Faugère's F5 Algorithm Revisited

Thesis For The Degree Of
Diplom-Mathematiker
By

Till Stegers

Advisor
Prof. Dr. Johannes Buchmann

September 2005 (revised April 27, 2007)

Department Of Mathematics
Technische Universität Darmstadt

Contents

List of Algorithms

1 Introduction

Discovered by Bruno Buchberger in 1965, Gröbner bases are now an indispensable tool in computational algebra, both for theory and applications. For our concerns, Gröbner bases are special bases of nonzero ideals in a multivariate polynomial ring $\mathbb{F}[\underline{X}]$ over a field $\mathbb{F}$. In his PhD thesis, Buchberger described an algorithm that, given a finite sequence of polynomials $F \subseteq \mathbb{F}[\underline{X}]$, computes a Gröbner basis $G \subseteq \mathbb{F}[\underline{X}]$ such that F and G generate the same ideal I.

Since a Gröbner basis enjoys nicer properties than an arbitrary polynomial sequence, many problems concerning I that are hard given only F become easy when G is available. Maybe the most fundamental such problem is the question of *ideal membership:* For an arbitrary polynomial $f \in \mathbb{F}[\underline{X}]$, decide whether $f \in I$. Certainly one of the most useful applications of Gröbner bases is that they can help in solving systems of multivariate polynomial equations, a problem of great interest in cryptography. In fact, computing a (certain kind of) Gröbner basis is usually the bulk of the work when solving such a system in this fashion.

Gröbner bases are so powerful that computing them is extremely hard: It can be shown that computing a Gröbner basis is, in the worst case, doubly exponential in the degree of the input polynomials. Fortunately, Gröbner bases for many examples can be obtained much faster, and all major computer algebra systems for desktop computers include a command named similar to `GroebnerBasis`.

As the basic Buchberger algorithm for computing Gröbner bases is quite inefficient, there has been a lot of effort [7, 9, 17, 18, 21] to improve Buchberger's algorithm and find new methods. Following an idea of Lazard [19], Jean-Charles Faugère suggested the algorithm F_4, which translates the polynomial arithmetic of a Gröbner basis computation to the solution of linear algebra problems over the coefficient field. This clever idea resulted in a major speed-up of Gröbner basis computations and has since drawn a lot of interest. Several public implementations and evaluations of F_4 are available [5, 20, 23, 24], and the algorithm is implemented in computer algebra systems such as Magma [10, 25].

As a follow-up to F_4, Faugère suggested in [14] the F_5 algorithm, which uses ideas from a paper by Möller, Mora, and Traverso [21] to avoid redundant computations when computing a Gröbner basis. In particular, it takes care that the additional criteria used to detect useless computations in advance are not so costly that they result in fact in a slow-down instead of a speed-up. The success of Faugère's approach was prominently demonstrated when he announced in a posting to the Usenet newsgroup `sci.crypt` [13], and later in a CRYPTO 2003 paper [16], the solution of a challenge posed by Patarin's using the $F_5/2$ variant of his algorithm.

While the literature on F_4 is growing, there are still not many publications treating

F_5. Save for Faugère's paper [14], only Bardet's works [2, 3], and, briefly, [1, 24] appear to treat F_5, although the algorithm was publicized in 2002. Bardet deeply explores the complexity theory of F_5 using algebraic geometry, but for proofs of the algorithm, she refers to the original paper, where they are only sketched. As for implementations, the author knows only of four, two of which are public, but not stable, namely those by Pearce [22], and Segers [24], respectively; Steel [26] and Faugère have efficient implementations, neither of which is available in source form.

In the present thesis, we try to overcome these deficiencies. After briefly giving the foundations of Gröbner basis theory, chapter 3 treats the F_5 algorithm. We present and prove the main theorem underlying F_5, and show, under certain assumptions, the correctness and termination of the algorithm. All these proofs were only sketched in the original paper. The pseudo code we list in section 3.3 corrects some minor errors in [14], and should be better suited for implementations. Generally, we hope to have made the presentation more accessible (at least to a reader with prior exposure to Gröbner basis algorithms) than it was possible under the space restrictions in the original paper proposing the algorithm.

