Recent Advancements in Noncommutative Gröbner Basis Software

Clemens Hofstadler
Institute for Symbolic Artificial Intelligence, JKU Linz, Austria
Heraklion, 15 July 2025

based on joint work with Maximilian Heisinger

JOHANNES KEPLER UNIVERSITY LINZ

FWF Der Wissenschaftsfonds.

Noncommutative polynomials

noncommutative = really noncommutative

Noncommutative polynomials

$$\begin{aligned}\text{noncommutative} &= \text{really noncommutative}\\ &= \text{no commutation rules}\end{aligned}$$

Noncommutative polynomials

$$\begin{aligned} \text{noncommutative} &= \text{really noncommutative} \\ &= \text{no commutation rules} \\ &= \text{free algebra } K\langle x_1, \ldots, x_n \rangle \end{aligned}$$

Noncommutative polynomials

$$\begin{aligned}\text{noncommutative} &= \text{really noncommutative}\\ &= \text{no commutation rules}\\ &= \text{free algebra } K\langle x_1, \dots, x_n\rangle\end{aligned}$$

Noncom. polynomial $\quad c_1 \cdot w_1 + \cdots + c_d \cdot w_d \quad \in K\langle x_1, \dots, x_n\rangle$

Noncommutative polynomials

noncommutative = really noncommutative

= no commutation rules

= free algebra $K\langle x_1, \ldots, x_n \rangle$

Noncom. polynomial $c_1 \cdot w_1 + \cdots + c_d \cdot w_d \in K\langle x_1, \ldots, x_n \rangle$

words over $x_1, \ldots, x_n$

Noncommutative polynomials

$$\begin{aligned} \text{noncommutative} &= \text{really noncommutative} \\ &= \text{no commutation rules} \\ &= \text{free algebra } K\langle x_1, \ldots, x_n \rangle \end{aligned}$$

Noncom. polynomial $\quad c_1 \cdot w_1 + \cdots + c_d \cdot w_d \in K\langle x_1, \ldots, x_n \rangle$

words over $x_1, \ldots, x_n$

Example $\quad xyyx + 2xy - yx - 2 \in \mathbb{Q}\langle x, y \rangle$

Noncommutative polynomials

$$\begin{aligned} \text{noncommutative} &= \text{really noncommutative} \\ &= \text{no commutation rules} \\ &= \text{free algebra } K\langle x_1, \dots, x_n \rangle \end{aligned}$$

Noncom. polynomial $\quad c_1 \cdot w_1 + \cdots + c_d \cdot w_d \in K\langle x_1, \dots, x_n \rangle$

words over $x_1, \dots, x_n$

Example $\quad xyyx + 2xy - yx - 2 \in \mathbb{Q}\langle x, y \rangle$

$$\begin{aligned} \text{Multiplication} &= \text{concatenation of words} \\ (xy - 1) \cdot (yx + 2) &= xyyx + 2xy - yx - 2 \end{aligned}$$

Noncommutative polynomials

noncommutative = really noncommutative

= no commutation rules

= free algebra $K\langle x_1, \ldots, x_n \rangle$

Noncom. polynomial $c_1 \cdot w_1 + \cdots + c_d \cdot w_d \in K\langle x_1, \ldots, x_n \rangle$

