Current Advances in

Open Source Gröbner Basis Algorithms

My name is **Christian Eder**.

I am from TECHNISCHE UNIVERSITÄT KAISERSLAUTERN.

Several new techniques and implementations

An integration in the computer algebra system **OSCAR**

TRR 195

SYMBOLIC TOOLS

#1
Computing Gröbner bases

$$I = \langle f_1, \ldots, f_r \rangle \subset K[x]$$

$$G \leftarrow \{f_1, \ldots, f_r\}$$

$$P \leftarrow \{(f_i, f_j) \mid 1 \leq i < j \leq r\}$$

While $(P \neq \emptyset)$ do

Choose (p, q)

$P \leftarrow P \setminus \{(p, q)\}$

$$h \leftarrow \mathsf{spoly}(p, q) = \lambda p - \sigma q$$

$$\text{s.t. } \mathsf{lt}(\lambda p) = \mathsf{lt}(\sigma q).$$

$$h \leftarrow \mathrm{reduce}(h, G)$$

$h \neq 0$? (Buchberger's criterion)

$P \leftarrow P \cup \{(h, g) \mid g \in G\}$

$G \leftarrow G \cup \{h\}$

Process next element from P

$$\frac{h = 0?}{\text{Process next element from } P}$$

$$\frac{P = \emptyset ?}{\text{Return } G}$$

All in all

$I = \langle f_1, \ldots, f_r \rangle \subset K[x]$

$G \leftarrow \{f_1, \ldots, f_r\}$

$P \leftarrow \{(f_i, f_j) \mid 1 \leq i < j \leq r\}$

While $(P \neq \emptyset)$ do

   Choose (p, q), $P \leftarrow P \setminus \{(p, q)\}$

   $h \leftarrow \text{spoly}(p, q) = \lambda p - \sigma q$

   $h \leftarrow \text{reduce}(h, G)$

   $h \neq 0$?

      $P \leftarrow P \cup \{(h, g) \mid g \in G\}$

      $G \leftarrow G \cup \{h\}$

Return G

Let $I = \langle f_1, f_2 \rangle \in \mathcal{R} := \mathbb{Q}[x, y, z]$ and let $<$ denote DRL where

$$g_1 := f_1 = xy - z^2,$$
$$g_2 := f_2 = y^2 - z^2.$$
$$G \leftarrow \{g_1, g_2\}$$
$$P \leftarrow \{(g_1, g_2)\}.$$

$\text{spoly}(g_1, g_2) = y\left(xy - z^2\right) - x\left(y^2 - z^2\right)$

No further reduction w.r.t. G.

$g_3 \leftarrow xz^2 - yz^2$

$P \leftarrow P \cup \{(g_1, g_3), (g_2, g_3)\}$

$G \leftarrow G \cup \{g_3\}$

How to avoid zero reductions?

Lead terms of p and q coprime? Then spoly$(p, q) \to 0$

Example:
spoly$(g_2, g_3) = xz^2 \left(y^2 - z^2\right) - y^2 \left(xz^2 - yz^2\right)$

Further reduce with $yz^2 \left(y^2 - z^2\right)$ and $z^2 \left(xz^2 - yz^2\right)$.

Chain of S-polynomials?

$\mathsf{spoly}(p, q) = \lambda\, \mathsf{spoly}(p, r) + \sigma\, \mathsf{spoly}(r, q)$

Two of those three are enough.

What's about spoly(g_1, g_3)?

Idea of signatures:
Faugère's F5 algorithm, GVW algorithm, . . .

Apply signatures in $\mathcal{R}^2$:

$\text{sig}(g_1) = e_1,$
$\text{sig}(g_2) = e_2.$

Order signatures by POT (e.g. $e_2 > x^{1000}e_1$).

In general:
$\text{sig}(\text{polynomial}) = \text{lt}(\text{module representation})$

Main idea: Try to keep signature minimal.

$$\mathrm{sig}(\mathrm{spoly}(g_1, g_2) = yg_1 - xg_2) = \mathrm{lt}(ye_1 - xe_2) = -xe_2.$$

$g_3 \leftarrow \mathrm{reduce}(\mathrm{spoly}(g_1, g_2), G)$
Have to ensure: $\mathrm{sig}(g_3) = \mathrm{sig}(\mathrm{spoly}(g_1, g_2))$.

Note: **Restriction of the reduction process**.

spoly(g_1, g_3) reduces to zero: (Syzygy/F5 criterion)

$$\begin{aligned} \text{sig}(\text{spoly}(g_1, g_3)) &= \text{lt}\left(z^2 e_1 - y(ye_1 - xe_2)\right) \\ &= xye_2. \end{aligned}$$

Use **syzygy** $g_1 e_2 - g_2 e_1$ with lead term xye_2:

▷ Reduce module representation.

▷ Lower signature for spoly(g_1, g_3).

