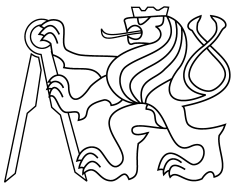
CENTER FOR
MACHINE PERCEPTION



CZECH TECHNICAL
UNIVERSITY IN PRAGUE

BACHELOR THESIS

Minimal Problem Solver Generator

Pavel Trutman

pavel.trutman@fel.cvut.cz

April 29, 2015

Available at
http://cmp.felk.cvut.cz/~trutmpav/theses/bsc-pavel-trutman.pdf

Thesis Advisor: Ing. Tomáš Pajdla, PhD.

Acknowledge grants here. Use centering if the text is too short.

Center for Machine Perception, Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University
Technická 2, 166 27 Prague 6, Czech Republic
fax +420 2 2435 7385, phone +420 2 2435 7637, www: http://cmp.felk.cvut.cz

Minimal Problem Solver Generator

Pavel Trutman

April 29, 2015

Text of acknowledgements...

Abstract

Text of abstract...

Resumé

Text of resumé...

Contents

1 Introduction **4**

2 Polynomial system solving **5**
- 2.1 Buchberger Algorithm . . . 5
  - 2.1.1 First implementation . . . 5
  - 2.1.2 Improved Buchberger Algorithm . . . 6
- 2.2 F_4 Algorithm . . . 7
  - 2.2.1 Improved Algorithm F_4 . . . 7
  - 2.2.2 Function Update . . . 10
  - 2.2.3 Function Reduction . . . 10
  - 2.2.4 Function Symbolic Preprocessing . . . 10
  - 2.2.5 Function Simplify . . . 11
  - 2.2.6 Selection strategy . . . 11
- 2.3 F_5 Algorithm . . . 12

3 Automatic generator **14**
- 3.1 Reimplementation . . . 14
- 3.2 Multiple eliminations solver . . . 14
- 3.3 Removing unnecessary polynomials . . . 14
- 3.4 Matrix partitioning . . . 14
- 3.5 F4 strategy . . . 14

4 Experiments **15**

5 Conclusion **16**

Bibliography **17**

List of Algorithms

1 Simple Buchberger Algorithm 5
2 Improved Buchberger Algorithm 6
3 Update 8
4 Improved Algorithm F_4 9
5 Reduction 10
6 Symbolic Preprocessing 11
7 Simplify 12
8 Sel – The normal strategy for F_4 12

List of Abbreviations

C(f), C(F)	Set of all coefficients of the polynomial f or of all polynomials from the set F.
$\overline{f}^F$	Remainder of the polynomial f on division by F.
gcd	Greatest common multiple.
lcm	Least common divisor.
LC(f), LC(F)	Leading coefficient(s) of the polynomial f or of all polynomials from the set F.
LM(f), LM(F)	Leading monomial(s) of the polynomial f or of all polynomials from the set F.
LT(f), LT(F)	Leading term(s) of the polynomial f or of all polynomials from the set F.
M(f), M(F)	Set of all monomials of the polynomial f or of all polynomials from the set F.
$S(f_1, f_2)$	S-polynomial of polynomials f_1 and f_2.
T(f), T(F)	Set of all terms of the polynomial f or of all polynomials from the set F.
$x \mid y$	x divides y.

1 Introduction

Here comes introduction.

2 Polynomial system solving

Firstly we review the state of the art algorithms for computing Gröbner basis. Better understanding of these algorithms helps us to integrate them into polynomial solving algorithms based on Gröbner basis computation more efficiently.

2.1 Buchberger Algorithm

Buchberger Algorithm [2], which was invented by Bruno Buchberger, was the first algorithm for computing Gröbner basis. The algorithm is described in details in [1, 3].

2.1.1 First implementation

The first and easy, but very inefficient implementation of the Buchberger Algorithm, Algorithm 1, is based on the observation that we can extend a set F of polynomials to a Gröbner basis only by adding all non-zero remainders $\overline{S(f_i, f_j)}^F$ of all pairs from F into F until there is no non-zero remainder generated.

