

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**

TR 195
SYMBOLIC TOOLS

#1

Computing Gröbner bases

$$I = \langle f_1, \dots, f_r \rangle \subset \mathbf{K}[x]$$

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

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

While ($P \neq \emptyset$) do

Choose (p, q)

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

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

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

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

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

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

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

Process next element from P

$h = 0?$

Process next element from P

$P = \emptyset?$

Return G

All in all

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

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

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

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

$$g_1 := f_1 = xy - z^2,$$

$$g_2 := f_2 = y^2 - z^2.$$

$$G \leftarrow \{g_1, g_2\}$$

$$P \leftarrow \{(g_1, g_2)\}.$$

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

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 $\text{spoly}(p, q) \rightarrow 0$

Example:

$$\text{spoly}(g_2, g_3) = xz^2 (y^2 - z^2) - y^2 (xz^2 - yz^2)$$

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

Chain of S-polynomials?

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

Two of those three are enough.

What's about $\text{spoly}(g_1, g_3)$?

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.

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

$$g_3 \leftarrow \text{reduce}(\text{spoly}(g_1, g_2), G)$$

Have to ensure: $\text{sig}(g_3) = \text{sig}(\text{spoly}(g_1, g_2))$.

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

$\text{spoly}(g_1, g_3)$ reduces to zero: (**Syzygy/F5 criterion**)

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

Use **syzygy** $g_1e_2 - g_2e_1$ with lead term xye_2 :

- ▷ Reduce module representation.
- ▷ Lower signature for $\text{spoly}(g_1, g_3)$.

#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

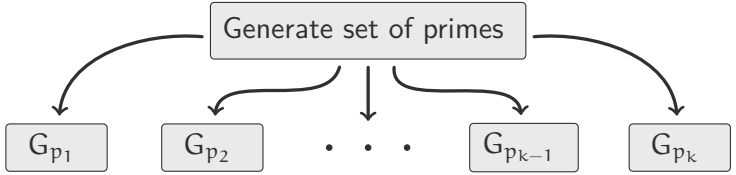
Sharwan K. Tiwari

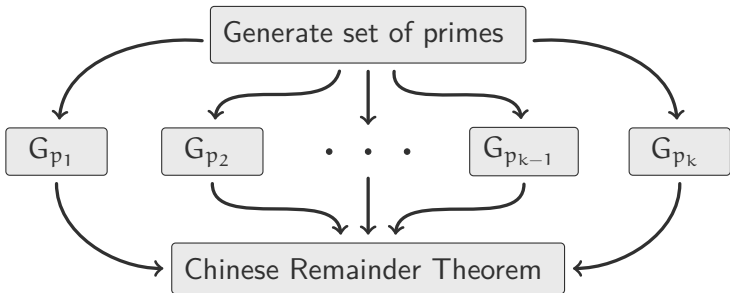
Modular GB computation over \mathbb{Q}

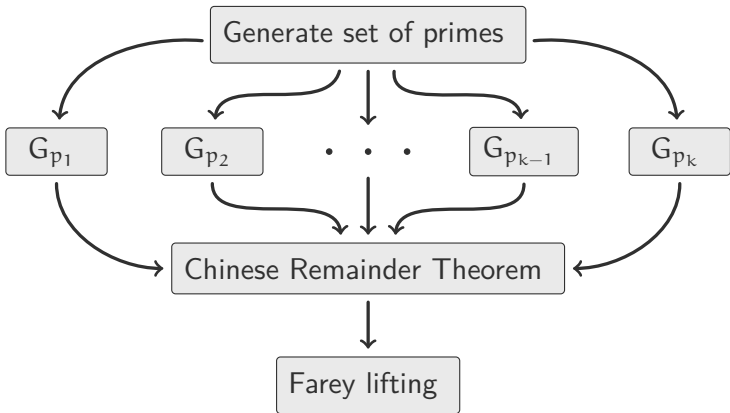
Extend techniques to G-algebras.

