F5C: A variant of Faugère's F5 algorithm with reduced Gröbner bases

Christian Eder
(joint work with John Perry)

Technische Universität Kaiserslautern

June 16th, 2009

What is this talk all about?

1. Efficient computations of Gröbner bases using Faugère's F5 Algorithm and variants of it
2. Explanation of the F5 Algorithm, its criteria used to detect useless pairs, and its points of **inefficiency**
3. Presentation of the variant F5C which reduces the stated inefficiencies of F5
4. Comparison of the variants of F5 under several aspects

What is this talk all about?

1. Efficient computations of Gröbner bases using Faugère's F5 Algorithm and variants of it
2. Explanation of the F5 Algorithm, its criteria used to detect useless pairs, and its points of **inefficiency**
3. Presentation of the variant F5C which reduces the stated inefficiencies of F5
4. Comparison of the variants of F5 under several aspects

Remark

These **ineffiencies** are the computations of polynomials **redundant** for the Gröbner basis G, i.e. polynomials whose head monomials are multiples of head monomials of other elements already in G.

The following section is about

Main property of Göbner bases

Lemma

Let G be a Gröbner basis of an ideal I. Then for all elements $g_i, g_j \in G$ it holds that

$$\mathrm{Spol}(g_i, g_j) \xrightarrow{G} 0,$$

where

- $\mathrm{Spol}(g_i, g_j) = \mathrm{hc}(g_j)u_i g_i - \mathrm{hc}(g_i)u_j g_j$ *and*
- $u_k = \frac{\mathrm{lcm}\big(\mathrm{hm}(g_i),\mathrm{hm}(g_j)\big)}{\mathrm{hm}(g_k)}$ *for* $k \in \{i, j\}$.

Computation of Gröbner bases

The standard **Buchberger Algorithm** to compute G follows easily from the previous stated property of G:

Input: Ideal $I = \langle f_1, \ldots, f_m \rangle$

Output: Gröbner basis G of I

1. $G = \emptyset$
2. $G := G \cup \{f_i\}$ for all $i \in \{1, \ldots, m\}$
3. Set $P := \{\mathrm{Spol}(g_i, g_j) \mid g_i, g_j \in G, i \neq j\}$

Computation of Gröbner bases

The standard **Buchberger Algorithm** to compute G follows easily from the previous stated property of G:

Input: Ideal $I = \langle f_1, \ldots, f_m \rangle$

Output: Gröbner basis G of I

1. $G = \emptyset$
2. $G := G \cup \{f_i\}$ for all $i \in \{1, \ldots, m\}$
3. Set $P := \{\mathrm{Spol}(g_i, g_j) \mid g_i, g_j \in G, i \neq j\}$
4. Choose one element $p \in P$, $P := P \setminus \{p\}$

Computation of Gröbner bases

The standard **Buchberger Algorithm** to compute G follows easily from the previous stated property of G:

Input: Ideal $I = \langle f_1, \ldots, f_m \rangle$

Output: Gröbner basis G of I

1. $G = \emptyset$
2. $G := G \cup \{f_i\}$ for all $i \in \{1, \ldots, m\}$
3. Set $P := \{\mathrm{Spol}(g_i, g_j) \mid g_i, g_j \in G, i \neq j\}$
4. Choose one element $p \in P$, $P := P \setminus \{p\}$
   (a) If $p \xrightarrow{G} 0 \Rightarrow$ **no new information**
   Go on with the next element in P.

Computation of Gröbner bases

The standard **Buchberger Algorithm** to compute G follows easily from the previous stated property of G:

Input: Ideal $I = \langle f_1, \ldots, f_m \rangle$

Output: Gröbner basis G of I

1. $G = \emptyset$
2. $G := G \cup \{f_i\}$ for all $i \in \{1, \ldots, m\}$
3. Set $P := \{\mathrm{Spol}(g_i, g_j) \mid g_i, g_j \in G, i \neq j\}$
4. Choose one element $p \in P$, $P := P \setminus \{p\}$
   (a) If $p \xrightarrow{G} 0 \Rightarrow$ **no new information**
   Go on with the next element in P.
   (b) If $p \xrightarrow{G} h \neq 0 \Rightarrow$ **new information**
   Add h to G.
   Build new S-polynomials with h and add them to P.
   Go on with the next element in P.

