Faugère’s F5 algorithm: variants and implementation issues

Christian Eder
(joint work with John Perry)

Technische Universität Kaiserslautern

June 24th, 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 variants F5R & F5C which reduce the stated inefficiencies of F5
4. Learning about other improvements due to F5C
5. Comparison of F5, F5R & F5C under several aspects
6. Reducing F4-ish in F5
The following section is about

1. Introducing Gröbner bases
   - Gröbner basics
   - Computation of Gröbner bases
   - Problem of zero reduction

2. The F5 Algorithm

3. Optimizations of F5

4. Further improvements in F5C

5. Comparison of the variants of F5

6. Symbolic preprocessing in F5
Basic problem

1. Given a ring $R$ and an ideal $I \lhd R$ we want to compute a **Gröbner basis** $G$ of $I$.

2. $G$ can be understood as a **nice representation for** $I$. Gröbner bases were discovered by Bruno Buchberger in 1965 [Bu65]. Having computed $G$ lots of **difficult questions** concerning $I$ are **easier to answer using** $G$ instead of $I$.

3. This is due to some nice properties of Gröbner bases. The following is very useful to understand how to compute a Gröbner basis.
Main property of Gőbner bases

Lemma
Let $G$ be a Gröbner basis of an ideal $I$. It holds that for all $p, q \in G$ it holds that

$$\text{Spol}(p, q) \xrightarrow{G} 0,$$

where

- $\text{Spol}(p, q) = \text{hc}(q)u_p p - \text{hc}(p)u_q q$ and
- $u_k = \frac{\text{lcm}(\text{hm}(p), \text{hm}(q))}{\text{hm}(k)}$. 
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 $P := \{\text{Spol}(g_i, g_j) \mid g_i, g_j \in G, i \neq j\}$
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 $P := \{Spol(g_i, g_j) \mid g_i, g_j \in G, i \neq j\}$
4. Choose $p \in P$, $P := P \setminus \{p\}$
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 $P := \{\text{Spol}(g_i, g_j) \mid g_i, g_j \in G, i \neq j\}$
4. Choose $p \in P$, $P := P \setminus \{p\}$
   
   (a) If $p \xrightarrow{G} 0 \Rightarrow \text{no new information}$
   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 $P := \{ \text{Spol}(g_i, g_j) \mid g_i, g_j \in G, i \neq j \}$
4. Choose $p \in P$, $P := P \setminus \{ p \}$
   
   (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 $P = \emptyset$ we are done and $G$ is a Gröbner basis of $I$. 
Computing Gröbner bases incrementally

A slightly variant of this algorithm is the following computing the Gröbner basis \textit{incrementally}:
Computing Gröbner bases incrementally

A slightly variant of this algorithm is the following computing the Gröbner basis **incrementally**:

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

**Output:** Gröbner basis $G$ of $I$
Computing Gröbner bases incrementally

A slightly variant of this algorithm is the following computing the Gröbner basis **incrementally**:

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

**Output:** Gröbner basis $G$ of $I$

1. Compute Gröbner basis $G_1$ of $\langle f_1 \rangle$. 
Computing Gröbner bases incrementally

A slightly variant of this algorithm is the following computing the Gröbner basis \textit{incrementally}:

**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 \( \langle f_1, f_2 \rangle \) by
   (a) adding \( f_2 \) to \( G_1 \), \( G_2 = G_1 \cup \{ f_2 \} \),
   (b) computing S-polynomials of \( f_2 \) with elements of \( G_1 \)
   (c) reducing all S-polynomials and possibly add new elements to \( G_2 \)
Computing Gröbner bases incrementally

A slightly variant of this algorithm is the following computing the Gröbner basis \textbf{incrementally}:

\textbf{Input:} Ideal \( I = \langle f_1, \ldots, f_m \rangle \)

\textbf{Output:} Gröbner basis \( G \) of \( I \)

1. Compute Gröbner basis \( G_1 \) of \( \langle f_1 \rangle \).
2. Compute Gröbner basis \( G_2 \) of \( \langle f_1, f_2 \rangle \) by
   (a) adding \( f_2 \) to \( G_1 \), \( G_2 = G_1 \cup \{f_2\} \),
   (b) computing S-polynomials of \( f_2 \) with elements of \( G_1 \)
   (c) reducing all S-polynomials and possibly add new elements to \( G_2 \)
3. \ldots
4. \( G := G_m \) is the Gröbner basis of \( I \)
Lots of useless computations

It is very time-consuming to compute $G$ such that $\text{Spol}(p, q)$ reduces to zero w.r.t. $G$ for all $p, q \in G$.