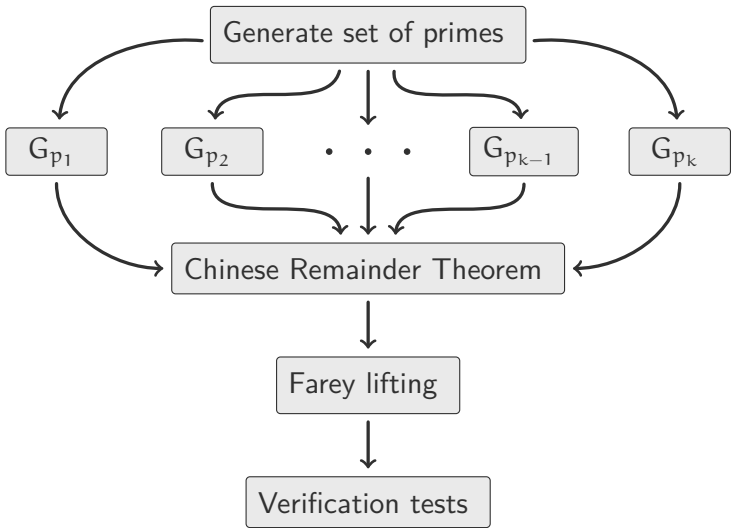
This allows also **parallel** computation.

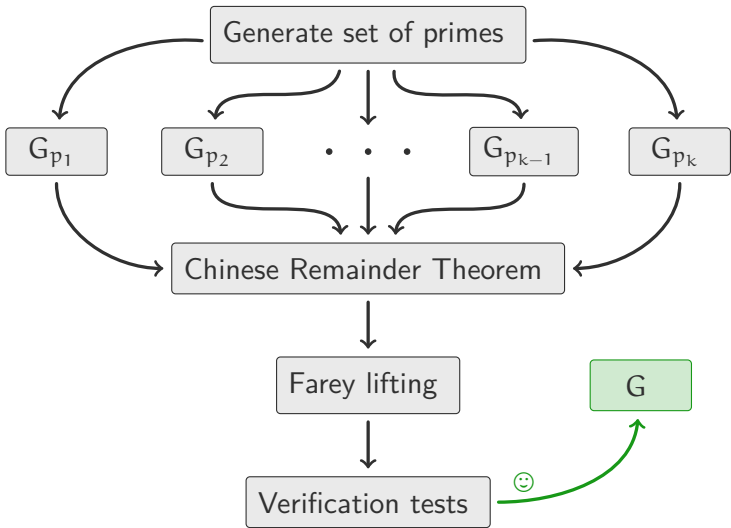
Generate set of primes

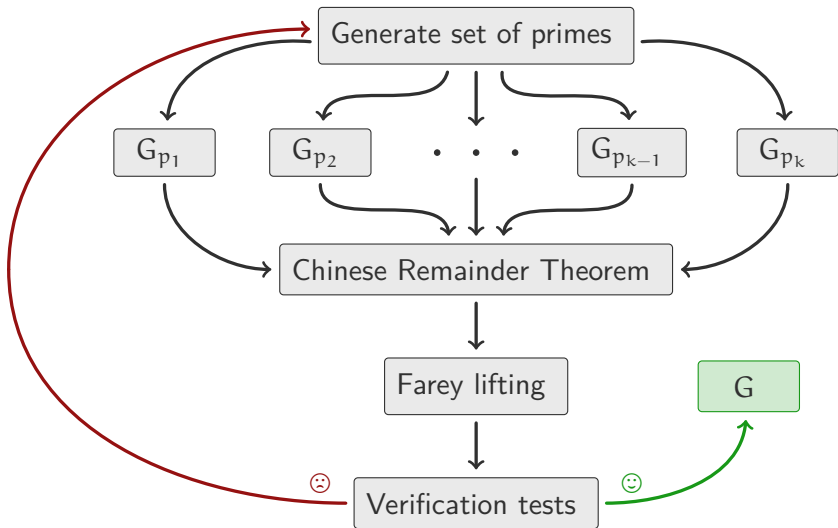












Examples	sgb	m-sgb-1	m-sgb-4	m-sgb-16
cyclic(8)	55.28 h	2.51 h	34.65 m	17.13 m
katsura(11)	199.71 h	4.32 h	1.59 h	24.52 m
katsura(12)	-	13.78 h	4.40 h	1.46 h
katsura(13)	-	50.14 h	17.74 h	5.80 h
reimer(5)	29.07 h	2.59 h	58.47 m	18.04 m
eco(15)	25.93 h	9.40 h	3.54 h	1.83 h
Reiffen(5,6)	63.86 h	12.25 m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