The main disadvantage of this simple algorithm is that so constructed Gröbner basis are often bigger than necessary. This implementation of the algorithm is also very inefficient because many of the S-polynomials that are constructed from the critical pairs are reduced to zero so after spending effort on computing them, there is nothing to add to the Gröbner basis G. How to decide which pairs need not be generated is described next.

Algorithm 1 Simple Buchberger Algorithm

Input:
F a finite set of polynomials
Output:
G a finite set of polynomials

1: $G \leftarrow F$
2: $B \leftarrow \{\{g_1, g_2\} \mid g_1, g_2 \in G,\ g_1 \neq g_2\}$
3: **while** $B \neq \emptyset$ **do**
4: select $\{g_1, g_2\}$ from B
5: $B \leftarrow B \setminus \{\{g_1, g_2\}\}$
6: $h \leftarrow S(g_1, g_2)$
7: $h_0 \leftarrow \overline{h}^G$
8: **if** $h_0 \neq 0$ **then**
9: $B \leftarrow B \cup \{\{g, h_0\} \mid g \in G\}$
10: $G \leftarrow G \cup \{h_0\}$
11: **end if**
12: **end while**
13: **return** G

2.1.2 Improved Buchberger Algorithm

The combinatorial complexity of the simple implementation of the Buchberger Algorithm can be reduced by testing out certain S-polynomials which need not be considered. To know which pairs can be deleted without treatment, we use the first and the second Buchberger's criterion [1]. Sometimes, we can even delete certain polynomials from the set G completely, knowing that every critical pair they will generate will reduce to zero and hence these polynomials themselves will be superfluous in the output set. In the next few paragraphs we will describe the implementation of the Improved Buchberger Algorithm and of the function *Update*, which deletes superfluous polynomials from G, according to Gebauer and Möller [7].

The Improved Buchberger Algorithm, Algorithm 2, has the same structure as the Simple Algorithm. The function *Update* is used at the beginning of the Improved Buchberger Algorithm to initialize the set B of critical pairs and the Gröbner basis G from the input set F of polynomials and at every moment when a new non-zero polynomial $h_0 = \overline{h}^G$ of an S-polynomial h has been found and the sets B and G are about to be updated.

Algorithm 2 Improved Buchberger Algorithm

Input:
F a finite set of polynomials
Output:
G a finite set of polynomials

1: $G \leftarrow \emptyset$
2: $B \leftarrow \emptyset$
3: **while** $F \neq \emptyset$ **do**
4: select f from F
5: $F \leftarrow F \backslash \{f\}$
6: $(G, B) \leftarrow Update(G, B, f)$
7: **end while**
8: **while** $B \neq \emptyset$ **do**
9: select $\{g_1, g_2\}$ from B
10: $B \leftarrow B \backslash \{\{g_1, g_2\}\}$
11: $h \leftarrow S(g_1, g_2)$
12: $h_0 \leftarrow \overline{h}^G$
13: **if** $h_0 \neq 0$ **then**
14: $(G, B) \leftarrow Update(G, B, h_0)$
15: **end if**
16: **end while**
17: **return** G

Now, let us look at the function *Update*, Algorithm 3. First, it makes pairs from the new polynomial h and all polynomials from the set G_{old} and puts them into the set C. The first while loop (lines 3 – 9) iterates over all pairs in the set C. In each iteration it select a pair $\{h, g_1\}$ from the set C and removes it from the set. Then it looks for another pair $\{h, g_2\}$ from the set C or the set D. If does not exists a pair $\{h, g_2\}$ such that (h, g_2, g_1) is a Buchberger triple, then the pair $\{h, g_1\}$ is put into the set D. The triple (h, g_2, g_1) of polynomials h, g_1 and g_2 is a Buchberger triple if the equivalent