Computation of Gröbner bases

The standard **Buchberger Algorithm** to compute G follows easily from the previous stated property of G:

Input: Ideal $I = \langle f_1, \dots, f_m \rangle$

Output: Gröbner basis G of I

1. $G = \emptyset$
2. $G := G \cup \{f_i\}$ for all $i \in \{1, \dots, m\}$
3. Set $P := \{\mathrm{Spol}(g_i, g_j) \mid g_i, g_j \in G, i \neq j\}$
4. Choose one element $p \in P$, $P := P \setminus \{p\}$
   (a) If $p \xrightarrow{G} 0 \Rightarrow$ **no new information**
   Go on with the next element in P.
   (b) If $p \xrightarrow{G} h \neq 0 \Rightarrow$ **new information**
   Add h to G.
   Build new S-polynomials with h and add them to P.
   Go on with the next element in P.
5. When there is no pair left we are done and G is a Gröbner basis of I.

An example of zero reduction

Example

Assume the ideal $I = \langle g_1, g_2 \rangle \lhd \mathbb{Q}[x, y, z]$ where $g_1 = xy - z^2$, $g_2 = y^2 - z^2$.

An example of zero reduction

Example

Assume the ideal $I = \langle g_1, g_2 \rangle \lhd \mathbb{Q}[x, y, z]$ where $g_1 = xy - z^2$, $g_2 = y^2 - z^2$.
Computing

$$\mathrm{Spol}(g_2, g_1) = \mathbf{xy^2} - xz^2 - \mathbf{xy^2} + yz^2 = -xz^2 + yz^2,$$

we get a new element $g_3 = xz^2 - yz^2$ for G.

An example of zero reduction

Example

Assume the ideal $I = \langle g_1, g_2 \rangle \lhd \mathbb{Q}[x, y, z]$ where $g_1 = xy - z^2$, $g_2 = y^2 - z^2$.
Computing

$$\mathrm{Spol}(g_2, g_1) = \mathbf{xy^2} - xz^2 - \mathbf{xy^2} + yz^2 = -xz^2 + yz^2,$$

we get a new element $g_3 = xz^2 - yz^2$ for G.
Let us compute $\mathrm{Spol}(g_3, g_1)$ next:

An example of zero reduction

Example

Assume the ideal $I = \langle g_1, g_2 \rangle \lhd \mathbb{Q}[x, y, z]$ where $g_1 = xy - z^2$, $g_2 = y^2 - z^2$.

Computing

$$\mathrm{Spol}(g_2, g_1) = \mathbf{xy^2} - xz^2 - \mathbf{xy^2} + yz^2 = -xz^2 + yz^2,$$

we get a new element $g_3 = xz^2 - yz^2$ for G.

Let us compute $\mathrm{Spol}(g_3, g_1)$ next:

$$\mathrm{Spol}(g_3, g_1) = \mathbf{xyz^2} - y^2z^2 - \mathbf{xyz^2} + z^4 = -y^2z^2 + z^4.$$

An example of zero reduction

Example

Assume the ideal $I = \langle g_1, g_2 \rangle \lhd \mathbb{Q}[x, y, z]$ where $g_1 = xy - z^2$, $g_2 = y^2 - z^2$.
Computing

$$\mathrm{Spol}(g_2, g_1) = \mathbf{xy^2} - xz^2 - \mathbf{xy^2} + yz^2 = -xz^2 + yz^2,$$

we get a new element $g_3 = xz^2 - yz^2$ for G.
Let us compute $\mathrm{Spol}(g_3, g_1)$ next:

$$\mathrm{Spol}(g_3, g_1) = \mathbf{xyz^2} - y^2z^2 - \mathbf{xyz^2} + z^4 = -y^2z^2 + z^4.$$

Now we can reduce further with $z^2 g_2$:

$$-y^2z^2 + z^4 + y^2z^2 - z^4 = 0.$$

The following section is about

Example revisited - with signatures

Faugère's idea is to give each generator f_i of the initial ideal the **signature** $\mathcal{S}(f_i) = (1, \mathrm{i})$.
Moreover, each element being newly computed in the algorithm gets the signature of the S-polynomial it comes from.

