generators of the field of identifiable functions via rational function interpolation

Model	# Gröbner bases	F4	F4 with tracing	Speedup
EAIHRD	420	300 s	49 s	6.1
LinComp2	4388	275 s	164 s	1.7
Pharm	34	54 s	36 s	1.5

Results

In `StructuralIdentifiability.jl`:

Table 1: The number of Gröbner bases and the total running times for computing generators of the field of identifiable functions via rational function interpolation

Model	# Gröbner bases	F4	F4 with tracing	Speedup
EAIHRD	420	300 s	49 s	6.1
LinComp2	4388	275 s	164 s	1.7
Pharm	34	54 s	36 s	1.5

In `RationalUnivariateRepresentation.jl`:

Table 2: Total running times of rational univariate representation computation.

System	F4	F4 with tracing	Speedup
Chandra-10	82.4 s	29.3 s	2.8
Chandra-11	851.4 s	373.5 s	2.3
Eco-10	1.0 s	0.4 s	2.5
Eco-11	11.2 s	4.2 s	2.7
Katsura-11	71.0 s	28.2 s	2.5
SEIR36	63.7 s	9.6 s	6.6

Thank you!

Table 3: Running times, and the number of computations required to amortize the cost of learning a trace compared against classic independent Gröbner basis computations.

System	ClassicF4	Precompute	Apply	Computations to break even
Chandra-11	1.0 s	3.3 s	0.1 s	4
Chandra-12	5.8 s	19.6 s	0.3 s	4
Katsura-12	2.2 s	17.2 s	0.8 s	13
Katsura-13	14.2 s	153.2 s	6.1 s	19
Cyclic-8	0.4 s	1.5 s	0.2 s	8
Cyclic-9	38.9 s	219.4 s	26.7 s	19
Cholera	13.7 s	40.9 s	7.7 s	7
Yang1	5.8 s	39.2 s	0.4 s	8