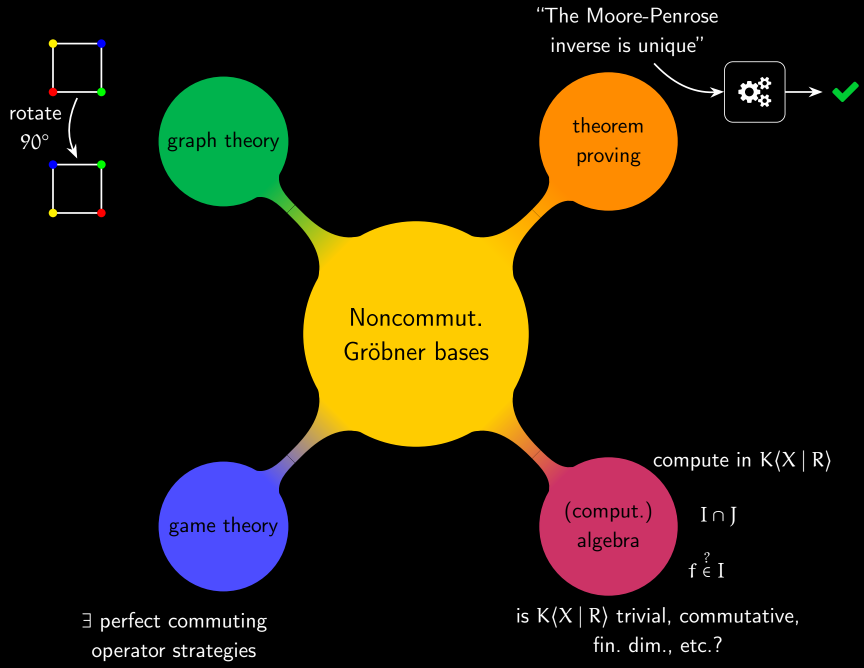
words over $x_1, \ldots, x_n$

Example $xyyx + 2xy - yx - 2 \in \mathbb{Q}\langle x, y \rangle$

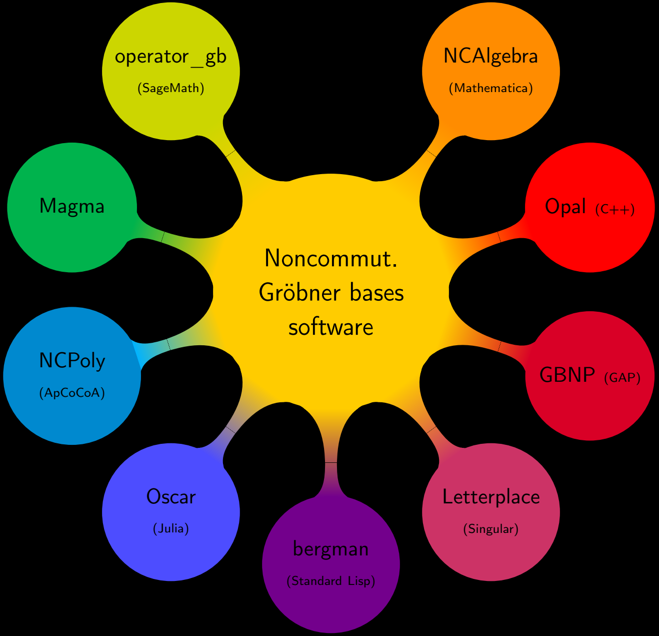
Multiplication = concatenation of words

$(xy - 1) \cdot (yx + 2) = xyyx + 2xy - yx - 2$

Noncomm. GB theory = comm. GB theory − finiteness

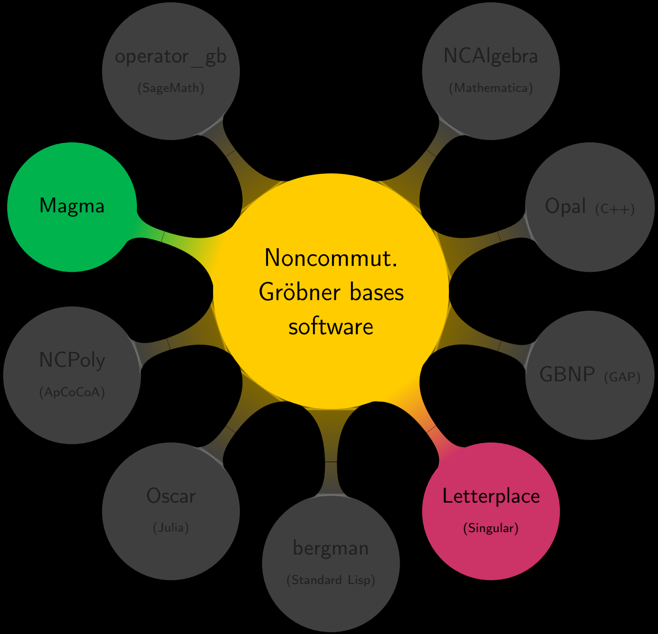


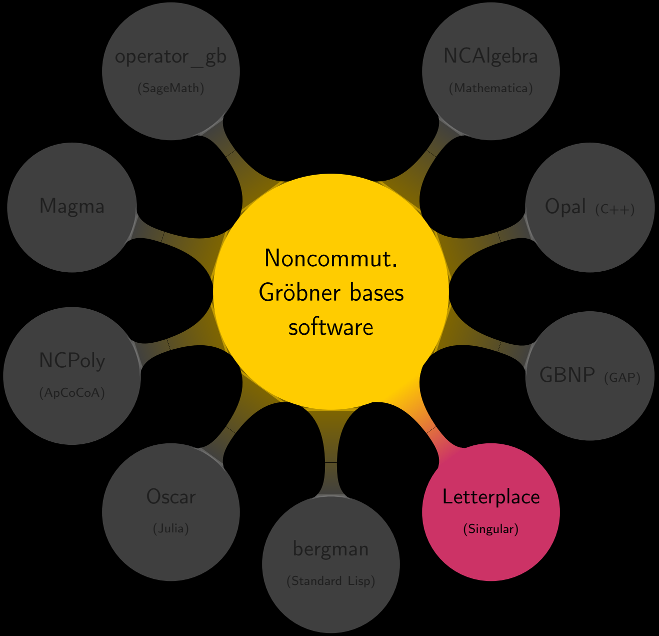
"The Moore-Penrose inverse is unique"
rotate 90°
graph theory
theorem proving
Noncommut. Gröbner bases
game theory
(comput.) algebra
compute in $K\langle X \mid R\rangle$