conditions

$$\mathrm{LM}(g_2) \mid \mathrm{lcm}(\mathrm{LM}(h), \mathrm{LM}(g_1)) \tag{2.1}$$
$$\mathrm{lcm}(\mathrm{LM}(h), \mathrm{LM}(g_2)) \mid \mathrm{lcm}(\mathrm{LM}(h), \mathrm{LM}(g_1)) \tag{2.2}$$
$$\mathrm{lcm}(\mathrm{LM}(g_2), \mathrm{LM}(g_1)) \mid \mathrm{lcm}(\mathrm{LM}(h), \mathrm{LM}(g_1)) \tag{2.3}$$

are satisfied. From the second Buchberger's criterion, we know that if a Buchberger triple (h, g_2, g_1) shows up in the Buchberger Algorithm and the pairs $\{g_1, g_2\}$ and $\{h, g_2\}$ are amongs the critical pairs, then the pair $\{h, g_1\}$ need not be generated. That means in the code that such a pair is not moved from the set C to the set D but it is only removed from the set C. This while loop keeps all pairs $\{h, g_1\}$ where $\mathrm{LM}(h)$ and $\mathrm{LM}(g_1)$ are disjoint, i.e. $\mathrm{LM}(h)$ and $\mathrm{LM}(g_1)$ have no variable in common. The reason of this is that if two or more pairs in C have the same lcm of their leading monomials, then there is a choice which one should be deleted. So we keep the pair where the leading monomials are disjoint. Pairs with disjoint leading monomials are removed in the second while loop, so we eventually remove them all.

The second while loop (lines 11 – 17) eliminates all pairs with disjoint leading monomials. We can remove such pairs thanks to the first Buchberger's criterion. All remaining pairs are stored in the set E.

The third while loop (lines 19 – 25) eliminates pairs $\{g_1, g_2\}$ where (g_1, h, g_2) is a Buchberger triple from the set B_{old}. Then the updated set of the old pairs and the new pairs are united into the set B_{new}.

Finally, the last while loop (lines 28 – 34) removes all polynomials g whose leading monomial is a multiple of the leading monomial of h from the set G_{old}. We can eliminate such polynomials for two reasons. Firstly, $\mathrm{LM}(h) \mid \mathrm{LM}(g)$ implies $\mathrm{LM}(h) \mid \mathrm{lcm}(\mathrm{LM}(g), \mathrm{LM}(f))$ for arbitrary polynomial f. We can see that (g, h, f) is a Buchberger triple for any f which in future appears in the set G. Moreover, polynomial g will not be missed at the end, because in the Gröbner basis G, polynomials with leading monomials which are multiples of leading monomials of another polynomial from G are superfluous, i.e. they will be eliminated in the reduced Gröbner basis.

In the end of the function, the polynomial h is added into the Gröbner basis G_{new}. The output of the function *Update* is the Gröbner basis G_{new} and the set B_{new} of critical pairs.

2.2 F_4 Algorithm

The F_4 Algorithm [5] by Jean-Charles Faugère is an improved version of the Buchberger's Algorithm. The F_4 replaces the classical polynomial reduction found in the Buchberger's Algorithm by a simultaneous reduction of several polynomials. This reduction mechanism is achieved by a symbolic precomputation followed by Gaussian elimination implemented using sparse linear algebra methods. F_4 speeds up the reduction step by exchanging multiple polynomial divisions for row-reduction of a single matrix.

2.2.1 Improved Algorithm F_4

The main function of the F_4 Algorithm, Algorithm 4, consists of two parts. The goal of the first part is to initialize the whole algorithm.

Algorithm 3 Update

Input:
G_{old} a finite set of polynomials
B_{old} a finite set of pairs of polynomials
h a polynomial such that $h \neq 0$

Output:
G_{new} a finite set of polynomials
B_{new} a finite set of pairs of polynomials