Example revisited - with signatures

Faugère's idea is to give each generator f_i of the initial ideal the **signature** $\mathcal{S}(f_i) = (1, \mathrm{i})$.
Moreover, each element being newly computed in the algorithm gets the signature of the S-polynomial it comes from.
In our example

$$\begin{aligned} g_3 &= \mathrm{Spol}(g_2, g_1) = xg_2 - yg_1 \\ \Rightarrow \mathcal{S}(g_3) &= x\mathcal{S}(g_2) = x(1,2) := (x, 2). \end{aligned}$$

Example revisited - with signatures

Faugère's idea is to give each generator f_i of the initial ideal the **signature** $\mathcal{S}(f_i) = (1, \mathrm{i})$.
Moreover, each element being newly computed in the algorithm gets the signature of the S-polynomial it comes from.
In our example

$$g_3 = \mathrm{Spol}(g_2, g_1) = xg_2 - yg_1$$
$$\Rightarrow \mathcal{S}(g_3) = x\mathcal{S}(g_2) = x(1, 2) := (x, 2).$$

It follows that $\mathrm{Spol}(g_3, g_1) = yg_3 - zg_1$ has

$$\mathcal{S}\left(\mathrm{Spol}(g_3, g_1)\right) = y\mathcal{S}(g_3) = (xy, 2).$$

Example revisited - with signatures

Faugère's idea is to give each generator f_i of the initial ideal the **signature** $\mathcal{S}(f_i) = (1, \mathrm{i})$.
Moreover, each element being newly computed in the algorithm gets the signature of the S-polynomial it comes from.
In our example

$$g_3 = \mathrm{Spol}(g_2, g_1) = xg_2 - yg_1$$
$$\Rightarrow \mathcal{S}(g_3) = x\mathcal{S}(g_2) = x(1,2) := (x, 2).$$

It follows that $\mathrm{Spol}(g_3, g_1) = yg_3 - zg_1$ has

$$\mathcal{S}\left(\mathrm{Spol}(g_3, g_1)\right) = y\mathcal{S}(g_3) = (xy, 2).$$

Now we see that $\mathcal{S}\left(\mathrm{Spol}(g_3, g_1)\right) = (xy, 2)$ and $\mathrm{hm}(g_1) = xy$.

Example revisited - with signatures

Faugère's idea is to give each generator f_i of the initial ideal the **signature** $\mathcal{S}(f_i) = (1, \mathrm{i})$.
Moreover, each element being newly computed in the algorithm gets the signature of the S-polynomial it comes from.
In our example

$$g_3 = \mathrm{Spol}(g_2, g_1) = xg_2 - yg_1$$
$$\Rightarrow \mathcal{S}(g_3) = x\mathcal{S}(g_2) = x(1,2) := (x,2).$$

It follows that $\mathrm{Spol}(g_3, g_1) = yg_3 - zg_1$ has

$$\mathcal{S}\left(\mathrm{Spol}(g_3, g_1)\right) = y\mathcal{S}(g_3) = (xy, 2).$$

Now we see that $\mathcal{S}\left(\mathrm{Spol}(g_3, g_1)\right) = (xy, 2)$ and $\mathrm{hm}(g_1) = xy$.
$\Rightarrow$ In F5 we **know** that $\mathrm{Spol}(g_3, g_1)$ will reduce to zero!

How does this work?

To understand the criteria of F5 on which this knowledge of zero reduction is based on we first need to give a general overview of a slightly different approach of implementing a Gröbner basis algorithm:

Computing Gröbner bases incrementally

Incremental nature of the F5 Algorithm

Input: Ideal $I = \langle f_1, \ldots, f_m \rangle$
Output: Gröbner basis G of I

Incremental nature of the F5 Algorithm

Input: Ideal $I = \langle f_1, \dots, f_m \rangle$
Output: Gröbner basis G of I

1. Compute Gröbner basis G_1 of $\langle f_1 \rangle$.
2. Compute Gröbner basis G_2 of $\langle f_1, f_2 \rangle$.
3. ...

Incremental nature of the F5 Algorithm

Input: Ideal $I = \langle f_1, \ldots, f_m \rangle$
Output: Gröbner basis G of I

1. Compute Gröbner basis G_1 of $\langle f_1 \rangle$.
2. Compute Gröbner basis G_2 of $\langle f_1, f_2 \rangle$.
3. ...

Remark

Note that from this point on $f_i = g_i$ is no longer true for all $i \in \{1, \ldots, m\}$, due to possible intermediate computations of S-polynomials.