$I \cap J$
$f \overset{?}{\in} I$
$\exists$ perfect commuting operator strategies
is $K\langle X \mid R\rangle$ trivial, commutative, fin. dim., etc.?



Noncommut. Gröbner bases software
operator_gb (SageMath)
NCAlgebra (Mathematica)
Magma
Opal (C++)
NCPoly (ApCoCoA)
GBNP (GAP)
Oscar (Julia)
bergman (Standard Lisp)
Letterplace (Singular)

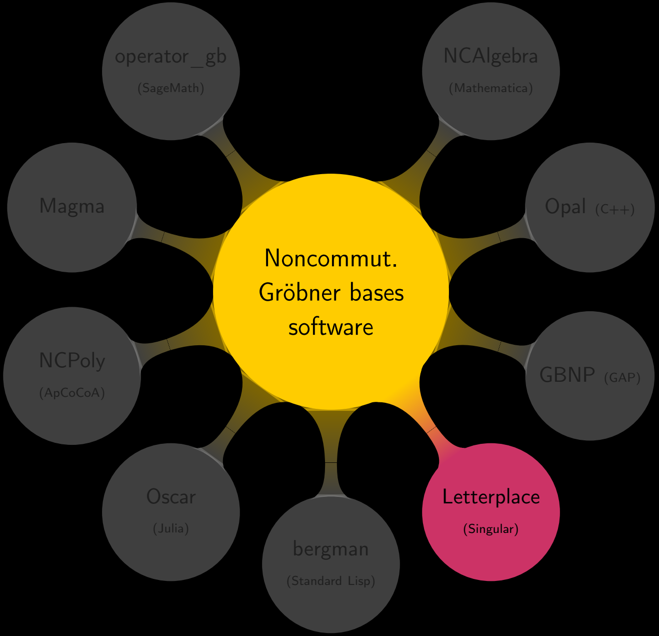

Noncommut.
Gröbner bases
software
operator_gb
(SageMath)
NCAlgebra
(Mathematica)
Magma
Opal (C++)
NCPoly
(ApCoCoA)
GBNP (GAP)
Oscar
(Julia)
bergman
(Standard Lisp)
Letterplace
(Singular)