1: $C \leftarrow \{\{h, g\} \mid g \in G_{old}\}$
2: $D \leftarrow \emptyset$
3: **while** $C \neq \emptyset$ **do**
4: select $\{h, g_1\}$ from C
5: $C \leftarrow C \backslash \{\{h, g_1\}\}$
6: **if** LM(h) and LM(g_1) are disjoint **or**
(lcm(LM(h), LM(g_2)) $\nmid$ lcm(LM(h), LM(g_1)) for all $\{h, g_2\} \in C$ **and**
lcm(LM(h), LM(g_2)) $\nmid$ lcm(LM(h), LM(g_1)) for all $\{h, g_2\} \in D$) **then**
7: $D \leftarrow D \cup \{\{h, g_1\}\}$
8: **end if**
9: **end while**
10: $E \leftarrow \emptyset$
11: **while** $D \neq \emptyset$ **do**
12: select $\{h, g\}$ from D
13: $D \leftarrow D \backslash \{\{h, g\}\}$
14: **if** LM(h) and LM(g) are not disjoint **then**
15: $E \leftarrow E \cup \{\{h, g\}\}$
16: **end if**
17: **end while**
18: $B_{new} \leftarrow \emptyset$
19: **while** $B_{old} \neq \emptyset$ **do**
20: select $\{g_1, g_2\}$ from B_{old}
21: $B_{old} \leftarrow B_{old} \backslash \{\{g_1, g_2\}\}$
22: **if** LM(h) $\nmid$ lcm(LM(g_1), LM(g_2)) **or**
lcm(LM(g_1), LM(h)) = lcm(LM(g_1), LM(g_2)) **or**
lcm(LM(h), LM(g_2)) = lcm(LM(g_1), LM(g_2)) **then**
23: $B_{new} \leftarrow B_{new} \cup \{\{g_1, g_2\}\}$
24: **end if**
25: **end while**
26: $B_{new} \leftarrow B_{new} \cup E$
27: $G_{new} \leftarrow \emptyset$
28: **while** $G_{old} \neq \emptyset$ **do**
29: select g from G_{old}
30: $G_{old} \leftarrow G_{old} \backslash \{g\}$
31: **if** LM(h) $\nmid$ LM(g) **then**
32: $G_{new} \leftarrow G_{new} \cup \{g\}$
33: **end if**
34: **end while**
35: $G_{new} \leftarrow G_{new} \cup \{h\}$
36: **return** (G_{new}, B_{new})

First, it generates the set P of critical pairs and initializes the Gröbner basis G. This is done by taking each polynomial from the input set F and calling the function *Update* on it, which updates the set P of pairs and the set G of basic polynomials.

The second part of the algorithm generates new polynomials and adds them into the set G. In each iteration, it selects some pairs from P using the function *Sel*. Many selection strategies are possible and is still an open question how to best select the pairs. Some selection strategies are described in the section 2.2.6 on page 11. Then, it splits each selected pair $\{f_1, f_2\}$ into two tuples. The first tuple contains the first polynomial f_1 of the pair and the monomial m_1 such that $\mathrm{LM}(m_1 \times f_1) = \mathrm{lcm}(\mathrm{LM}(f_1), \mathrm{LM}(f_2))$. The second tuple is constructed in the same way from the second polynomial f_2 of the pair. All tuples from all selected pairs are put into the set L, i.e. duplicates are removed.

Next, function *Reduction* is called on the set L. It stores result in the set $\tilde{F}^+$. In the end of the algorithm it iterates through all new polynomials in the set $\tilde{F}^+$ and calls the function *Update* on each of them. This generates new pairs into the set P of critical pairs and extends the Gröbner basis G.

This algorithm terminates when the set P of pairs is empty. Then the set G is a Gröbner basis and it is the output of the algorithm.

Algorithm 4 Improved Algorithm F_4

Input:
F a finite set of polynomials
Sel a function $List(Pairs) \to List(Pairs)$ such that $Sel(l) \neq \emptyset$ if $l \neq \emptyset$

Output:
G a finite set of polynomials

1: $G \leftarrow \emptyset$
2: $P \leftarrow \emptyset$
3: $d \leftarrow 0$
4: **while** $F \neq \emptyset$ **do**
5: select f form F
6: $F \leftarrow F \backslash \{f\}$
7: $(G, P) \leftarrow Update(G, P, f)$
8: **end while**
9: **while** $P \neq \emptyset$ **do**
10: $d \leftarrow d + 1$
11: $P_d \leftarrow Sel(P)$
12: $P \leftarrow P \backslash P_d$
13: $L_d \leftarrow Left(P_d) \cup Right(P_d)$
14: $(\tilde{F}_d^+, F_d) \leftarrow Reduction(L_d, G, (F_i)_{i=1,\ldots,(d-1)})$
15: **for** $h \in \tilde{F}_d^+$ **do**
16: $(G, P) \leftarrow Update(G, P, h)$
17: **end for**
18: **end while**
19: **return** G