When such an $S$-polynomial reduces to an element $h \neq 0$ w.r.t. $G$ then we get **new information** for the structure of $G$, namely adding $h$ to $G$.

But most of the $S$-polynomials considered during the algorithm reduce to zero w.r.t $G$.

$\Rightarrow$ No new information from zero reductions
Problem of zero reduction

Lots of useless computations
It is very time-consuming to compute $G$ such that $\text{Spol}(p, q)$ reduces to zero w.r.t. $G$ for all $p, q \in G$.
When such an S-polynomial reduces to an element $h \neq 0$ w.r.t. $G$ then we get new information for the structure of $G$, namely adding $h$ to $G$.
But most of the S-polynomials considered during the algorithm reduce to zero w.r.t $G$.
$\Rightarrow$ No new information from zero reductions

Problem to be solved
Detect a zero reduction of $\text{Spol}(p, q)$ before we even start to compute the S-polynomial.

Let’s have a look at the following example:
An example of zero reduction

Example
Assume the ideal \( I = \langle g_1, g_2 \rangle \triangleleft \mathbb{Q}[x, y, z] \) where \( g_1 = xy - z^2 \), \( g_2 = y^2 - z^2 \).
An example of zero reduction

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

\[
\text{Spol}(g_2, g_1) = xy^2 - xz^2 - xy^2 + yz^2 = -xz^2 + yz^2,
\]
we get a new element \( g_3 = xz^2 - yz^2 \) for \( G \).
An example of zero reduction

Example

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

Computing

\[
\text{Spol}(g_2, g_1) = xy^2 - xz^2 - xy^2 + yz^2 = -xz^2 + yz^2,
\]

we get a new element \( g_3 = xz^2 - yz^2 \) for \( G \).

Let us compute \( \text{Spol}(g_3, g_1) \) next:
An example of zero reduction

Example

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

Computing

\[
\text{Spol}(g_2, g_1) = xy^2 - xz^2 - xy^2 + yz^2 = -xz^2 + yz^2,
\]

we get a new element \( g_3 = xz^2 - yz^2 \) for \( G \).

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

\[
\text{Spol}(g_3, g_1) = xyz^2 - y^2z^2 - xyz^2 + z^4 = -y^2z^2 + z^4.
\]
An example of zero reduction

Example

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

Computing

\[
\text{Spol}(g_2, g_1) = xy^2 - xz^2 - xy^2 + yz^2 = -xz^2 + yz^2,
\]

we get a new element \( g_3 = xz^2 - yz^2 \) for \( G \).

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

\[
\text{Spol}(g_3, g_1) = xyz^2 - y^2z^2 - xyz^2 + z^4 = -y^2z^2 + z^4.
\]

Now we can reduce further with \( z^2g_2 \):

\[
-y^2z^2 + z^4 + y^2z^2 - z^4 = 0.
\]
How to detect zero reductions in advance?

There are different attempts to detect zero reductions:

1. Buchberger’s criteria and the well-known implementation of Gebauer & Möller [GM88].

2. In 2002 Faugère has published the **F5 Algorithm** [Fa02], a Gröbner basis algorithm which uses new criteria to detect such useless pairs.
How to detect zero reductions in advance?

There are different attempts to detect zero reductions:

1. Buchberger’s criteria and the well-known implementation of Gebauer & Möller [GM88].

2. In 2002 Faugère has published the F5 Algorithm [Fa02], a Gröbner basis algorithm which uses new criteria to detect such useless pairs.

⇒ In the following we need to understand how Faugère’s criteria work.
The following section is about

1. Introducing Gröbner bases

2. The F5 Algorithm
   - F5 basics
   - Implementation of signatures
   - The inefficiency of F5

3. Optimizations of F5

4. Further improvements in F5C

5. Comparison of the variants of F5

6. Symbolic preprocessing in F5
Signatures of polynomials

Faugère’s idea is to give each polynomial during the computations of the algorithm a so-called \textbf{signature}:
Signatures of polynomials

Faugère’s idea is to give each polynomial during the computations of the algorithm a so-called signature:

1. Assuming a polynomial $p$, its signature is defined to be $S(p) = (t, \ell)$ where $t$ is its monomial and $\ell \in \mathbb{N}$ is its index.
2. A generating element $f_i$ of $I$ gets the signature $S(f_i) = (1, i)$.
3. We have an ordering $\prec$ on the signatures:

$$(t_1, \ell_1) \succ (t_2, \ell_2) \iff (a) \ell_1 > \ell_2 \quad \text{or} \quad (b) \ell_1 = \ell_2 \text{ and } t_1 > t_2$$
Signatures of polynomials

Faugère’s idea is to give each polynomial during the computations of the algorithm a so-called signature:

1. Assuming a polynomial $p$ its signature is defined to be $S(p) = (t, \ell)$ where $t$ is its monomial and $\ell \in \mathbb{N}$ is its index.
2. A generating element $f_i$ of $I$ gets the signature $S(f_i) = (1, i)$.
3. We have an ordering $\prec$ on the signatures:

   $$(t_1, \ell_1) \succ (t_2, \ell_2) \iff (a) \ell_1 > \ell_2 \text{ or } (b) \ell_1 = \ell_2 \text{ and } t_1 > t_2$$

Example

Assume $\mathbb{Q}[x, y, z]$ with degree reverse lexicographical ordering. Then

1. $(x^2y, 3) \succ (z^3, 3)$,
2. $(1, 5) \succ (x^{12}y^{234}z^{3456}, 4)$. 
Signatures of polynomials

Remark
Note that there are other ways to define the ordering $\prec$ such that it prefers the degree of the monomial and not the index [MTM92]. Recently Ars and Hashemi have implemented F5 with different orderings [AH09].
Signatures of polynomials

Remark
Note that there are other ways to define the ordering $\prec$ such that it prefers the degree of the monomial and not the index [MTM92]. Recently Ars and Hashemi have implemented F5 with different orderings [AH09].

Using the signatures in the F5 Algorithm we also need to define them for S-polynomials:

$$\text{Spol}(p, q) = \text{hc}(q)u_p p - \text{hc}(p)u_q q \text{ where } S(\text{Spol}(p, q)) = u_p S(p)$$

where we assume that $u_p S(p) \succ u_q S(q)$. 
Example revisited - with signatures

In our example

\[ g_3 = \text{Spol}(g_2, g_1) = xg_2 - yg_1 \]

\[ \Rightarrow S(g_3) = xS(g_2) = x(1, 2) := (x, 2). \]
Example revisited - with signatures

In our example

\[ g_3 = \text{Spol}(g_2, g_1) = xg_2 - yg_1 \]

\[ \Rightarrow S(g_3) = xS(g_2) = x(1, 2) := (x, 2). \]

It follows that \( \text{Spol}(g_3, g_1) = yg_3 - z^2g_1 \) has

\[ S(\text{Spol}(g_3, g_1)) = yS(g_3) = (xy, 2). \]
Example revisited - with signatures

In our example

\[ g_3 = \text{Spol}(g_2, g_1) = xg_2 - yg_1 \]
\[ \Rightarrow S(g_3) = xS(g_2) = x(1, 2) := (x, 2). \]

It follows that \( \text{Spol}(g_3, g_1) = yg_3 - z^2g_1 \) has

\[ S(\text{Spol}(g_3, g_1)) = yS(g_3) = (xy, 2). \]

Note that \( S(\text{Spol}(g_3, g_1)) = (xy, 2) \) and \( \text{hm}(g_1) = xy. \)
Example revisited - with signatures

In our example

\[ g_3 = \text{Spol}(g_2, g_1) = xg_2 - yg_1 \]
\[ \Rightarrow S(g_3) = xS(g_2) = x(1, 2) := (x, 2). \]

It follows that \( \text{Spol}(g_3, g_1) = yg_3 - z^2g_1 \) has

\[ S(\text{Spol}(g_3, g_1)) = yS(g_3) = (xy, 2). \]

Note that \( S(\text{Spol}(g_3, g_1)) = (xy, 2) \) and \( \text{hm}(g_1) = xy. \)
\[ \Rightarrow \text{In F5 we know that } \text{Spol}(g_3, g_1) \text{ will reduce to zero!} \]
How does this work?

Remember that F5 computes a Gröbner basis incrementally.
How does this work?

Theorem (F5 Criterion)

An $S$-polynomial $\text{Spol}(p, q) = \text{hc}(q)u_p p - \text{hc}(p)u_q q$ does not need to be computed, let alone reduced, if $S(p) = (t, \ell)$ and there exists an element $g$ in $G_{\ell - 1}$ such that

$$\text{hm}(g) \mid u_p t.$$ 

A similar statement holds for $S(q)$.

Remember that F5 computes a Gröbner basis incrementally.
How does this work?

**Theorem (F5 Criterion)**

An $S$-polynomial $Spol(p, q) = hc(q)u_pp - hc(p)u_qq$ does not need to be computed, let alone reduced, if $S(p) = (t, \ell)$ and there exists an element $g$ in $G_{\ell-1}$ such that

$$hm(g) | u_pt.$$ 

A similar statement holds for $S(q)$.

**Example**

In our example $g = g_1$ and $u_pt = xy \Rightarrow hm(g_1) = xy | xy$.