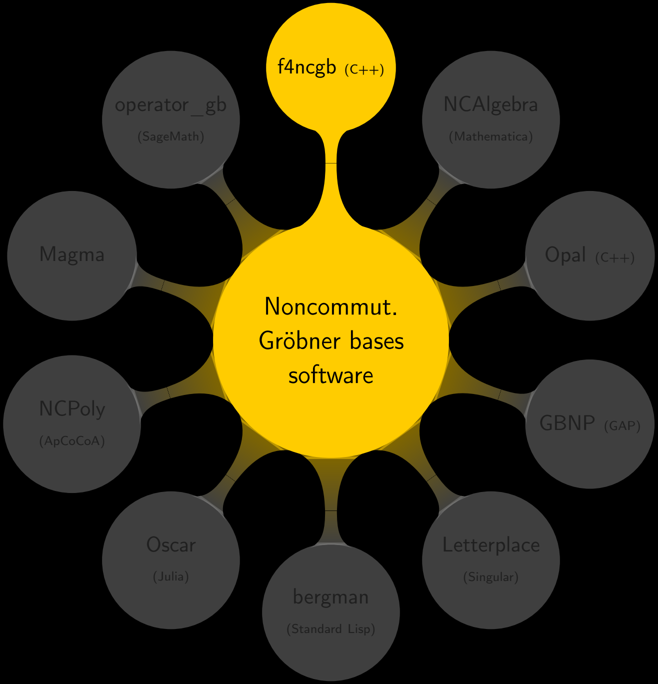

Noncommut.
Gröbner bases
software
operator_gb
(SageMath)
NCAlgebra
(Mathematica)
Magma
Opal (C++)
NCPoly
(ApCoCoA)
GBNP (GAP)
Oscar
(Julia)
bergman
(Standard Lisp)
Letterplace
(Singular)


Noncommut. Gröbner bases software
operator_gb
(SageMath)
NCAlgebra
(Mathematica)
Magma
Opal (C++)
NCPoly
(ApCoCoA)
GBNP (GAP)
Oscar
(Julia)
bergman
(Standard Lisp)
Letterplace
(Singular)

Example	Letterplace	f4ncgb 1 core	f4ncgb 4 cores	f4ncgb 16 cores
4nilp5s-10	1282	150	79	63
braid3-16	18 953	105	34	18
braidXY-12	1847	62	52	52
holt_G3562h-17	>43 200	25 021	12 671	6824
lascala_neuh-13	171	9	5	4
lp1-15	24 166	266	179	155
lv2d10-100	>43 200	48	27	47
malle_G12h-100	4142	89	74	73

(Timings in sec)


Noncommut. Gröbner bases software
operator_gb
(SageMath)
NCAlgebra
(Mathematica)
Magma
Opal (C++)
NCPoly
(ApCoCoA)
GBNP (GAP)
Oscar
(Julia)
bergman
(Standard Lisp)
Letterplace
(Singular)


Noncommut.
Gröbner bases
software
f4ncgb (C++)
NCAlgebra
(Mathematica)
Opal (C++)
GBNP (GAP)
Letterplace
(Singular)
bergman
(Standard Lisp)
Oscar
(Julia)
NCPoly
(ApCoCoA)
Magma
operator_gb
(SageMath)

f4ncgb

Open-source C++ library that ports commutative advancements to the noncommutative setting.

- Gröbner basis computation in $\mathbb{Q}\langle X\rangle$ and $\mathbb{Z}_p\langle X\rangle$ for prime $p < 2^{31}$
- Several orders of magnitude faster than current state of the art
- Proof logging via cofactor representations
- Soon part of OSCAR SYMBOLIC TOOLS

Data structures

- Monomials are unique and referenced
- Coefficients are shared
- Prefix tree for divisions

Algorithms

- Noncomm. F4 algorithm
- Sparse linear algebra (multi-modular, parallelized, probabilistic)
- Proof logging

f4ncgb

Data structures

- Monomials are unique and referenced
- Coefficients are shared
- Prefix tree for divisions

Algorithms

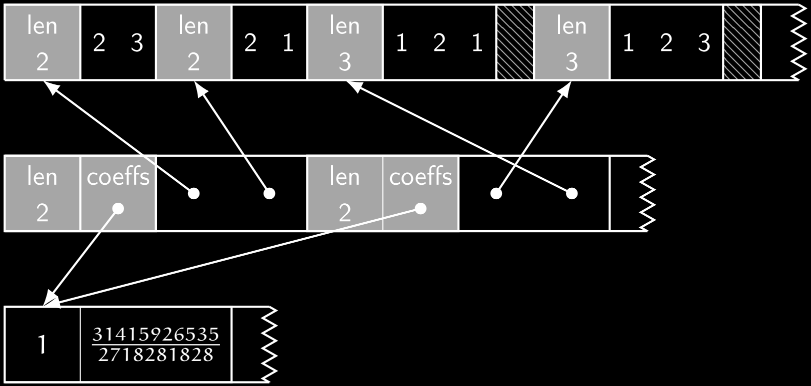
- Noncomm. F4 algorithm
- Sparse linear algebra (multi-modular, parallelized, probabilistic)
- Proof logging

f4ncgb

Monomials & Polynomials

Represent vars by index according to mon. order: $x_1 \prec x_2 \prec \cdots \prec x_n$