#2
Computing with signatures over the integers

joint work with

Gerhard Pfister
Adrian Popescu

Over the integers stuff gets more difficult.

$$\text{spoly}(g_i, g_j) = \lambda g_i - \sigma g_j$$
$$\text{sig}(\lambda g_i) = c_i \tau e_k$$
$$\text{sig}(\sigma g_j) = c_j \tau e_k$$
$$\text{sig}(\text{spoly}(g_i, g_j)) = (c_i - c_j)\tau e_k$$

Concept of **signature drops**

Idea

- ▷ Stop computation at this point.
- ▷ Interreduce intermediate basis without considering signatures.
- ▷ Apply new signatures / module representations and restart.

Restarting is a **bottleneck** in general.

But the **intermediate basis** is quite good.

Main optimization: Hybrid algorithm

- ▷ Start with signature-based algorithm.
- ▷ If signature drops, restart for a (**small**) number of times the signature-based algorithm.
- ▷ Take intermediate basis and start a Gröbner basis computation which is **not signature-based**.

Examples	STD	HBA	STD/HBA
1	10.43	0.37	28.19
2	24.91	0.10	249.10
3	87.27	0.39	223.77
4	83.51	0.20	417.55
5	23,200.05	5,873.21	3.95
6	134.29	0.61	220.15
7	1,004.56	1,128.07	0.89
8	554.02	337.55	1.641

#3
The noncommutative world

joint work with

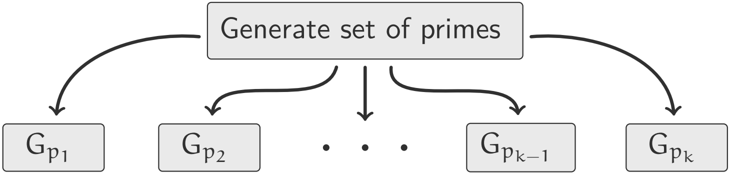
Wolfram Decker
Viktor Levandovskyy
Sharwan K. Tiwari

Modular GB computation over $\mathbb{Q}$

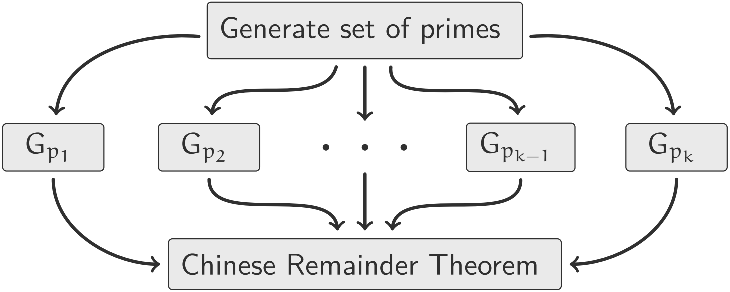
Extend techniques to G-algebras.

This allows also **parallel** computation.