Remember that F5 computes a Gröbner basis incrementally.
How does this work?

Theorem (Rewritten Criterion)

An $S$-polynomial $\text{Spol}(p, q) = \text{hc}(q)u_pp - \text{hc}(p)u_qq$ does not need to be computed, let alone reduced, if $S(p) = (t, \ell)$ and there exists an element $g$ with $S(g) = (v, \ell)$ in $G$ which was computed after $p$ and such that

$$v \mid u_pt.$$ 

A similar statement holds for $S(q)$. 

How does this work?

**Theorem (Rewritten Criterion)**

An $S$-polynomial $Spol(p, q) = hc(q)u_p p - hc(p)u_q q$ does not need to be computed, let alone reduced, if $S(p) = (t, \ell)$ and there exists an element $g$ with $S(g) = (v, \ell)$ in $G$ which was computed after $p$ and such that

$$v \mid u_pt.$$ 

A similar statement holds for $S(q)$.

**Remark**

OK, and now forget about all this stuff.

Faugère’s criteria are based on the signatures.
Idea behind the signatures

The main idea is to have

- **small data** added to polynomials, and
- **strong criteria** detecting useless S-polynomials based on this data.
Idea behind the signatures

The main idea is to have

- **small data** added to polynomials, and
- **strong criteria** detecting useless S-polynomials based on this data.

Remark

\[ \text{signature} \leftrightarrow \text{monomial plus an integer} \]
Remark
Monomials are terms without coefficients.

\[
\text{monomial} \leftrightarrow \text{integer vector}
\]
Remark
Monomials are terms without coefficients.

\[
\text{monomial} \leftrightarrow \text{integer vector}
\]

Example
Assume the ring $\mathbb{Q}[x, y, z]$ in the 3 variables $x, y, z$.

$xy^3z^2 \Rightarrow (1, 3, 2)$

Note that the length of the integer vector equals the number of variables of the ring.
Implementation of signatures

The data structure of a signature follows easily:

```
integer vector for the monomial of the signature
+
integer for the index of the signature
```
Implementation of signatures

The data structure of a signature follows easily:

- integer vector for the monomial of the signature
- integer for the index of the signature

Example

\[ S(g) = (xy^3z^2, 7) \Rightarrow (1, 3, 2, 7). \]
Implementation of signatures

The data structure of a signature follows easily:

- integer vector for the monomial of the signature
  + integer for the index of the signature

Example

\[ S(g) = (xy^3z^2, 7) \Rightarrow (1, 3, 2, 7). \]

⇒ signature ↔ integer vector with length \#\text{var}+1
Difficulty 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.
Difficulty 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.

Remark
We will see in the following example that we do not only need to be careful if we are allowed to reduce an element, but also must be able to generate new polynomials during reduction when reducing with elements generated in the current iteration step.
Difficulty 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.

Remark
We will see in the following example that we do not only need to be careful if we are allowed to reduce an element, but also must be able to generate new polynomials during reduction when reducing with elements generated in the current iteration step.

Example
Assume the polynomial $p = xy^2 - z^3$ with $S(p) = (t_p, \ell)$ and a possible reducer $q = y^2 - xz$ with $S(q) = (t_q, \ell)$. 
Difficulty 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.

**Remark**
We will see in the following example that we do not only need to be careful if we are allowed to reduce an element, but also must be able to generate new polynomials during reduction when reducing with elements generated in the current iteration step.

**Example**
Assume the polynomial $p = xy^2 - z^3$ with $S(p) = (t_p, \ell)$ and a possible reducer $q = y^2 - xz$ with $S(q) = (t_q, \ell)$. In Buchberger-like implementations the top-reduction would take place, i.e. we would compute $p - xq$. 
Example
In F5 the following can happen:

1. If $xq$ satisfies the F5 Criterion $\Rightarrow$ no reduction!
Difficulty of top-reduction in F5

Example
In F5 the following can happen:

1. If $xq$ satisfies the F5 Criterion $\Rightarrow$ no reduction!
2. If $xq$ satisfies the Rewritten Criterion $\Rightarrow$ no reduction!
Difficulty of top-reduction in F5

Example
In F5 the following can happen:

1. If \( xq \) satisfies the F5 Criterion \( \Rightarrow \) no reduction!
2. If \( xq \) satisfies the Rewritten Criterion \( \Rightarrow \) no reduction!
3. None of the above cases holds and \( xS(q) \prec S(p) \Rightarrow p - xq \) is computed and gets the signature \( S(p) \).
Example

In F5 the following can happen:

1. If $xq$ satisfies the F5 Criterion $\Rightarrow \text{no reduction!}$
2. If $xq$ satisfies the Rewritten Criterion $\Rightarrow \text{no reduction!}$
3. None of the above cases holds and $xS(q) \prec S(p) \Rightarrow p - xq$ is computed and gets the signature $S(p)$.
4. None of the first two cases holds and $xS(q) \succ S(p) \Rightarrow$, which leads to
Difficulty of top-reduction in F5

Example

In F5 the following can happen:

1. If $xq$ satisfies the F5 Criterion $\Rightarrow$ no reduction!
2. If $xq$ satisfies the Rewritten Criterion $\Rightarrow$ no reduction!
3. None of the above cases holds and $xS(q) \prec S(p) \Rightarrow p - xq$ is computed and gets the signature $S(p)$.
4. None of the first two cases holds and $xS(q) \succ S(p) \Rightarrow$, which leads to
   (a) No reduction of $p$, but searching for another possible reducer of it.
Difficulty of top-reduction in F5

Example

In F5 the following can happen:

1. If $xq$ satisfies the F5 Criterion ⇒ no reduction!
2. If $xq$ satisfies the Rewritten Criterion ⇒ no reduction!
3. None of the above cases holds and $xS(q) \prec S(p) \Rightarrow p - xq$ is computed and gets the signature $S(p)$.
4. None of the first two cases holds and $xS(q) \succ S(p) \Rightarrow$, which leads to
   (a) No reduction of $p$, but searching for another possible reducer of it.
   (b) a new S-polynomial $r := xq - p$ whereas $S(r) = xS(q)$. 
Remark
Note the following important details:

• If we reduce with elements which signatures have lower index than the current index, we do not check for any criterion. Moreover due to the definition of $\prec$ we do not need to compare the signatures.

• F5 only performs top-reductions, so no interreductions are done.

• Due to the last case of the previous example it is possible that the top-reduction procedure returns two polynomials, i.e. the number of elements to be reduced increases!
Redundant polynomials

Example
Assuming the first two cases of the previous example and moreover that there exists no other top-reducer of $p$ we would end up with both, $p$ and $q$ being in $G$ whereas clearly $\text{hm}(q) \mid \text{hm}(p)$. Thus $p$ is **redundant** for $G$. 
Redundant polynomials

Example
Assuming the first two cases of the previous example and moreover that there exists no other top-reducer of \( p \) we would end up with both, \( p \) and \( q \) being in \( G \) whereas clearly \( \text{hm}(q) \mid \text{hm}(p) \).
Thus \( p \) is redundant for \( G \).

But…
For the F5 Algorithm itself and the criteria based on the signatures \( p \) could be necessary in this iteration step!
⇒ Disrespecting the way F5 top-reduces polynomials would harm the correctness of F5 in this iteration step!
Points of inefficiency

The difficulty of top-reduction in F5 leads to an inefficiency, namely we have way too many polynomials in the intermediate $G_i$s which are possible reducers, i.e. more checks for divisibility and the criteria have to be done, and

1. with which we compute new S-polynomials, i.e. more (for the resulting Gröbner basis redundant) data is generated.
Points of inefficiency

The difficulty of top-reduction in F5 leads to an inefficiency, namely we have way too many polynomials in the intermediate $G_i$'s which are possible reducers, i.e. more checks for divisibility and the criteria have to be done, and

1. with which we compute new S-polynomials, i.e. 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. Further improvements in F5C
5. Comparison of the variants of F5
6. Symbolic preprocessing in F5
F5R: reduced GB reduction

An idea how to fix the first inefficiency, was given by Till Stegers in 2005. His slight optimization of 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 slight optimization of 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 slight optimization of 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.
Question

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

Answer

Interreducing $G_i$ to $B_i \leftrightarrow$ reduction steps rejected by F5
Why is $B_i$ only used for reduction purposes, but not for new-pair computations?

Interreducing $G_i$ to $B_i \iff$ reduction steps rejected by F5

⇒ 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.  
⇒ 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.
⇒ 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.
⇒ 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 $S(g_k) = (1, k)$.
4. For all elements $g_j, g_k \in B_i$ recompute signatures for $\text{Spol}(g_j, g_k)$.
5. Start the next iteration step with $f_{i+1}$ by computing all pairs with elements from $B_i$. 
Recomputation of signatures?

Why do we recompute the signatures of the S-polynomials in $B_i$?

1. Both criteria are based on signatures.
2. More signatures $\Rightarrow$ possibly more rejections of useless elements.
3. Also a **zero polynomial** should have a signature.
Recomputation of signatures?

Why do we recompute the signatures of the S-polynomials in $B_i$?

1. Both criteria are based on signatures.
2. More signatures $\Rightarrow$ possibly more rejections of useless elements.
3. Also a zero polynomial should have a signature.