$\downarrow$ 1 $\downarrow$ 2 $\downarrow$ n



Monomial divisibility tests

Observation: divisor candidates are always the same (lm's of the GB)

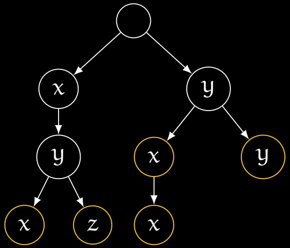
Exploit this information $\rightsquigarrow$ keep prefix tree of all leading monomials

Monomial divisibility tests

Observation: divisor candidates are always the same (lm's of the GB)

Exploit this information $\rightsquigarrow$ keep prefix tree of all leading monomials

$$G = \{\, xyx - \ldots,\ xyz + \ldots,\ yx + \ldots,\ yxx - \ldots,\ yy + \ldots \}$$

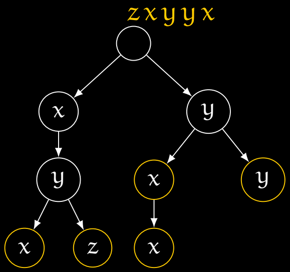


Monomial divisibility tests

Observation: divisor candidates are always the same (lm's of the GB)

Exploit this information $\rightsquigarrow$ keep prefix tree of all leading monomials

$$G = \{\, xyx - \ldots,\; xyz + \ldots,\; yx + \ldots,\; yxx - \ldots,\; yy + \ldots \}$$

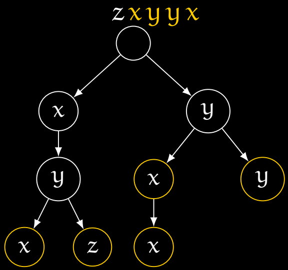


Monomial divisibility tests

Observation: divisor candidates are always the same (lm's of the GB)

Exploit this information $\rightsquigarrow$ keep prefix tree of all leading monomials

$$G = \{\, xyx - \ldots,\; xyz + \ldots,\; yx + \ldots,\; yxx - \ldots,\; yy + \ldots \}$$

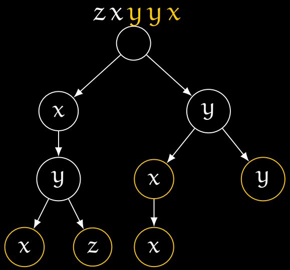


Monomial divisibility tests

Observation: divisor candidates are always the same (lm's of the GB)

Exploit this information ⇝ keep prefix tree of all leading monomials

$$G = \{\, xyx - \dots,\; xyz + \dots,\; yx + \dots,\; yxx - \dots,\; yy + \dots \}$$

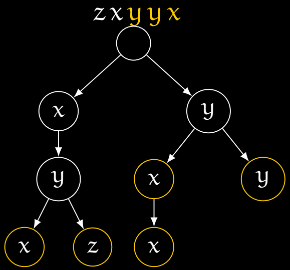


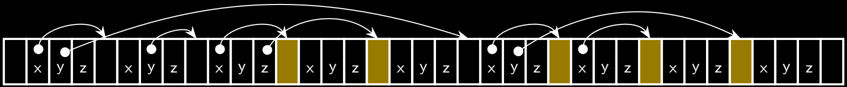
Monomial divisibility tests

Observation: divisor candidates are always the same (lm's of the GB)

Exploit this information $\rightsquigarrow$ keep prefix tree of all leading monomials

$$G = \{\, xyx - \ldots,\; xyz + \ldots,\; yx + \ldots,\; yxx - \ldots,\; yy + \ldots \,\}$$