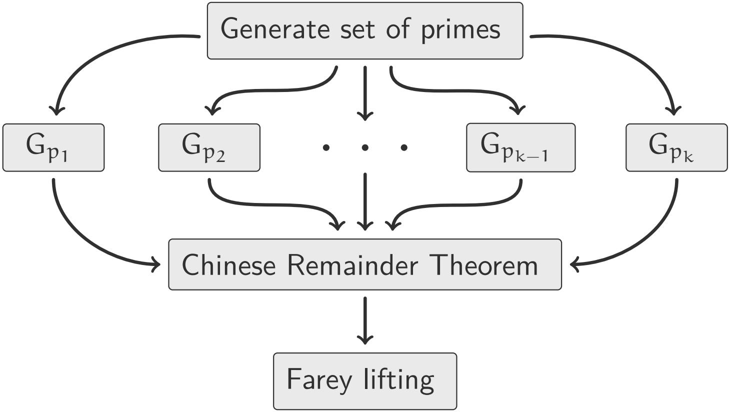
Generate set of primes

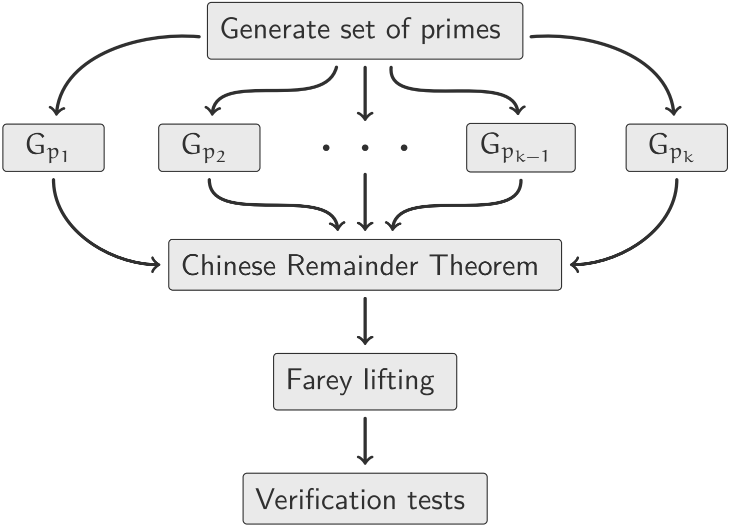


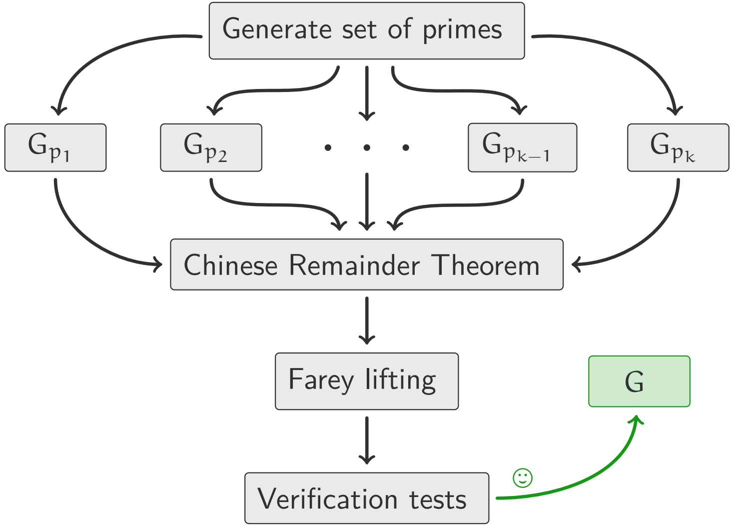
Generate set of primes
G_{p_1}
G_{p_2}
...
$G_{p_{k-1}}$
G_{p_k}



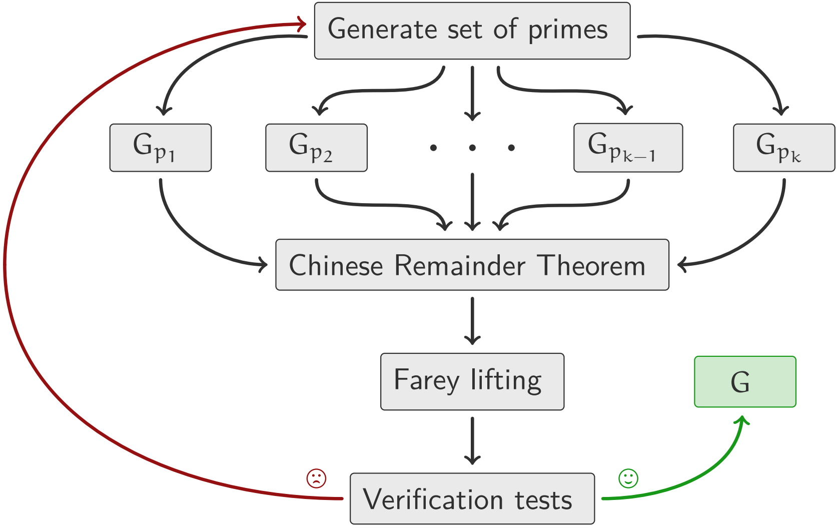
Generate set of primes
G_{p_1}
G_{p_2}
$G_{p_{k-1}}$
G_{p_k}
Chinese Remainder Theorem


Generate set of primes
G_{p_1}
G_{p_2}
$G_{p_{k-1}}$
G_{p_k}
Chinese Remainder Theorem
Farey lifting


Generate set of primes
G_{p_1}
G_{p_2}
. . .
$G_{p_{k-1}}$
G_{p_k}
Chinese Remainder Theorem
Farey lifting
Verification tests



Generate set of primes
G_{p_1}
G_{p_2}
. . .
$G_{p_{k-1}}$
G_{p_k}
Chinese Remainder Theorem
Farey lifting
Verification tests
☺
G



Generate set of primes
G_{p_1}
G_{p_2}
. . .
$G_{p_{k-1}}$
G_{p_k}
Chinese Remainder Theorem
Farey lifting
G
☹
Verification tests
☺

Examples	sgb	m-sgb-1	m-sgb-4	m-sgb-16
cyclic(8)	55.28 h	2.51 h	34.65 m	17.13 m
katsura(11)	199.71 h	4.32 h	1.59 h	24.52 m
katsura(12)	-	13.78 h	4.40 h	1.46 h
katsura(13)	-	50.14 h	17.74 h	5.80 h
reimer(5)	29.07 h	2.59 h	58.47 m	18.04 m
eco(15)	25.93 h	9.40 h	3.54 h	1.83 h
Reiffen(5,6)	63.86 h	12.25 m	4.70 m	2.60 m
Reiffen(6,7)	-	10.43 h	4.65 h	3.54 h
Reiffen(7,8)	-	336.25 h	170.32 h	118.17 h

Further generalizations to **letterplace** algebras coming soon.

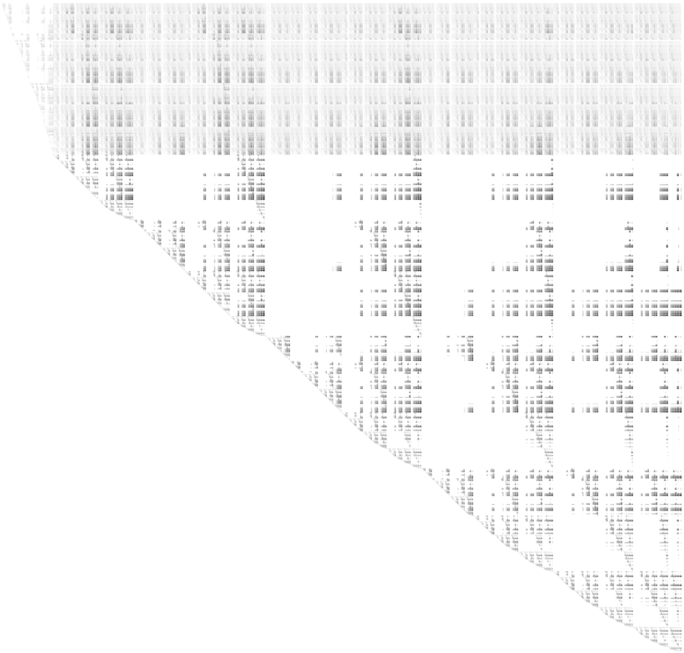
#4
Using linear algebra

Or:

How does Faugère's F4 algorithm works?

joint work with

Jean-Charles Faugère



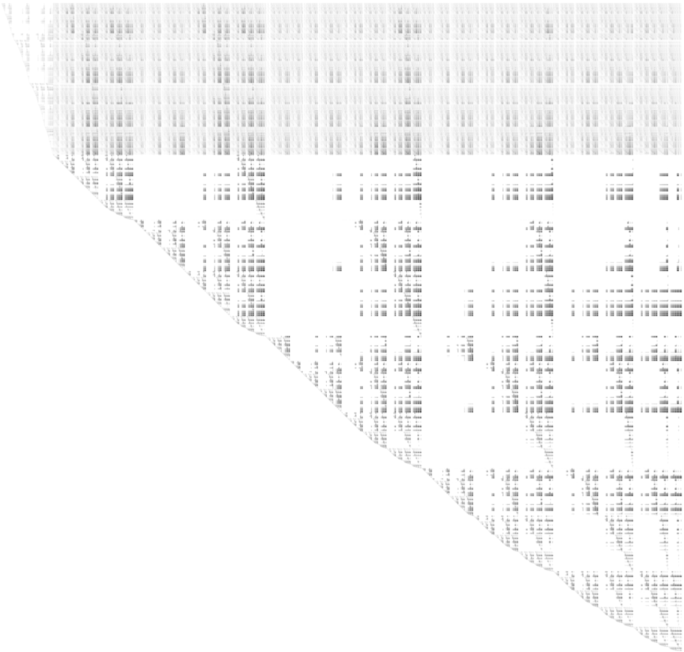
Problem

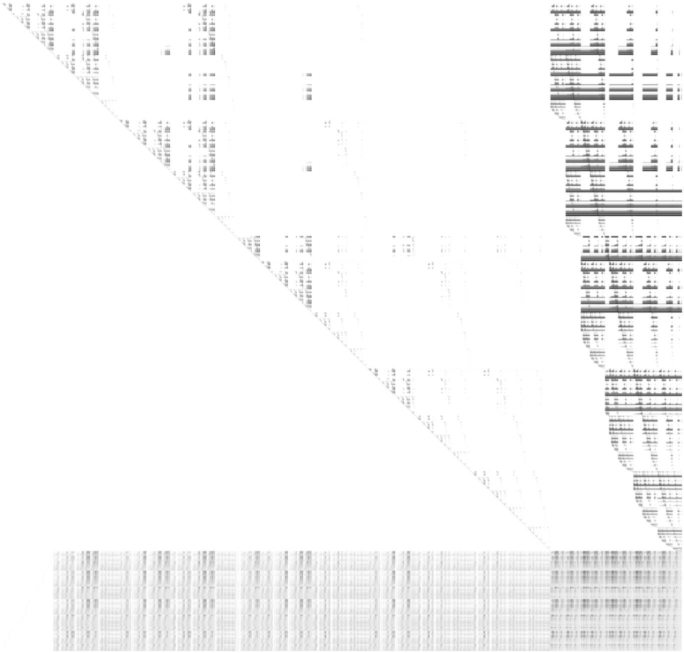
When applying Gaussian Elimination
we cannot swap columns:

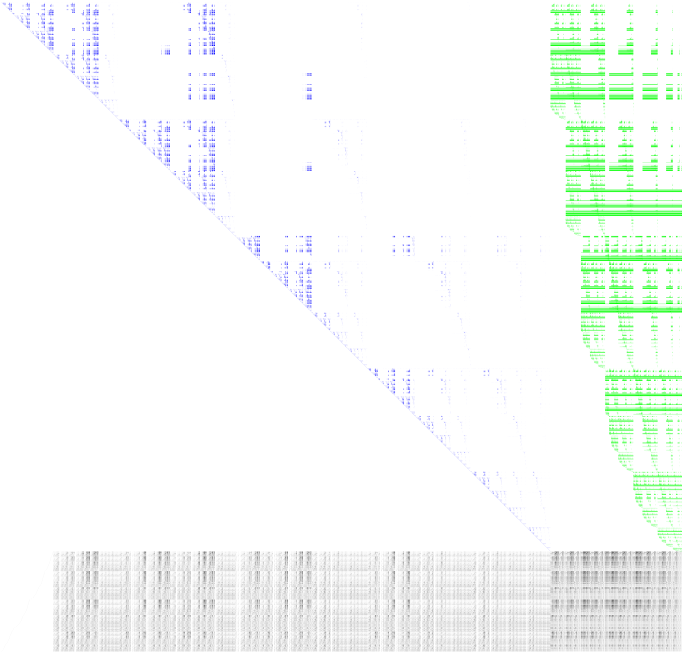
column order = **monomial order**

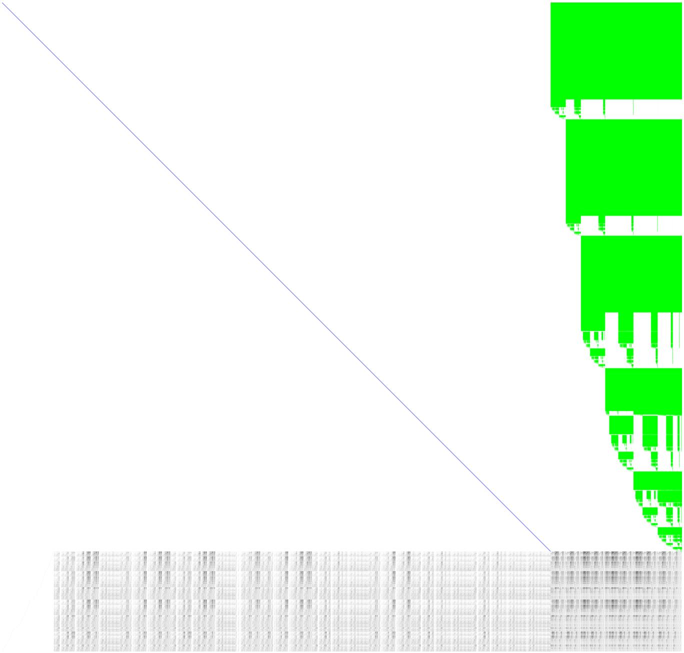
Idea

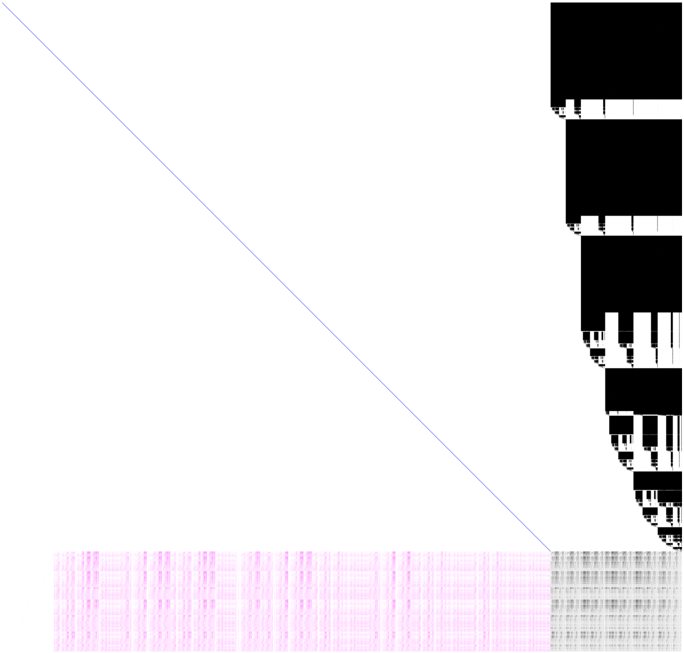
- ▷ Order columns in a “nice” way.
- ▷ Apply specialized Gaussian Elimination.
- ▷ Reorder columns.