2.2.2 Function Update

In the F_4 Algorithm the standard implementation of the Buchberger's Criteria such as the Gebauer and Möller installation [7] is used. Details about the function *Update* can be found in the section 2.1.2. The pseudocode of the function is shown in Algorithm 3.

2.2.3 Function Reduction

Function *Reduction*, Algorithm 5, performs polynomial division using methods of linear algebra.

Input of the function *Reduction* is a set L containing tuples of monomial and polynomial. These tuples were constructed in the main function of the F_4 Algorithm from all selected pairs.

First, the function *Reduction* calls the function *Symbolic Preprocessing* on the set L. This returns a set F of polynomials to be reduced. To use linear algebra methods to perform polynomial division, the polynomials have to be represented by a matrix. Each column of the matrix corresponds to a monomial. Columns have to be ordered with respect to the monomial ordering used so that the most right column corresponds to "1". Each row of the matrix corresponds to a polynomial from the set F. The matrix is constructed as follows. On the (i, j) position in the matrix, we put the coefficient of the term corresponding to j-th monomial from the i-th polynomial from the set F.

We next reduce the matrix to a row echelon form using, for example, Gauss-Jordan elimination. Note that this matrix is typically sparse so we can use sparse linear algebra methods to save computation time and memory. After elimination, we construct resulting polynomials by multiplying the reduced matrix by a vector of monomials from the right.

In the end, the function returns the set $\tilde{F}^+$ of reduced polynomials such that their leading monomials are not leading monomials of any polynomial from the set F of polynomials before reduction.

Algorithm 5 Reduction

Input:
L a finite set of tuples of monomial and polynomial
G a finite set of polynomials
$\mathcal{F} = (F_i)_{i=1,\ldots,(d-1)}$, where F_i is finite set of polynomials

Output:
$\tilde{F}^+$ a finite set of polynomials
F a finite set of polynomials

1: $F \leftarrow$ *Symbolic Preprocessing*$(L, G, \mathcal{F})$
2: $\tilde{F} \leftarrow$ Reduction to a Row Echelon Form of F
3: $\tilde{F}^+ \leftarrow \left\{ f \in \tilde{F} \mid \mathrm{LM}(f) \notin \mathrm{LM}(F) \right\}$
4: **return** $(\tilde{F}^+, F)$

2.2.4 Function Symbolic Preprocessing

Function *Symbolic Preprocessing*, Algorithm 6, starts with a set L of tuples each containing a monomial and a polynomial. These tuples were constructed in the main

function of the F_4 Algorithm from the selected pairs. Then, the tuples are simplified by the function *Simplify* and after multiplying polynomials with corresponding monomials, the results are put into the set F.

Next, the function goes through all monomials in the set F and for each monomial m looks for some polynomial f from the Gröbner basis G such $m = m' \times \mathrm{LM}(f)$ where m' is some monomial. All such polynomials f and monomials m' are after simplification multiplied and put into the set F. The goal of this search is to have for every monomial in F a polynomial in F with the same leading monomial. This will ensure that all polynomials from F will be reduced for G after polynomial division by linear algebra.