Question
Do we really need them?
Recomputation of signatures?

Why do we recompute the signatures of the S-polynomials in $B_i$?

1. Both criteria are based on signatures.
2. More signatures $\Rightarrow$ possibly more rejections of useless elements.
3. Also a zero polynomial should have a signature.

Question
Do we really need them?

Answer
Not in F5C :)

The following section is about

1. Introducing Gröbner bases
2. The F5 Algorithm
3. Optimizations of F5
4. Further improvements in F5C
   - Simplified signatures
   - Avoiding recomputations of signatures
   - Fewer criteria checks
   - Implementation of signature revisited
5. Comparison of the variants of F5
6. Symbolic preprocessing in F5
Simplified signatures

The implementation of F5C has some nice improvements for the usage of the criteria. These are based on the following fact:

Each element $g_k$ in $B_i$ has the signature $(1, k)$. 
Simplified signatures

The implementation of F5C has some nice improvements for the usage of the criteria. These are based on the following fact:

Each element $g_k$ in $B_i$ has the signature $(1, k)$.

When generating $\text{Spol}(g_j, g_k)$ during the computations of $G_{i+1}$ we get

$$\text{Spol}(g_j, g_k) = \text{hc}(g_k)u_jg_j - \text{hc}(g_j)u_kg_k.$$
The implementation of F5C has some nice improvements for the usage of the criteria. These are based on the following fact:

Each element \( g_k \) in \( B_i \) has the signature \((1, k)\).

When generating \( \text{Spol}(g_j, g_k) \) during the computations of \( G_{i+1} \) we get

\[
\text{Spol}(g_j, g_k) = \text{hc}(g_k) u_j g_j - \text{hc}(g_j) u_k g_k.
\]

Closer look at the signatures:

\[
u_k S(g_k) = u_k (1, k) = (u_k, k).
\]
Re-doing stuff is never nice

Recomputing the signatures of the S-polynomials in $B_i$ is the only part of F5C 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 F5C which seems to be annoying.

Further improvement

In 2009 Perry & Eder have shown that:

Theorem

In F5C there is no need to recompute the signatures of the S-polynomials of elements of the previous iteration step.
Thus we have to do the following after each iteration of F5:

1. Delete all signatures.
2. Interreduce $G_i$ to $B_i$.
3. For each $g_k \in B_i$ set $S(g_k) = (1, k)$.
4. Start the next iteration step with $f_{i+1}$.
Re-doing stuff is never nice

Thus we have to do the following after each iteration of F5:

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

Remark
Note that this also leads to fewer criteria checks.
Differences using F5 Criterion

**Faugère:** F5 Criterion only for polynomials computed in current iteration step

**Stegers:** F5 Criterion for all polynomials, also those computed in the previous iteration steps
**Differences using F5 Criterion**

**Faugère:** F5 Criterion only for polynomials computed in current iteration step

**Stegers:** F5 Criterion for all polynomials, also those computed in the previous iteration steps

Clearly Faugère’s attempt performs **fewer checks** than Stegers’. But possibly Stegers’ attempt **rejects more elements**.
Differences using F5 Criterion

Faugère: F5 Criterion only for polynomials computed in current iteration step

Stegers: F5 Criterion for all polynomials, also those computed in the previous iteration steps

Clearly Faugère’s attempt performs fewer checks than Stegers’. But possibly Stegers’ attempt rejects more elements.

Using F5C we have the following wonderful position:

Faugère’s way $\Rightarrow$ Stegers’ way
Which elements are even checked now?

1. Polynomials computed in the current iteration step are checked by both criteria.
2. Polynomials computed in previous iteration steps are **not** checked at all.
Which elements are even checked now?

1. Polynomials computed in the current iteration step are checked by both criteria.
2. Polynomials computed in previous iteration steps are **not checked at all**.

**Benefits**

1. No need to distinguish the signatures by their index anymore.
2. Less criteria checks.
3. Signatures are just monomials added to polynomials in the current iteration step.
Which elements are even checked now?

1. Polynomials computed in the current iteration step are checked by both criteria.
2. Polynomials computed in previous iteration steps are **not checked at all**.

Benefits

1. No need to distinguish the signatures by their index anymore.
2. Less criteria checks.
3. Signatures are just monomials added to polynomials in the current iteration step.