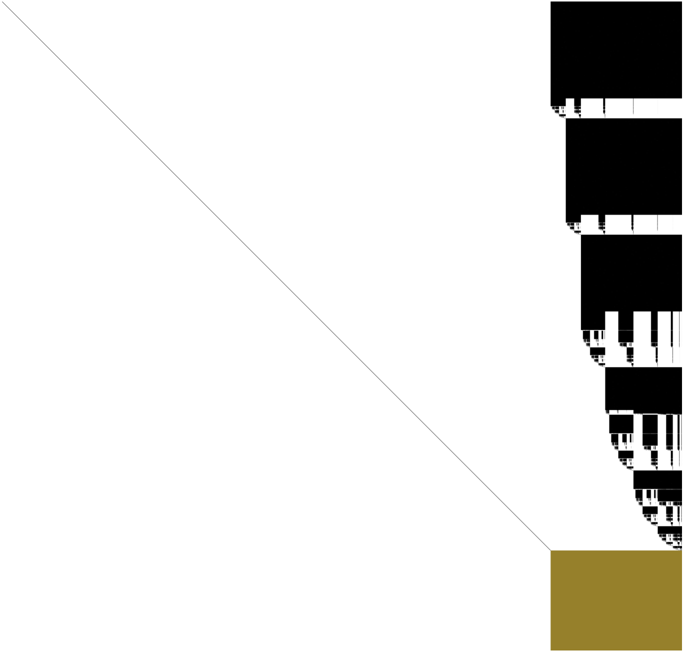


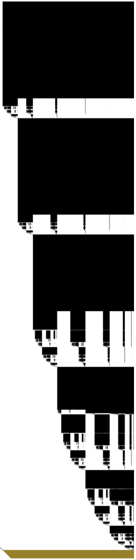




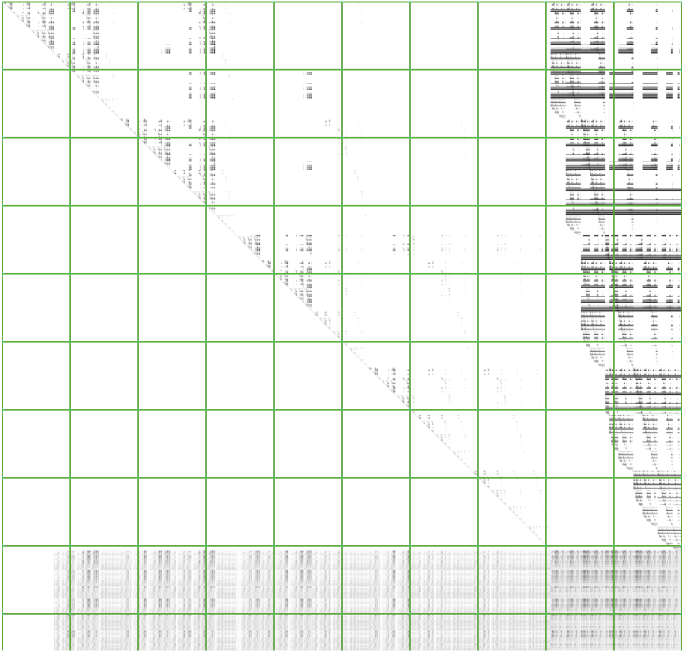


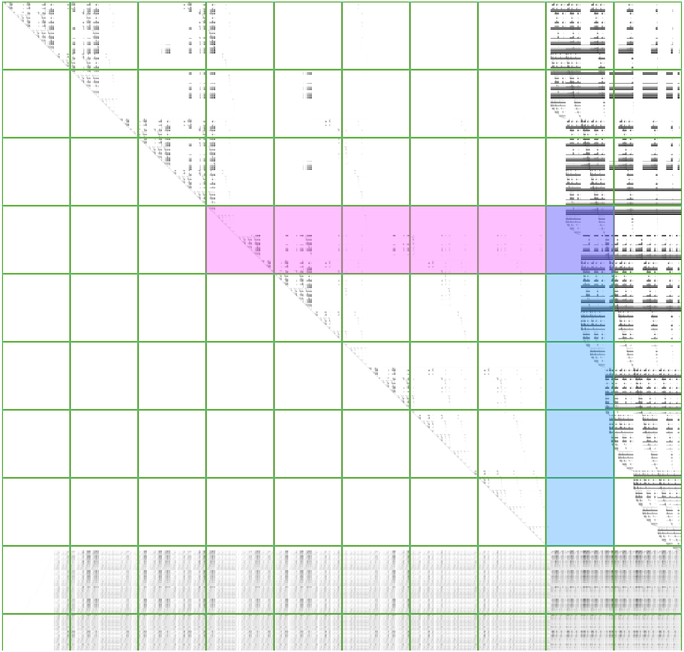


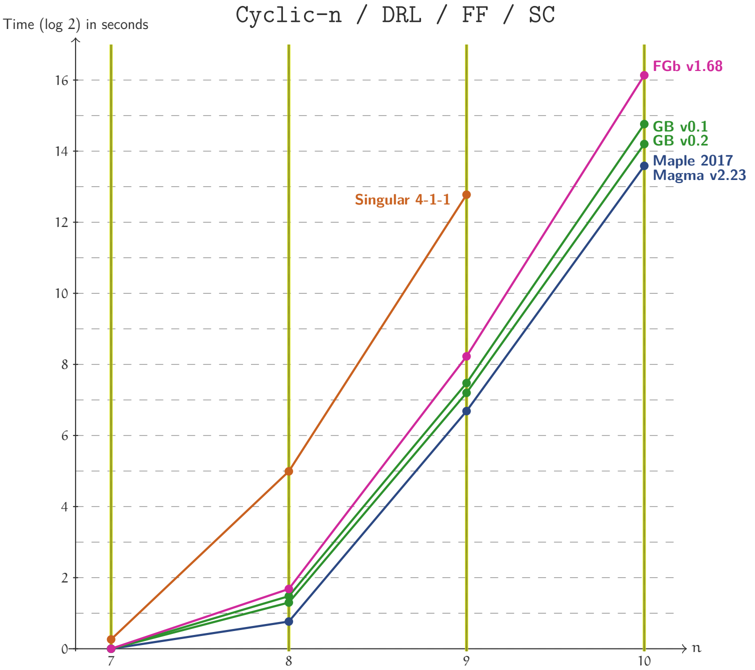




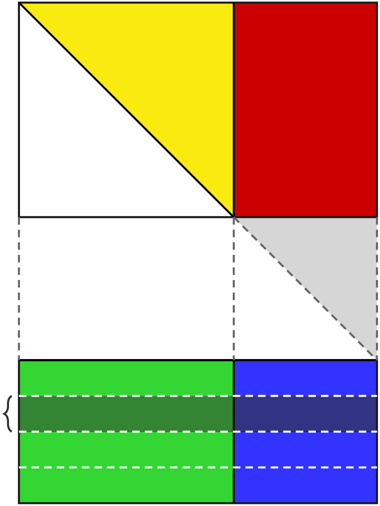
Parallel computation?