Algorithm 6 Symbolic Preprocessing

Input:
L a finite set of tuples of monomial and polynomial
G a finite set of polynomials
$\mathcal{F} = (F_i)_{i=1,\ldots,(d-1)}$, where F_i is finite set of polynomials
Output:
F a finite set of polynomials

1: $F \leftarrow \{multiply(Simplify(m, f, \mathcal{F})) \mid (m, f) \in L\}$
2: $Done \leftarrow \mathrm{LM}(F)$
3: **while** $\mathrm{M}(F) \neq Done$ **do**
4: m an element of $\mathrm{M}(F) \backslash Done$
5: $Done \leftarrow Done \cup \{m\}$
6: **if** m is top reducible modulo G **then**
7: $m = m' \times \mathrm{LM}(f)$ for some $f \in G$ and some monomial m'
8: $F \leftarrow F \cup \{multiply(Simplify(m', f, \mathcal{F}))\}$
9: **end if**
10: **end while**
11: **return** F

2.2.5 Function Simplify

The function *Simplify*, Algorithm 7, simplifies a polynomial $m \times f$ which is a product of a given monomial m and a polynomial f.

The function recursively looks for a monomial m' and a polynomial f' such that $\mathrm{LM}(m' \times f') = \mathrm{LM}(m \times f)$. The polynomial f' is selected from all polynomials that have been reduced in previous iterations (sets $\tilde{F}$). We select polynomial f' such that the total degree of m' is minimal.

This is done in the function *Symbolic Preprocessing* to insert polynomials that are mostly reduced and have a small number of monomials into the set F of polynomials to be reduced. This of course speeds up following reduction.

2.2.6 Selection strategy

For the speed of the F_4 Algorithm, it is very important how the critical pairs from the list of all critical pairs P are selected in each iteration. This of course depends on the implementation of the function *Sel*. There are more possible selection strategies:

- The easiest implementation is to select all pairs from P. In this case we reduce all critical pairs at the same time.

Algorithm 7 Simplify

Input:
m a monomial
f a polynomial
$\mathcal{F} = (F_i)_{i=1,\ldots,(d-1)}$, where F_i is finite set of polynomials
Output:
(m', f') a non evaluated product of a monomial and a polynomial

1: **for** $u \in$ list of all divisors of m **do**
2: **if** $\exists j$ $(1 \leq j \leq d)$ such that $(u \times f) \in F_j$ **then**
3: $\tilde{F}_j$ is the Row Echelon Form of F_j
4: there exists a (unique) $p \in \tilde{F}_j$ such that $\mathrm{LM}(p) = \mathrm{LM}(u \times f)$
5: **if** $u \neq m$ **then**
6: **return** $Simplify(\frac{m}{u}, p, \mathcal{F})$
7: **else**
8: **return** $(1, p)$
9: **end if**
10: **end if**
11: **end for**
12: **return** (m, f)

- If the function *Sel* selects only one critical pair then the F_4 Algorithm is the Buchberger's Algorithm. In this case the *Sel* function corresponds to the selection strategy in the Buchberger's Algorithm.
- The best function that Faugère has tested is to select all critical pairs with a minimal total degree. Faugère calls this strategy the *normal strategy for* F_4. Pseudocode of this function can be found as Algorithm 8.

Algorithm 8 Sel – The normal strategy for F_4

Input:
P a list of critical pairs
Output:
P_d a list of critical pairs

1: $d \leftarrow min\,\{\deg(\mathrm{lcm}(p)) \mid p \in P\}$
2: $P_d \leftarrow \{p \in P \mid \deg(\mathrm{lcm}(p)) = d\}$
3: **return** P_d

2.3 F_5 Algorithm

Since in the Buchberger Algorithm or in the F_4 Algorithm we spend much computation time to compute S-polynomials which will reduce to zero, the F_5 Algorithm [6] by Jean-Charles Faugère was proposed. The F_5 Algorithm saves computation time by removing useless critical pairs which will reduce to zero. The syzygies are used to recognize useless critical pairs in advance. For more details about syzygies look into [3].

There are several approaches how to use syzygies to remove useless pairs. For example the idea of [8] is to compute a basis of the module of syzygies together with the computing of the Gröbner basis of the given polynomial system. Then a critical pair can be removed if the corresponding syzygy is a linear combination of the the elements of the basis of syzygies.

The strategy of the F_5 Algorithm is to consider only principal syzygies without computing the basis of the syzygies. The principal syzygy is a syzygy such that $f_i f_j - f_j f_i = 0$ where f_i and f_j are polynomials. This restriction implies that not all useless critical pairs have to be removed so a reduction to zero can appear. However it was proved that if the input system is a regular sequence then there is no reduction to zero.