Data structures

- Monomials are unique and referenced
- Coefficients are shared
- Prefix tree for divisions

Algorithms

- Noncomm. F4 algorithm
- Sparse linear algebra (multi-modular, parallelized, probabilistic)
- Proof logging

f4ncgb

Data structures

- Monomials are unique and referenced
- Coefficients are shared
- Prefix tree for divisions

Algorithms

- Noncomm. F4 algorithm
- Sparse linear algebra (multi-modular, parallelized, probabilistic)
- Proof logging

f4ncgb

Proof logging

Proof logging

Given input $f_1, \ldots, f_r$, write each $g \in GB$ as

$$g = \sum_j p_j \cdot f_j \cdot q_j$$

"*cofactor representation*"

with $p_j, q_j \in K\langle X \rangle$.

Proof logging

Given input $f_1, \ldots, f_r$, write each $g \in GB$ as

$$g = \sum_j p_j \cdot f_j \cdot q_j$$

"*cofactor representation*"

with $p_j, q_j \in K\langle X\rangle$.

Cofactor representations certify ideal membership.

Can be computed during Gaussian elimination:

Proof logging

Given input $f_1, \ldots, f_r$, write each $g \in GB$ as

$$g = \sum_j p_j \cdot f_j \cdot q_j \qquad \text{“\textit{cofactor representation}”}$$

with $p_j, q_j \in K\langle X \rangle$.

Cofactor representations certify ideal membership.

Can be computed during Gaussian elimination:

$$\begin{pmatrix} — & p_1 & — \\ & \vdots & \\ — & p_k & — \end{pmatrix} \rightsquigarrow \quad \text{RRef}$$

Proof logging

Given input $f_1, \ldots, f_r$, write each $g \in GB$ as

$$g = \sum_j p_j \cdot f_j \cdot q_j \qquad \text{“\textit{cofactor representation}”}$$

with $p_j, q_j \in K\langle X \rangle$.

Cofactor representations certify ideal membership.

Can be computed during Gaussian elimination:

$$T \cdot \begin{pmatrix} — & p_1 & — \\ & \vdots & \\ — & p_k & — \end{pmatrix} = \text{RRef}$$

The rows of T give cofactor representations of g_i in terms of $f_1, \ldots, f_r$ and $g_1, \ldots, g_{i-1}$.

Substitution yields representations w.r.t. $f_1, \ldots, f_r$ (⚠ exp. blowup!)

Data structures

- Monomials are unique and referenced
- Coefficients are shared
- Prefix tree for divisions

Algorithms

- Noncomm. F4 algorithm
- Sparse linear algebra (multi-modular, parallelized, probabilistic)
- Proof logging

f4ncgb

Next steps

Next steps

Signature-based algorithms

- Extend polynomials by signatures
- Avoid (almost) all reductions to zero
- Syzygy basis for free

Next steps

Signature-based algorithms

- Extend polynomials by signatures
- Avoid (almost) all reductions to zero
- Syzygy basis for free

Coefficient rings

- Computations over $\mathbb{Z}$
- Folklore: RRef $\rightsquigarrow$ HNF should be enough
- Which lin. algebra tricks still apply?

Next steps

Signature-based algorithms

- Extend polynomials by signatures
- Avoid (almost) all reductions to zero
- Syzygy basis for free

Coefficient rings

- Computations over $\mathbb{Z}$
- Folklore: RRef $\rightsquigarrow$ HNF should be enough ?
- Which lin. algebra tricks still apply?

Next steps

Signature-based algorithms

- Extend polynomials by signatures
- Avoid (almost) all reductions to zero
- Syzygy basis for free

Coefficient rings

- Computations over $\mathbb{Z}$
- Folklore: RRef $\rightsquigarrow$ HNF should be enough !
- Which lin. algebra tricks still apply?

f4ncgb

Open-source C++ library that ports commutative advancements to the noncommutative setting.

- Gröbner basis computation in $\mathbb{Q}\langle X\rangle$ and $\mathbb{Z}_p\langle X\rangle$ for prime $p < 2^{31}$
- Several orders of magnitude faster than current state of the art
- Proof logging via cofactor representations
- Soon part of OSCAR SYMBOLIC TOOLS