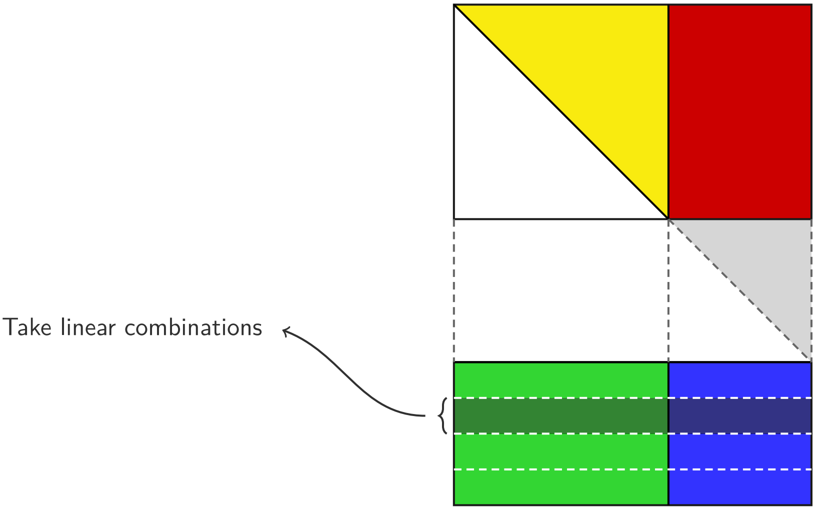


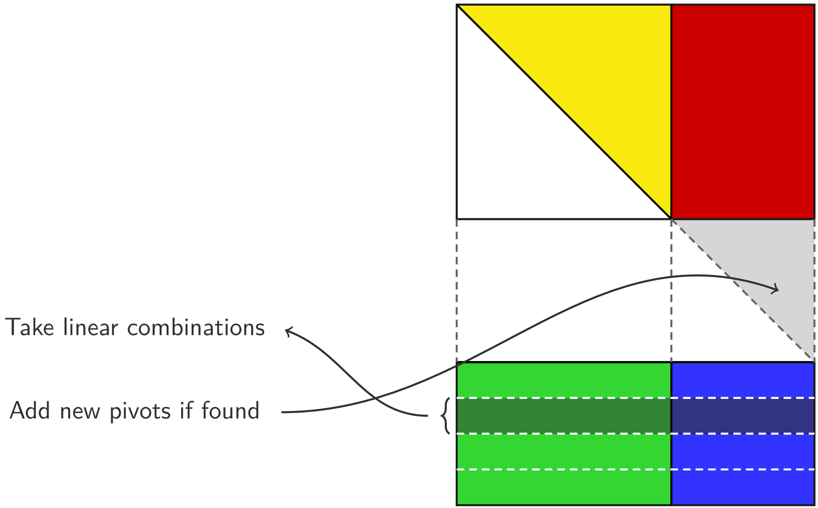


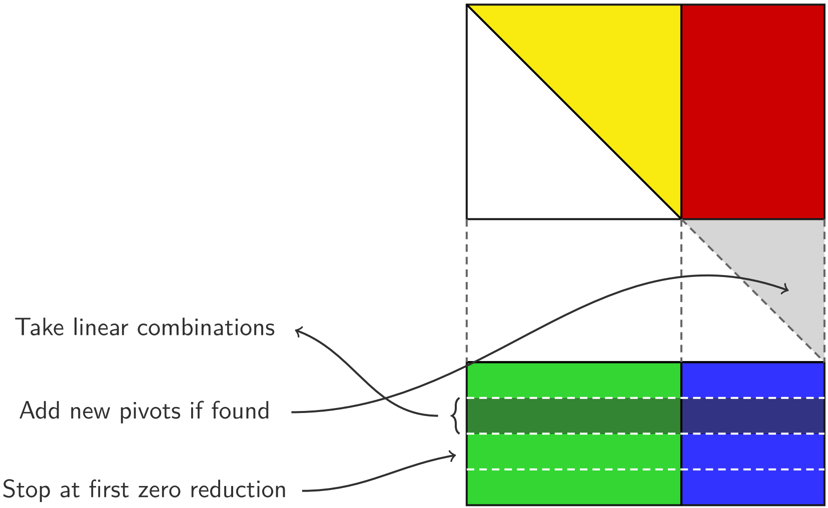

Cyclic-n / DRL / FF / SC
Time (log 2) in seconds
FGb v1.68
GB v0.1
GB v0.2
Maple 2017
Magma v2.23
Singular 4-1-1
n
0
2
4
6
8
10
12
14
16
7
8
9
10