F5 and Rewritten Criterion

Theorem (F5 Criterion)

An S-polynomial $\mathrm{Spol}(g_i, g_j) = u_i g_i - u_j g_j$ *does not need to be computed, let alone reduced, if for* $k \in \{i, j\}$ *and* $\mathcal{S}(g_k) = (t_k, \ell_k)$ *there exists an element* g *in* $G_{\ell_k - 1}$ *such that*

$$\mathrm{hm}(g) \mid u_k t_k.$$

F5 and Rewritten Criterion

Theorem (F5 Criterion)

An S-polynomial $\mathrm{Spol}(g_i, g_j) = u_i g_i - u_j g_j$ *does not need to be computed, let alone reduced, if for* $k \in \{i, j\}$ *and* $\mathcal{S}(g_k) = (t_k, \ell_k)$ *there exists an element* g *in* $G_{\ell_k - 1}$ *such that*

$$\mathrm{hm}(g) \mid u_k t_k.$$

Theorem (Rewritten Criterion)

An S-polynomial $\mathrm{Spol}(g_i, g_j) = u_i g_i - u_j g_j$ *does not need to be computed, let alone reduced, if for* $k \in \{i, j\}$ *and* $\mathcal{S}(g_k) = (t_k, \ell_k)$ *there exists an element* g_ν *with* $\mathcal{S}(g_\nu) = (t_\nu, \ell_k)$ *in* G *such that*

$$\nu > k \qquad \text{and} \qquad t_\nu \mid u_k t_k.$$

Complexity of top-reduction in F5

On the one hand adding signatures to polynomials makes it possible to use these powerful criteria,
on the other hand we have to keep track of the signatures, i.e. we must be very careful when reducing elements.

Complexity of top-reduction in F5

On the one hand adding signatures to polynomials makes it possible to use these powerful criteria,
on the other hand we have to keep track of the signatures, i.e. we must be very careful when reducing elements.

Example

Assume the polynomial $g_i = xy^2 - z^3$ with $\mathcal{S}(g_i) = (xy^2, \ell)$ and a possible reducer $g_j = y^2 - xz$ with $\mathcal{S}(g_j) = (t_j, \ell)$.

Complexity of top-reduction in F5

On the one hand adding signatures to polynomials makes it possible to use these powerful criteria,
on the other hand we have to keep track of the signatures, i.e. we must be very careful when reducing elements.

Example

Assume the polynomial $g_i = xy^2 - z^3$ with $\mathcal{S}(g_i) = (xy^2, \ell)$ and a possible reducer $g_j = y^2 - xz$ with $\mathcal{S}(g_j) = (t_j, \ell)$.
Note that the signatures of both polynomials have **the same index**.

Complexity of top-reduction in F5

On the one hand adding signatures to polynomials makes it possible to use these powerful criteria,
on the other hand we have to keep track of the signatures, i.e. we must be very careful when reducing elements.

Example

Assume the polynomial $g_i = xy^2 - z^3$ with $\mathcal{S}(g_i) = (xy^2, \ell)$ and a possible reducer $g_j = y^2 - xz$ with $\mathcal{S}(g_j) = (t_j, \ell)$.
Note that the signatures of both polynomials have **the same index**. In Buchberger-like implementations the top-reduction would take place, i.e. we would compute $g_i - xg_j$.

Complexity of top-reduction in F5

Example

In F5 the following can happen:

1. If xg_j satisfies the F5 Criterion $\Rightarrow$ **no reduction**!

Complexity of top-reduction in F5

Example

In F5 the following can happen:

1. If xg_j satisfies the F5 Criterion $\Rightarrow$ **no reduction**!
2. If xg_j satisfies the Rewritten Criterion $\Rightarrow$ **no reduction**!

Complexity of top-reduction in F5

Example

In F5 the following can happen:

1. If xg_j satisfies the F5 Criterion $\Rightarrow$ **no reduction**!
2. If xg_j satisfies the Rewritten Criterion $\Rightarrow$ **no reduction**!
3. None of the above cases holds and $xt_j < xy^2 \Rightarrow g_i - xg_j$ is computed with the signature (xy^2, ℓ).