We are proud to report the first stable public implementation, with code available on the author's homepage:

http://wwwcsif.cs.ucdavis.edu/~stegers/

The implementation is written in the Magma language, and thus not very efficient, but useful to support further research on F_5, including efficient implementations. As a first indicator of the performance of F_5, we report some benchmarks, comparing our implementation to F_4 using the Gebauer-Möller criteria as implemented by Segers during the course of his ambitious Master's project [24]. Resulting from work on the implementation of F_5, we pass on some hints that we feel could be of use to researchers implementing the algorithm in future projects, and, finally, conclude with a section on open problems.

Thanks! I would like to thank my advisor, Johannes Buchmann, for supporting me, Ralf-Philipp Weinmann for his help, availability, and encouragement, and Michael Joswig for serving on my committee in a time of need. I also would like to thank Magali Bardet and Allan Steel for helpful discussions.

My deep gratitude goes to my family and my friends, who supported me all along: Medda, Christoph, Fiete, my dear friend Vidya, and my friends Ha-Jü and Andreas.

Thank you all!

Till Stegers

Revisions. April 27, 2007: Fixed errors in Definition 3.8 (thanks to Jerzy Browkin). March 16, 2007: Fixed an error in the pseudocode (thanks to John Perry for reporting this!), and updated the web address for obtaining the source code.

2 Preliminaries

This section introduces and describes the fundamental tool and subject of this thesis, namely, the concept of a Gröbner basis of an ideal in a polynomial ring over a field. We make no claim of self-sufficiency; instead, we refer the reader to the standard text books [4] and [11]. A reader familiar with Gröbner bases may skip this chapter, or just skim through it if he or she is not acquainted with the notation we adopted from [4].

2.1 Notation

We introduce some notation used throughout this thesis. The set of natural numbers is denoted $\mathbb{N} = \{0, 1, 2, \dots\}$, and the set of positive natural numbers $\mathbb{N}_{>0} = \mathbb{N} \setminus \{0\}$.

Rings will always be assumed to contain an element 1 such that 1 is a neutral element. For a subset S of a ring R, we denote by $\langle S \rangle_R$ the ideal of R generated by S. If the ring R is understood, we will often just write $\langle S \rangle$ for the same object. The group of units of R is denoted $R^\times$.

Let $\mathbb{F}$ be a field and $\mathbb{F}[\underline{X}]$ the polynomial ring over $\mathbb{F}$ in the finite sequence of variables $\underline{X}$. The set of terms of $\mathbb{F}[\underline{X}]$ will be denoted $\mathcal{T}(\underline{X})$, or simply $\mathcal{T}$, and the set of monomials will be denoted $\mathcal{M} = \mathbb{F} \cdot \mathcal{T}$. The support of a polynomial $f \in \mathbb{F}[\underline{X}]$, i. e., the set of its terms, is written $\mathcal{T}(f) = \{t \in \mathcal{T} \mid t \text{ occurs in } f\}$. In particular $\mathcal{T}(0) = \emptyset$.

Note that the above is the terminology (and notation) used in [4] and large parts of the literature, whereas other works, such as [11], interchange the definitions of "monomial" and "term".

2.2 Order theory

This section briefly recalls the order theory we shall need.

Definition 2.1. A *(binary) relation* on a set M is a subset $R \subseteq M \times M$. The *inverse relation* of R is the relation $R^{-1} = \{(m_2, m_1) \mid (m_1, m_2) \in R\}$. Instead of $(m_1, m_2) \in R$, we also write $m_1 \, R \, m_2$. We say that

(i) R is *reflexive* if $(\forall m \in M)\ m \, R \, m$,

(ii) R is *symmetric* if $(\forall m, m' \in M)\ m \, R \, m' \Rightarrow m' \, R \, m$,

(iii) R is *antisymmetric* if $(\forall m, m' \in M)\ m \, R \, m'$ and $m' \, R \, m \Rightarrow m = m'$,

(iv) R is *strict* if $(\forall m, m' \in M)\ m \, R \, m' \Rightarrow m \neq m'$,

(v) R is *connex* if $(\forall m, m' \in M)\ m \, R \, m'$ or $m' \, R \, m$,

(vi) R is a *quasi-order* if R is reflexive and transitive,

(vii) R is a *partial order on* M if R is reflexive, antisymmetric and transitive,

(viii) R is *total* or *linear* order if R is a connex partial order. ■

We give some ways construct new relations from given ones.