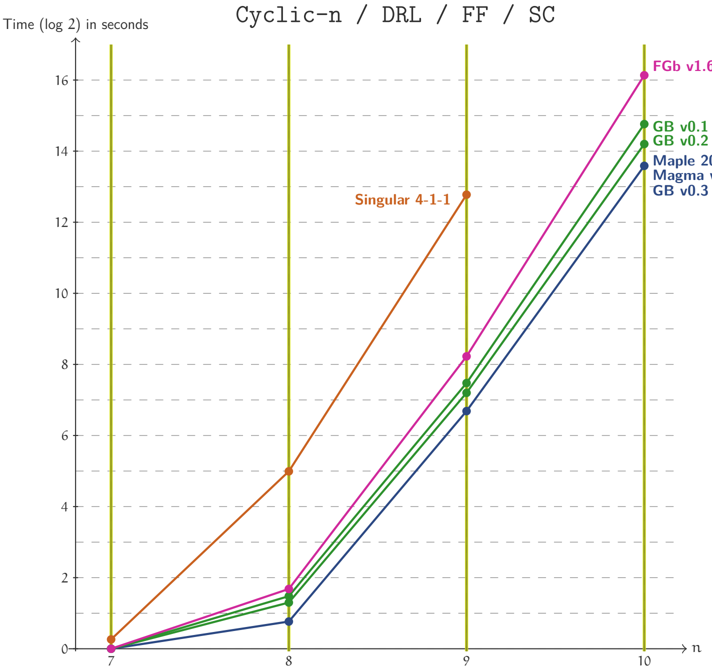
#5
Probabilistic linear algebra



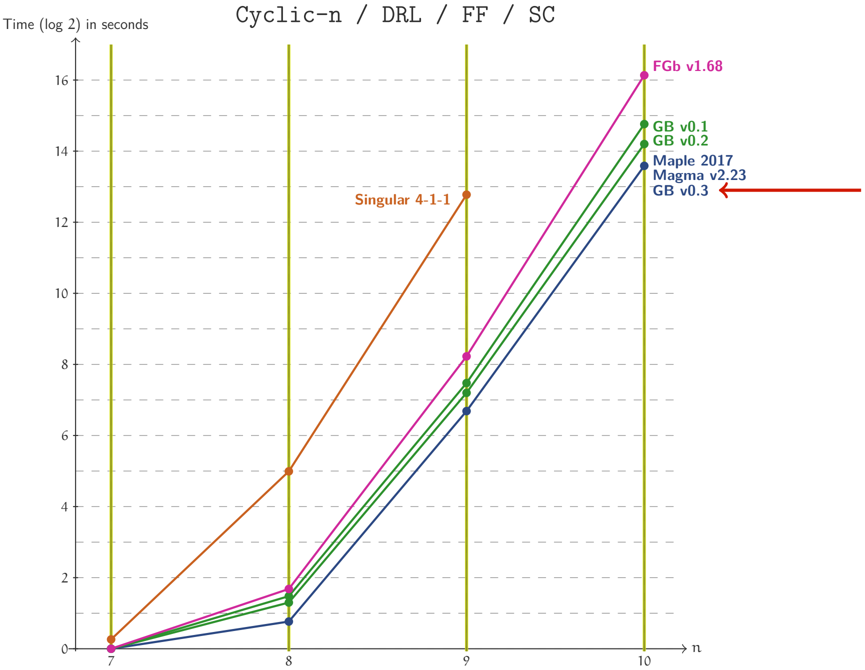

Take linear combinations


Take linear combinations
Add new pivots if found


Take linear combinations
Add new pivots if found
Stop at first zero reduction



Cyclic-n / DRL / FF / SC
Time (log 2) in seconds
0
2
4
6
8
10
12
14
16
7
8
9
10
n
FGb v1.68
GB v0.1
GB v0.2
Maple 2017
Magma v2.23
GB v0.3
Singular 4-1-1


Cyclic-n / DRL / FF / SC
Time (log 2) in seconds
16
14
12
10
8
6
4
2
0
7
8
9
10
n
FGb v1.68
GB v0.1
GB v0.2
Maple 2017
Magma v2.23
GB v0.3
Singular 4-1-1

New features in GB v0.3:

Probabilistic linear algebra

Prime fields up to 32 bits

julia interface to **SINGULAR** (**OSCAR**)

Start your **julia** session. Then

```
//Load the GB.jl library, also loads Singular.jl.
using GB

// Next we define a ring R of characteristic 2^31-1
// with DRL order and the ideal I in R generated by the
// cyclic generators with 10 variables.
R,I := GB.cyclic_10(2^31-1)

// Compute Groebner basis G for I using standard
// settings of GB's F4 implementation.
G := Gb.f4(I)

// Same computation, but with specialized setting:
// hash table size = 2^21, 8 threads,
// max.  2500 s-polynomials, probabilistic linear algebra
G := Gb.f4(I,21,8,2500,42)

// Further process G in Singular
Singular.ngens(G)
```

Next steps for GB:

Better hashing, k-d-trees
ARM chips
GPU for linear algebra (OpenCL)
On-chip GPU usage for hashing (OpenCL)
Distributed computation
Multi-modular computation in **Julia**
Signature-based linear algebra

More infos?

www.mathematik.uni-kl.de/~ederc

Thank you for your attention.

Questions? Remarks?