⇒ signature ↔ integer vector with length $\#\text{var}$
The following section is about

1. Introducing Gröbner bases
2. The F5 Algorithm
3. Optimizations of F5
4. Further improvements in F5C
5. Comparison of the variants of F5
   - Implementations
   - Comparison of the variants
   - Comparison of F5, F5R & F5C
6. Symbolic preprocessing in F5
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.
Preliminaries

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

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

SINGULAR 3.1.0, kernel implementation; Linux-gentoo-r8 2009 x86_64, Intel Xeon © 3.16 GHz, 64 GB RAM
## Number of reductions

<table>
<thead>
<tr>
<th>system</th>
<th># red in F5</th>
<th># red in F5R</th>
<th># red in F5C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Katsura 4</td>
<td>774</td>
<td>289</td>
<td>222</td>
</tr>
<tr>
<td>Katsura 5</td>
<td>14,597</td>
<td>5,355</td>
<td>3,985</td>
</tr>
<tr>
<td>Katsura 6</td>
<td>1,029,614</td>
<td>77,756</td>
<td>58,082</td>
</tr>
<tr>
<td>Cyclic 5</td>
<td>512</td>
<td>506</td>
<td>446</td>
</tr>
<tr>
<td>Cyclic 6</td>
<td>41,333</td>
<td>23,780</td>
<td>14,167</td>
</tr>
</tbody>
</table>

Sage 3.2.1, Python implementation; Ubuntu Linux 8.10, Intel Core 2 Quad @ 2.66 GHz, 3 GB RAM
Number of polynomials generated

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

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

<table>
<thead>
<tr>
<th>i</th>
<th># $G_i$ in F5</th>
<th># $G_i$ in F5C</th>
<th>max # $P$ in F5</th>
<th>max # $P$ in F5C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>15</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>29</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>60</td>
<td>51</td>
<td>17</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>132</td>
<td>109</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>9</td>
<td>524</td>
<td>472</td>
<td>89</td>
<td>71</td>
</tr>
<tr>
<td>10</td>
<td>1,165</td>
<td>778</td>
<td>276</td>
<td>89</td>
</tr>
</tbody>
</table>

Sage 3.2.1, Python implementation; Ubuntu Linux 8.10, Intel Core 2 Quad @ 2.66 GHz, 3 GB RAM
The following section is about

1. Introducing Gröbner bases
2. The F5 Algorithm
3. Optimizations of F5
4. Further improvements in F5C
5. Comparison of the variants of F5
6. Symbolic preprocessing in F5
   F5’s reduction revisited
F5’s reduction revisited

F5’s reduction with current iteration polynomials:

1. Only top-reductions
2. Some top-reductions rejected
3. Possibly new polynomials generated
F5’s reduction revisited

F5’s reduction with current iteration polynomials:

1. Only top-reductions
2. Some top-reductions rejected
3. Possibly new polynomials generated

Remark
The 2nd and 3rd property of reduction cannot be changed due to the signatures. But the 1st property can be changed!
F5’s reduction revisited

F5’s reduction with current iteration polynomials:

1. Only top-reductions
2. Some top-reductions rejected
3. Possibly new polynomials generated

Remark
The 2nd and 3rd property of reduction cannot be changed due to the signatures. But the 1st property can be changed!

reduce polynomials “F5-completely” ⇒ sparser polynomials
sparser polynomials ⇒ faster reduction in higher degree
F5’s reduction revisited

F5’s reduction with current iteration polynomials:

1. Only top-reductions
2. Some top-reductions rejected
3. Possibly new polynomials generated

Remark
The 2nd and 3rd property of reduction cannot be changed due to the signatures. But the 1st property can be changed!

reduce polynomials “F5-completely” ⇒ sparser polynomials
sparser polynomials ⇒ faster reduction in higher degree

Question
What is an F5-complete reduction?
F5-complete reduction

Let’s try some F4-ish **symbolic preprocessing**. Assume the element \( p \) to be reduced in F5:

1. Set \( \mathcal{M} := \{ \text{monomials of } p \} \), \( G := \emptyset \), \( B := \emptyset \).

2. Choose the greatest monomial \( m \) w.r.t. \(<\) from \( \mathcal{M} \) and set \( \mathcal{M} = \mathcal{M} \setminus \{ m \} \).