Definition 2.2. Let R be a binary relation on M.

(i) The *diagonal of* M is the relation $\Delta(M) = \{(m, m) \mid m \in M\}$.

(ii) Set $R_0 = \Delta(M)$, $R_1 = R$, and for $j \geq 2$, set

$$R_j = \{(m_1, m_3) \in M \times M \mid (\exists m_2 \in M)\ m_1 \, R_{j-1} \, m_2 \text{ and } m_2 \, R_{j-1} \, m_3\}.$$

The relation

$$R^* = \bigcup_{j=0}^{\infty} R_j$$

is called the *reflexive transitive closure* of R.

(iii) The set $R_s = R \setminus \Delta(M)$ is the *strict part* of R. ■

Definition 2.3. Let R be a binary relation on M, and let $S \subseteq M$. Then $m \in S$ is an *R-minimal element* of S if there is no $m' \in S$ that is strictly below m, i. e., that satisfies $m' \, R_s \, m$. An *R-maximal element* of S is an R^{-1}-minimal element of S, i. e., an $m \in S$ such that there is no $m' \in S$ strictly above m. The relation R is said to be *well-founded*, respectively, *noetherian*, if all nonempty subsets of M have an R-minimal, respectively, R-maximal element. ■

Example 2.4. If R is a ring, then the set $\mathcal{I}(R)$ of ideals of R endowed with the set inclusion $\subseteq$ is a partial order. One can show (using the Axiom of Choice) that if $\mathbb{F}$ is a field, then $\mathcal{I}(R)$ is noetherian for every multivariate polynomial ring $R = \mathbb{F}[\underline{X}]$, and that this is equivalent to every ideal of R being finitely generated.

2.3 Gröbner bases

This section introduces Gröbner bases, which were first described in Bruno Buchberger's PhD dissertation.

Definition 2.5. A *term order* is a total order $\leq$ on $\mathcal{T}$ such that $1 \leq t$ for all $t \in \mathcal{T}$, and $t_1 \leq t_2 \Rightarrow st_1 \leq st_2$ for all $s, t_1, t_2 \in \mathcal{T}$. ■

Suppose $\underline{X} = x_1, \ldots, x_n$. Mapping each term $x_1^{\alpha_1}, \ldots, x_n^{\alpha_n}$ to its exponent vector $\alpha \in \mathbb{N}^n$ is clearly an isomorphism between the commutative monoids $(\mathcal{T}, \cdot)$ and $(\mathbb{N}^n, +)$. (Relying on the order of the variables, this isomorphism is not canonical.) So to define a term order, it suffices to give an order $\leq' \subset \mathbb{N}^n \times \mathbb{N}^n$ that makes $\mathbb{N}^n$ an ordered

monoid and satisfies $(0,\dots,0) \leq' \alpha$ for all $\alpha \in \mathbb{N}^n$. This is what we are going to do in the following examples of term orders.

The *total degree* of a term $t = X_1^{\alpha_1} \cdot \dots \cdot X_n^{\alpha_n}$ is $\deg(t) = \sum_{j=1}^n \alpha_j$. For a polynomial $f \in \mathbb{F}[\underline{X}]$, we set $\deg(f) = -\infty$ if $f = 0$ and $\deg(f) = \max\{\deg(t) \mid t \in \mathcal{T}(f)\}$ otherwise.

Example 2.6. The following relations are term orders.