Year	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030																																																																																																																								
Population	1000000	1050000	1100000	1150000	1200000	1250000	1300000	1350000	1400000	1450000	1500000	1550000	1600000	1650000	1700000	1750000	1800000	1850000	1900000	1950000	2000000	2050000	2100000	2150000	2200000	2250000	2300000	2350000	2400000	2450000	2500000	2550000	2600000	2650000	2700000	2750000	2800000	2850000	2900000	2950000	3000000	3050000	3100000	3150000	3200000	3250000	3300000	3350000	3400000	3450000	3500000	3550000	3600000	3650000	3700000	3750000	3800000	3850000	3900000	3950000	4000000	4050000	4100000	4150000	4200000	4250000	4300000	4350000	4400000	4450000	4500000	4550000	4600000	4650000	4700000	4750000	4800000	4850000	4900000	4950000	5000000	5050000	5100000	5150000	5200000	5250000	5300000	5350000	5400000	5450000	5500000	5550000	5600000	5650000	5700000	5750000	5800000	5850000	5900000	5950000	6000000	6050000	6100000	6150000	6200000	6250000	6300000	6350000	6400000	6450000	6500000	6550000	6600000	6650000	6700000	6750000	6800000	6850000	6900000	6950000	7000000	7050000	7100000	7150000	7200000	7250000	7300000	7350000	7400000	7450000	7500000	7550000	7600000	7650000	7700000	7750000	7800000	7850000	7900000	7950000	8000000	8050000	8100000	8150000	8200000	8250000	8300000	8350000	8400000	8450000	8500000	8550000	8600000	8650000	8700000	8750000	8800000	8850000	8900000	8950000	9000000	9050000	9100000	9150000	9200000	9250000	9300000	9350000	9400000	9450000	9500000	9550000	9600000	9650000	9700000	9750000	9800000	9850000	9900000	9950000	10000000
GDP	10000000000	10500000000	11000000000	11500000000	12000000000	12500000000	13000000000	13500000000	14000000000	14500000000	15000000000	15500000000	16000000000	16500000000	17000000000	17500000000	18000000000	18500000000	19000000000	19500000000	20000000000	20500000000	21000000000	21500000000	22000000000	22500000000	23000000000	23500000000	24000000000	24500000000	25000000000	25500000000	26000000000	26500000000	27000000000	27500000000	28000000000	28500000000	29000000000	29500000000	30000000000	30500000000	31000000000	31500000000	32000000000	32500000000	33000000000	33500000000	34000000000	34500000000	35000000000	35500000000	36000000000	36500000000	37000000000	37500000000	38000000000	38500000000	39000000000	39500000000	40000000000	40500000000	41000000000	41500000000	42000000000	42500000000	43000000000	43500000000	44000000000	44500000000	45000000000	45500000000	46000000000	46500000000	47000000000	47500000000	48000000000	48500000000	49000000000	49500000000	50000000000	50500000000	51000000000	51500000000	52000000000	52500000000	53000000000	53500000000	54000000000	54500000000	55000000000	55500000000	56000000000	56500000000	57000000000	57500000000	58000000000	58500000000	59000000000	59500000000	60000000000	60500000000	61000000000	61500000000	62000000000	62500000000	63000000000	63500000000	64000000000	64500000000	65000000000	65500000000	66000000000	66500000000	67000000000	67500000000	68000000000	68500000000	69000000000	69500000000	70000000000	70500000000	71000000000	71500000000	72000000000	72500000000	73000000000	73500000000	74000000000	74500000000	75000000000	75500000000	76000000000	76500000000	77000000000	77500000000	78000000000	78500000000	79000000000	79500000000	80000000000	80500000000	81000000000	81500000000	82000000000	82500000000	83000000000	83500000000	84000000000	84500000000	85000000000	85500000000	86000000000	86500000000	87000000000	87500000000	88000000000	88500000000	89000000000	89500000000	90000000000	90500000000	91000000000	91500000000	92000000000	92500000000	93000000000	93500000000	94000000000	94500000000	95000000000	95500000000	96000000000	96500000000	97000000000	97500000000	98000000000	98500000000	99000000000	99500000000	100000000000