Complexity of top-reduction in F5

Example

In F5 the following can happen:

1. If xg_j satisfies the F5 Criterion $\Rightarrow$ **no reduction**!
2. If xg_j satisfies the Rewritten Criterion $\Rightarrow$ **no reduction**!
3. None of the above cases holds and $xt_j < xy^2 \Rightarrow g_i - xg_j$ is computed with the signature (xy^2, ℓ).
4. None of the first two cases holds and $xt_j > xy^2 \Rightarrow$ **the signature of the reducer is greater than the signature of the to be reduced element**, which leads to

Complexity of top-reduction in F5

Example

In F5 the following can happen:

1. If xg_j satisfies the F5 Criterion $\Rightarrow$ **no reduction**!
2. If xg_j satisfies the Rewritten Criterion $\Rightarrow$ **no reduction**!
3. None of the above cases holds and $xt_j < xy^2 \Rightarrow g_i - xg_j$ is computed with the signature (xy^2, ℓ).
4. None of the first two cases holds and $xt_j > xy^2 \Rightarrow$ **the signature of the reducer is greater than the signature of the to be reduced element**, which leads to
   (a) **No reduction** of g_i, but searching for another possible reducer of it.

Complexity of top-reduction in F5

Example

In F5 the following can happen:

1. If xg_j satisfies the F5 Criterion $\Rightarrow$ **no reduction**!
2. If xg_j satisfies the Rewritten Criterion $\Rightarrow$ **no reduction**!
3. None of the above cases holds and $xt_j < xy^2 \Rightarrow g_i - xg_j$ is computed with the signature (xy^2, ℓ).
4. None of the first two cases holds and $xt_j > xy^2 \Rightarrow$ **the signature of the reducer is greater than the signature of the to be reduced element**, which leads to
   (a) **No reduction** of g_i, but searching for another possible reducer of it.
   (b) a new **S-polynomial** $g_{\mathrm{new}} := xg_j - g_i$ whereas $\mathcal{S}(g_{\mathrm{new}}) = (xt_j, \ell)$.

Redundant polynomials

Example

Assume that there is **no other reducer** of g_i.
$\Rightarrow$ In the first two cases g_i is added to G but $\mathrm{hm}(g_j) \mid \mathrm{hm}(g_i)$.
$\Rightarrow$ g_i is redundant for G.

Redundant polynomials

Example

Assume that there is **no other reducer** of g_i.
$\Rightarrow$ In the first two cases g_i is added to G but $\mathrm{hm}(g_j) \mid \mathrm{hm}(g_i)$.
$\Rightarrow$ g_i is redundant for G.

But...

For the F5 Algorithm itself and the criteria based on the signatures g_i could be necessary **in this iteration step**!
$\Rightarrow$ Disrespecting the way F5 top-reduces polynomials would harm the correctness of F5 **in this iteration step**!

Points of inefficiency

The complexity of top-reduction in F5 leads to an **inefficiency**, namely we have way too many polynomials in the intermediate G_is

1. which are possible reducers,
   $\Rightarrow$ more checks for divisibility and the criteria have to be done,
2. with which we compute newly S-polynomials.
   $\Rightarrow$ more (for the resulting Gröbner basis redundant) data is generated

Points of inefficiency

The complexity of top-reduction in F5 leads to an **inefficiency**, namely we have way too many polynomials in the intermediate G_is

1. which are possible reducers,
   $\Rightarrow$ more checks for divisibility and the criteria have to be done,
2. with which we compute newly S-polynomials.
   $\Rightarrow$ more (for the resulting Gröbner basis redundant) data is generated

Question

How can these two points be avoided as far as possible?

The following section is about

1. Introducing Gröbner bases
2. The F5 Algorithm
3. Optimizations of F5
   - F5R: F5 Algorithm Reducing by reduced Gröbner bases
   - F5C: F5 Algorithm Computing with reduced Gröbner bases
4. Comparison of the variants of F5

F5R: reduced GB reduction

An idea how to fix the first inefficiency, was given by Till Stegers in 2005. His slightly optimized F5 **using reduced Gröbner bases for reduction** is called **F5R** in the following:

F5R: reduced GB reduction

An idea how to fix the first inefficiency, was given by Till Stegers in 2005. His slightly optimized F5 **using reduced Gröbner bases for reduction** is called **F5R** in the following:

1. Compute a Gröbner basis G_i of $\langle f_1, \ldots, f_i \rangle$.
2. Compute the reduced Gröbner basis B_i of G_i.
3. Compute a Gröbner basis G_{i+1} of $\langle f_1, \ldots, f_{i+1} \rangle$ where
   (a) G_i is used to build the new pairs with f_{i+1},
   (b) B_i is used to reduce polynomials.