3. Check for reducers of \( m \).
Let’s try some F4-ish symbolic preprocessing. Assume the element $p$ to be reduced in F5:

1. Set $\mathcal{M} := \{\text{monomials of } p\}$, $\mathcal{G} := \emptyset$, $\mathcal{B} := \emptyset$.

2. Choose the greatest monomial $m$ w.r.t. $<$ from $\mathcal{M}$ and set $\mathcal{M} = \mathcal{M} \setminus \{m\}$.

3. Check for reducers of $m$.

4. Reducer $q \Rightarrow$ Generate pair $(u, q)$ where $uhm(q) = m$. 
F5-complete reduction

Let’s try some F4-ish symbolic preprocessing. Assume the element $p$ to be reduced in F5:

1. Set $\mathcal{M} := \{\text{monomials of } p\}$, $\mathcal{G} := \emptyset$, $\mathcal{B} := \emptyset$.

2. Choose the greatest monomial $m$ w.r.t. $<$ from $\mathcal{M}$ and set $\mathcal{M} = \mathcal{M} \setminus \{m\}$.

3. Check for reducers of $m$.

4. Reducer $q \Rightarrow$ Generate pair $(u, q)$ where $uhm(q) = m$.
   
   (a) If $uS(q) \succ S(p) \Rightarrow \mathcal{B} = \mathcal{B} \cup \{q\}$.
   
   (b) If $uS(q) \prec S(p) \Rightarrow \mathcal{G} = \mathcal{G} \cup \{(u, q)\}$,
       $\mathcal{M} = \mathcal{M} \cup \{\text{monomials of } u(q - hm(q))\}$.
F5-complete reduction

Let’s try some F4-ish symbolic preprocessing.
Assume the element \( p \) to be reduced in F5:

1. Set \( M := \{ \text{monomials of } p \} \), \( G := \emptyset \), \( B := \emptyset \).
2. Choose the greatest monomial \( m \) w.r.t. \( < \) from \( M \) and set \( M = M \setminus \{ m \} \).
3. Check for reducers of \( m \).
4. Reducer \( q \) ⇒ Generate pair \( (u, q) \) where \( u \text{hm}(q) = m \).
   (a) If \( uS(q) \triangleright S(p) \Rightarrow B = B \cup \{ q \} \).
   (b) If \( uS(q) \triangleleft S(p) \Rightarrow G = G \cup \{ (u, q) \} \), \( M = M \cup \{ \text{monomials of } u(q - \text{hm}(q)) \} \).
5. When \( M = \emptyset \)
   (a) Reduce \( p \) with all generated polynomials \( uq \) of \( G \).
   (b) Check again if \( \text{hm}(q) \mid \text{hm}(p) \) for any \( q \in B \).
      If so: New S-polynomial \( r = vq - p \) with \( S(r) = vS(q) \) where \( v\text{hm}(q) = \text{hm}(p) \).
Some last remarks

1. Note that the elements in $\mathcal{G}$ are the non-signature corrupting reducers.
Some last remarks

1. Note that the elements in $\mathcal{G}$ are the non-signature corrupting reducers.

2. Note also that when computing $vq$ in the last step of the reduction it is clear that $vS(q) \succ S(p)$. 
Some last remarks

1. Note that the elements in $G$ are the non-signature corrupting reducers.

2. Note also that when computing $vq$ in the last step of the reduction it is clear that $vS(q) \succ S(p)$.

3. Assume a given reduction procedure $\mathcal{R}$ reducing one element w.r.t to an ideal. Then it is highly problematic to implement the stated reduction:
Some last remarks

1. Note that the elements in $G$ are the non-signature corrupting reducers.

2. Note also that when computing $vq$ in the last step of the reduction it is clear that $vS(q) \succ S(p)$.

3. Assume a given reduction procedure $\mathcal{R}$ reducing one element w.r.t to an ideal. Then it is highly problematic to implement the stated reduction:
   (a) The ideal $I$ must be homogeneous.
Some last remarks

1. Note that the elements in $G$ are the non-signature corrupting reducers.

2. Note also that when computing $vq$ in the last step of the reduction it is clear that $vS(q) \succ S(p)$.

3. Assume a given reduction procedure $\mathcal{R}$ reducing one element w.r.t to an ideal. Then it is highly problematic to implement the stated reduction:
   - (a) The ideal $I$ must be homogeneous.
   - (b) The multiple $uq$ must be computed and given to $\mathcal{R}$. 
Some last remarks

1. Note that the elements in $G$ are the non-signature corrupting reducers.

2. Note also that when computing $vq$ in the last step of the reduction it is clear that $vS(q) \succ S(p)$.

3. Assume a given reduction procedure $\mathcal{R}$ reducing one element w.r.t to an ideal. Then it is highly problematic to implement the stated reduction:
   - (a) The ideal $I$ must be homogeneous.
   - (b) The multiple $uq$ must be computed and given to $\mathcal{R}$.

Remark

Without these constraints signature corrupting reductions can happen: An element $q \in G$ can be a “good” reducer and a “bad” reducer at the same time.
References

G. Ars and A. Hashemi.
Extended F5 Criteria

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


H. M. Möller, T. Mora and C. Traverso.
Gröbner bases computation using syzygies

W. A. Stein et al.

T. Stegers.
Faugère’s F5 Algorithm Revisited