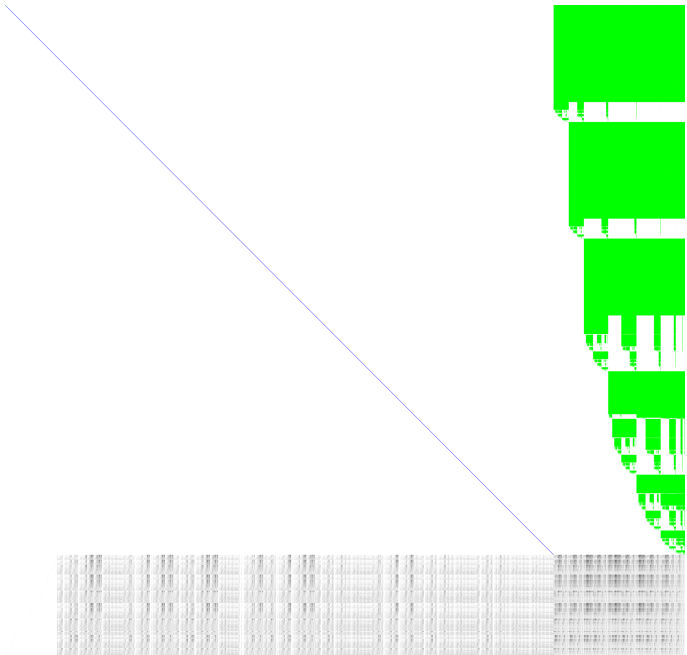
Problem

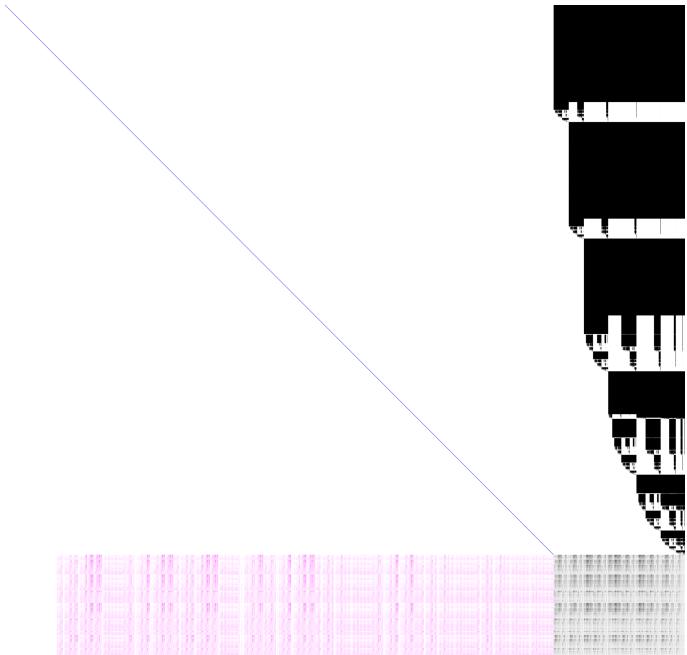
When applying Gaussian Elimination we cannot swap columns:

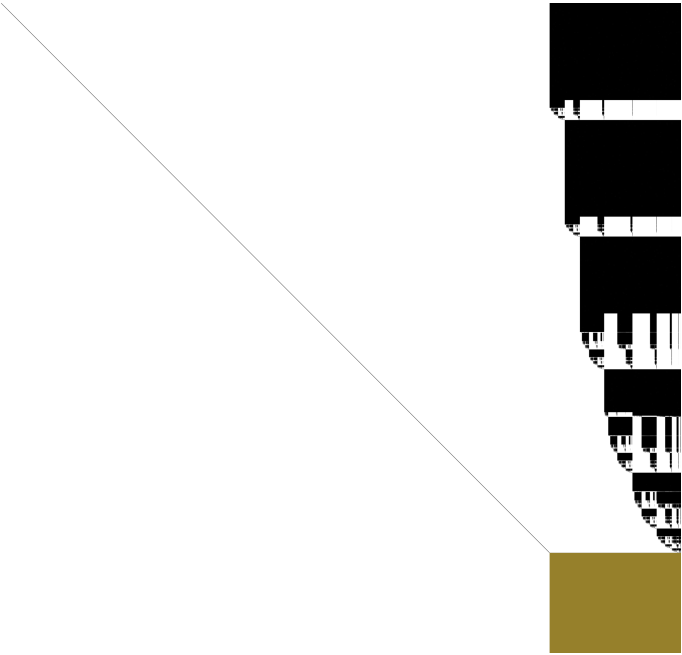
column order = monomial order

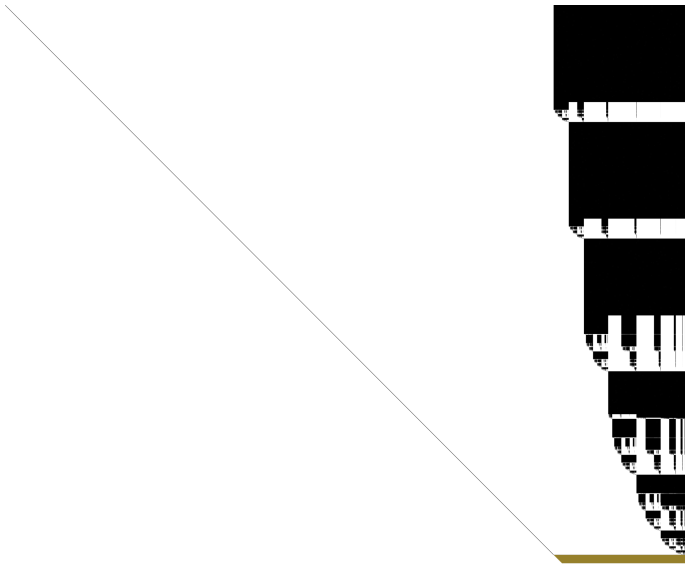
Idea

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





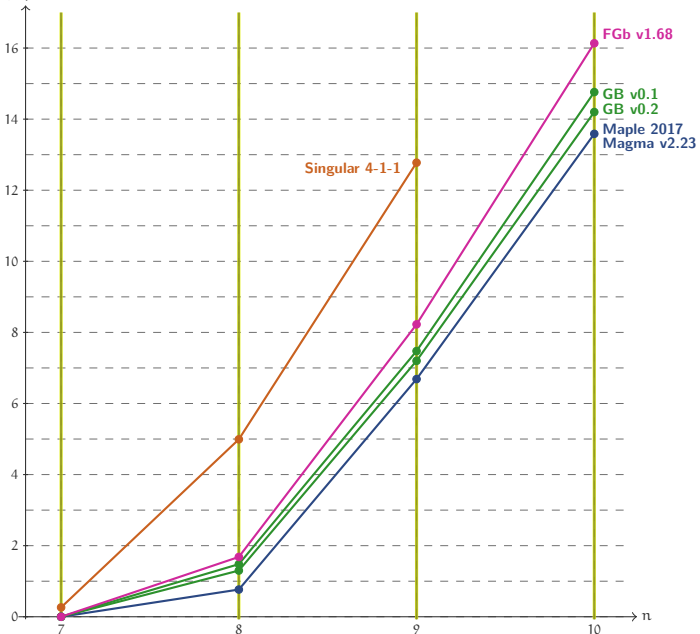




Parallel computation?