F5R: reduced GB reduction

An idea how to fix the first inefficiency, was given by Till Stegers in 2005. His slightly optimized F5 **using reduced Gröbner bases for reduction** is called **F5R** in the following:

1. Compute a Gröbner basis G_i of $\langle f_1, \ldots, f_i \rangle$.
2. Compute the reduced Gröbner basis B_i of G_i.
3. Compute a Gröbner basis G_{i+1} of $\langle f_1, \ldots, f_{i+1} \rangle$ where
   (a) G_i is used to build the new pairs with f_{i+1},
   (b) B_i is used to reduce polynomials.

$\Rightarrow$ **Fewer reductions** in F5R but still the **same number of pairs considered and polynomials generated** as in F5.

B_i only for reduction?

Question

Why is B_i only used for reduction purposes, but not for new-pair computations?

B_i only for reduction?

Question

Why is B_i only used for reduction purposes, but not for new-pair computations?

Answer

$$\boxed{\text{Interreducing } G_i \text{ to } B_i \leftrightarrow \text{reduction steps rejected by F5}}$$

B_i only for reduction?

Question

Why is B_i only used for reduction purposes, but not for new-pair computations?

Answer

$$\boxed{\text{Interreducing } G_i \text{ to } B_i \leftrightarrow \text{reduction steps rejected by F5}}$$

$\Rightarrow$ Reducing G_i to B_i renders the data saved in the **signatures** of the polynomials **useless**!

F5C: Computations with reduced GB

In 2008 John Perry & Christian Eder have implemented a new variant of the F5 Algorithm, called **F5C**.

F5C: Computations with reduced GB

In 2008 John Perry & Christian Eder have implemented a new variant of the F5 Algorithm, called **F5C**.
F5C uses the reduced Gröbner basis not only for reduction purposes, but also for the generation of new pairs:

F5C: Computations with reduced GB

In 2008 John Perry & Christian Eder have implemented a new variant of the F5 Algorithm, called **F5C**.
F5C uses the reduced Gröbner basis not only for reduction purposes, but also for the generation of new pairs:

1. Compute a Gröbner basis G_i of $\langle f_1, \ldots, f_i \rangle$.
2. Compute the reduced Gröbner basis B_i of G_i.
3. Compute a Gröbner basis G_{i+1} of $\langle f_1, \ldots, f_{i+1} \rangle$ where
   (a) B_i is used to build new pairs with f_{i+1},
   (b) B_i is used to reduce polynomials.

F5C: Computations with reduced GB

In 2008 John Perry & Christian Eder have implemented a new variant of the F5 Algorithm, called **F5C**.

F5C uses the reduced Gröbner basis not only for reduction purposes, but also for the generation of new pairs:

1. Compute a Gröbner basis G_i of $\langle f_1, \ldots, f_i \rangle$.
2. Compute the reduced Gröbner basis B_i of G_i.
3. Compute a Gröbner basis G_{i+1} of $\langle f_1, \ldots, f_{i+1} \rangle$ where
   - (a) B_i is used to build new pairs with f_{i+1},
   - (b) B_i is used to reduce polynomials.

$\Rightarrow$ **Fewer reductions than F5 & F5R** and **fewer polynomials generated and considered during the algorithm**

How to use B_i for computations?

We have seen that **if we interreduce** G_i then **the current signatures are useless** in the following.

How to use B_i for computations?

We have seen that **if we interreduce** G_i then **the current signatures are useless** in the following.
$\Rightarrow$ If the current signatures are useless, then **throw them away** and **compute new useful ones**!

How to use B_i for computations?

We have seen that **if we interreduce** G_i then **the current signatures are useless** in the following.
$\Rightarrow$ If the current signatures are useless, then **throw them away** and **compute new useful ones**!

Recomputation of signatures

How to use B_i for computations?

We have seen that **if we interreduce** G_i then **the current signatures are useless** in the following.
$\Rightarrow$ If the current signatures are useless, then **throw them away** and **compute new useful ones**!

Recomputation of signatures

1. Delete all signatures.
2. Interreduce G_i to B_i.
3. For each element $g_k \in B_i$ set $\mathcal{S}(g_k) = (1, \mathrm{k})$.
4. For all elements $g_j, g_k \in B_i$ **recompute signatures** for $\mathrm{Spol}(g_j, g_k)$.
5. Start the next iteration step with f_{i+1} by computing all pairs with elements from B_i.