To show how to distinguish which pairs need not be considered we use the example taken from [6]. Consider polynomials f_1, f_2 and f_3. Then the principal syzygies $f_i f_j - f_j f_i = 0$ can be written as follows:

$$u(f_2 f_1 - f_1 f_2) + v(f_3 f_1 - f_1 f_3) + w(f_2 f_3 - f_3 f_2) \quad = \quad 0 \tag{2.4}$$

where u, v and w are arbitrary polynomials. This can be also rewritten as

$$(uf_2 + vf_3)f_1 - uf_1 f_2 - vf_1 f_3 + wf_2 f_3 - wf_3 f_2 \quad = \quad 0. \tag{2.5}$$

We can see that all relations hf_1 are such that h is in the ideal generated by polynomials f_2 and f_3. So if we have computed Gröbner basis of the polynomials f_2 and f_3 it is easy to decide which new generated polynomials can be removed. We can remove all polynomials in the form $t \times f_1$ such that t is a term divisible by leading monomial of an element of the ideal generated by f_2 and f_3. Therefore the F_5 Algorithm is an incremental algorithm so if we have polynomials $f_1, \ldots, f_m$ on the input we have to compute all Gröbner basis of the following ideals: $(f_m), (f_{m-1}, f_m), \ldots, (f_1, \ldots, f_m)$ in this order.

Many reviews, implementations and modifications of the F_5 Algorithm has been made. Let us emphasize some of them. The first implementation of the F_5 was made by Jean-Charles Faugère himself in the language C. Then there is an implementation in Magma by A. J. M. Segers [9]. Another review and implementation in Magma was done by Till Stegers [10]. Since there is no proof of termination of the F_5 Algorithm see the modification [4] which terminates in any case.

3 Automatic generator

3.1 Reimplementation

3.2 Multiple eliminations solver

3.3 Removing unnecessary polynomials

3.4 Matrix partitioning

3.5 F4 strategy

4 Experiments

5 Conclusion

Bibliography

[1] Thomas Becker and Volker Weispfenning. *Gröbner Bases, A Computational Approach to Commutative Algebra*. Number 141 in Graduate Texts in Mathematics. Springer-Verlag, New York, NY, 1993. 5, 6

[2] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, Mathematical Institute, University of Innsbruck, Austria, 1965. 5

[3] David Cox, John Little, and Donald O'Shea. *Ideals, Varieties, and Algorithms : An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer, New York, USA, 2nd edition, 1997. 5, 12

[4] Christian Eder, Justin Gash, and John Perry. Modifying faugère's f5 algorithm to ensure termination. *ACM Communications in Computer Algebra*, 45(2):70–89, 6 2011. 13

[5] Jean-Charles Faugère. A new efficient algorithm for computing gröbner bases (f_4). *Journal of pure and applied algebra*, 139(1–3):61–88, 7 1999. 7

[6] Jean-Charles Faugère. A new efficient algorithm for computing gröbner bases without reduction to zero (f_5). In *Papers from the International Symposium on Symbolic and Algebraic Computation*, ISSAC '02, pages 75–83, New York, NY, USA, 2002. ACM. 12, 13

[7] Rüdiger Gebauer and Hans-Michael Möller. On an installation of buchberger's algorithm. *Journal of Symbolic Computation*, 6(2–3):275–286, 10 1988. 6, 10

[8] Hans-Michael Möller, Teo Mora, and Carlo Traverso. Gröbner bases computation using syzygies. In *Papers from the International Symposium on Symbolic and Algebraic Computation*, ISSAC '92, pages 320–328, New York, NY, USA, 1992. ACM. 13

[9] A. J. M. Segers. Algebraic attacks from a gröbner basis perspective. Master's thesis, Department of Mathematics and Computing Science, Technische Universiteit Eindhoven, 2004. 13

[10] Till Stegers. Faugère's f5 algorithm revisited. Master's thesis, Department Of Mathematics, Technische Universit ät Darmstadt, revisited version 2007. 13