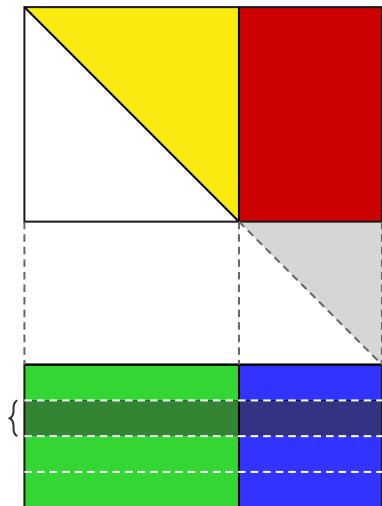
Cyclic-n / DRL / FF / SC

Time (log 2) in seconds

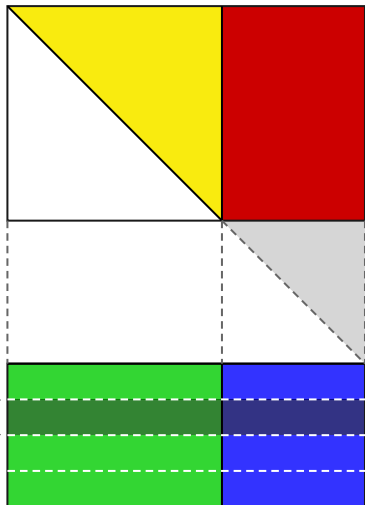


#5

Probabilistic linear algebra

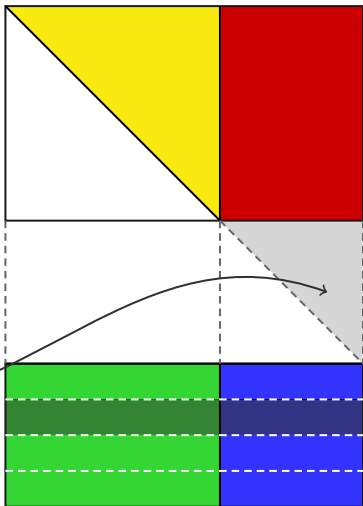


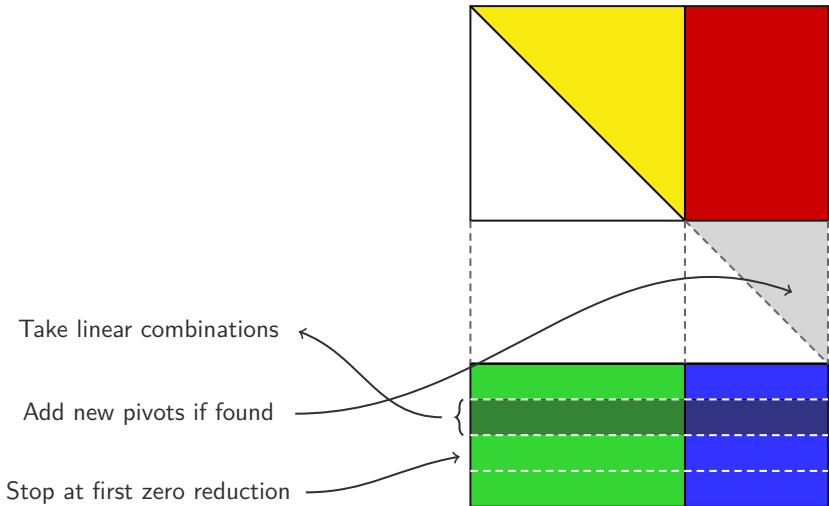
Take linear combinations



Take linear combinations

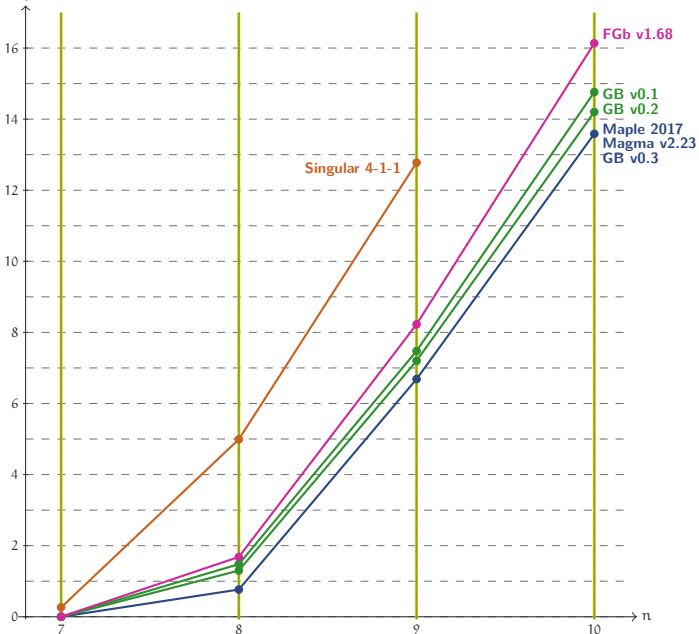
Add new pivots if found





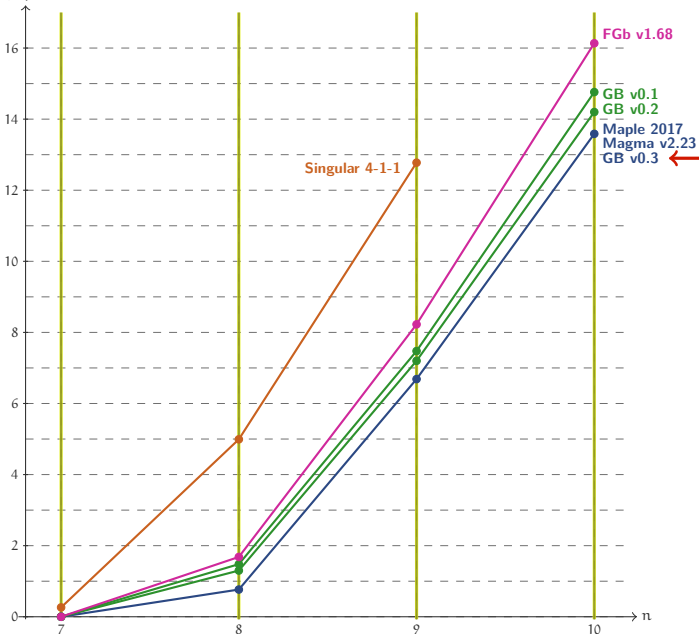
Cyclic-n / DRL / FF / SC

Time (log 2) in seconds



Cyclic-n / DRL / FF / SC

Time (log 2) in seconds



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?