Re-doing stuff is never nice

Recomputing the signatures of the S-polynomials in B_i is the only part of the optimization which seems to be annoying.

Re-doing stuff is never nice

Recomputing the signatures of the S-polynomials in B_i is the only part of the optimization which seems to be annoying.

Further improvement

In 2009 Perry & Eder have shown that **in F5C** it is **not necessary** to recompute the signatures of $\mathrm{Spol}(g_j, g_k)$ for $g_j, g_k \in B_i$.

Re-doing stuff is never nice

Recomputing the signatures of the S-polynomials in B_i is the only part of the optimization which seems to be annoying.

Further improvement

In 2009 Perry & Eder have shown that **in F5C** it is **not necessary** to recompute the signatures of $\mathrm{Spol}(g_j, g_k)$ for $g_j, g_k \in B_i$.
Thus as a last summary what we have to do after an intermediate Gröbner basis G_i is computed by F5:

Re-doing stuff is never nice

Recomputing the signatures of the S-polynomials in B_i is the only part of the optimization which seems to be annoying.

Further improvement

In 2009 Perry & Eder have shown that **in F5C** it is **not necessary** to recompute the signatures of $\mathrm{Spol}(g_j, g_k)$ for $g_j, g_k \in B_i$.
Thus as a last summary what we have to do after an intermediate Gröbner basis G_i is computed by F5:

1. Delete all signatures.
2. Interreduce G_i to B_i.
3. For each $g_k \in B_i$ set $\mathcal{S}(g_k) = (1, \mathrm{k})$.
4. Start the next iteration step with f_{i+1}.

The following section is about

Implementations

Three free available implementations:

1. F5, F5R & F5C as a SINGULAR library (Perry & Eder)
2. F5, F5R & F5C implemented in Python for Sage (Perry & Albrecht): **F4-ish** reduction possible.
3. F5, F5R & F5C implementation in the SINGULAR kernel: **under development**

We are comparing the three variants of F5 in the way that we use the **same implementation** of the **core algorithm** for all variants.

We are comparing the three variants of F5 in the way that we use the **same implementation** of the **core algorithm** for all variants.

Moreover we do not only compare

1. **timings**, but also
2. the **number of reductions**, and
3. the **number of polynomials generated**.

Timings

Instead of the timings themselves we present the ratios of the timings comparing the three variants.

Timings

Instead of the timings themselves we present the ratios of the timings comparing the three variants.

system	F5R / F5	F5C / F5R	F5C / F5
Katsura 7	1.13	0.94	1.06
Katsura 8	1.09	0.75	0.83
Katsura 9	1.14	0.54	0.62
Schrans-Troost	1.01	0.70	0.71
Cyclic 6	0.60	1.00	0.60
Cyclic 7	0.80	0.61	0.49
Cyclic 8	0.93	0.66	0.62

Number of reductions

system	# red in F5	# red in F5R	# red in F5C
Katsura 4	774	289	222
Katsura 5	14,597	5,355	3,985
Katsura 6	9,506,808	77,756	58,082
Cyclic 5	512	506	446
Cyclic 6	41,333	23,780	14,167

Number of polynomials generated

In the following we present internal data from the computation of Katsura 9.

i	# G_i in F5	# G_i in F5C	max $\#P$ in F5	max $\#P$ in F5C
2	2	2	none	none
3	4	4	1	1
4	8	8	2	2
5	16	15	4	4
6	32	29	8	6
7	60	51	17	12
8	132	109	29	29
9	524	472	89	71
10	1,165	778	276	89

Conclusions

F5C
is way **faster**,
is **more efficient**,
computes **fewer data**,
computes **fewer reductions**

than F5 and F5R.

References

B. Buchberger.
Ein Algorithmus zum Auffinden der Basiselement des Restklassenrings nach einem nulldimensionalen Polynomideal

J.-C. Faugère.
A new efficient algorithm for computing Gröbner bases without reduction to zero F_5

R. Gebauer and H.M. Möller.
On an Installation of Buchberger's Algorithm

G.-M. Greuel, G. Pfister and H. Schönemann.
SINGULAR 3-1-0. *A computer algebra system for polynomial computations*, TU Kaiserslautern, 2009, `http://www.singular.uni-kl.de`.

T. Stegers.
Faugère's F5 Algorithm Revisited