- *Lexicographical* order corresponds to the product of the natural order on $\mathbb{N}$. It is also referred to as *lex* ordering and written $\leq_{\text{lex}}$. We have $\alpha \leq_{\text{lex}} \beta$ if and only if either $\alpha = \beta$, or there exists $1 \leq i \leq n$ such that $\alpha_i < \beta_i$ and for all $1 \leq j < i$, we have $\alpha_j = \beta_j$.
- *Reverse lexicographical* or *inverse lexicographical* is abbreviated *revlex* and defined by $\alpha \leq_{\text{revlex}} \beta$ if and only if $(\alpha_n, \dots, \alpha_1) \leq_{\text{lex}} (\beta_1, \dots, \beta_n)$.
- *Graded lexicographical order* or *total degree* order is abbreviated *glex* or *tdeg*. For $\alpha, \beta \in \mathbb{N}^n$, we have $\alpha \leq_{\text{glex}} \beta$ if and only if either $\sum_{i=1}^n \alpha_i < \sum_{i=1}^n \beta_i$ or these sums are equal and $\alpha \leq_{\text{lex}} \beta$.
- *Graded reverse lexicographical order* or *reverse total degree* order is abbreviated *grevlex*. We have $\alpha \leq_{\text{grevlex}} \beta$ if and only if either $\sum_{i=1}^n \alpha_i < \sum_{i=1}^n \beta_i$ or these sums are equal and $\alpha \leq_{\text{revlex}} \beta$.

Note that, somewhat nonintuitively, if $t_1, t_2 \in \mathcal{T}$ satisfy $\deg(t_1) > \deg(t_2)$, we do not necessarily have $t_1 > t_2$. For instance, if $x > y$ are two indeterminates, and the ring $\mathbb{F}[x, y]$ is ordered using the lexicographic ordering, then we have $x > y^2$.

With an ordering on the terms, we can now give the definition of the head term of a multivariate polynomial.

Definition 2.7. Let $f \in \mathbb{F}[\underline{X}] \setminus \{0\}$, and let $\leq$ be some term order on $\mathcal{T}(\underline{X})$. The *head term* of f is the term $\text{HT}_{\leq}(f) = \max \mathcal{T}(f)$, where the maximum is taken with respect to the order $\leq$. The *head coefficient* of f, written $\text{HC}_{\leq}(f)$, is the coefficient of $\text{HT}_{\leq}(f)$ in f. The *head monomial* of f is $\text{HM}_{\leq}(f) = \text{HC}_{\leq}(f) \cdot \text{HT}_{\leq}(f)$. For a subset $S \subseteq \mathbb{F}[\underline{X}]$, we denote by $\text{HT}(S)$ the set of head terms of all *nonzero* elements of S. ■

Convention. Most of the time, we will be working with one order exclusively, and not consider distinct term orders on the same ring. Hence we will from now on assume every multivariate polynomial ring $R[\underline{X}]$ to be endowed with a term order $\leq$ with strict part $<$. Consequently, we shall drop the relation $\leq$ from our notation, simply writing HT, HC, and HM .

Similar to the Euclidean algorithm in the univariate case, we introduce a notion of *reducing* multivariate polynomials, an analogue to long division with remainder.

Definition 2.8. Let $f, p \in \mathbb{F}[\underline{X}] \setminus \{0\}$ and let $t \in \mathcal{T}(f)$ occur with coefficient c in f. If $\text{HT}(p) \mid t$, then $r = f - \frac{ct}{\text{HM}(p)} p$ satisfies $t \notin \mathcal{T}(r)$, and we write

$$f \to_p r \ [t].$$

In this context, we say f *can be reduced to* r *modulo* p *by eliminating* t. We call p the *reductor*, f the *reductee*, and r the *reductum*. If t is left unspecified, we mean that such a term exists in $\mathcal{T}(f)$. In case $t = \mathrm{HT}(f)$, we speak of a *top-reduction*, and that f is *top-reducible* by p. For a subset P of $\mathbb{F}[\underline{X}]$, we write $f \to_P r$ if there exists $p \in P$ such that $f \to_p r$. We say f *is reducible modulo* P, or, more precisely, f *reduces to* r *modulo* P. By $\to_P^*$, we denote the reflexive transitive closure of $\to_P$. If $f \to_P^* f'$ and f' is not reducible modulo P, then we say that f' is a *normal form* of f with respect to $\to_P$. ■

With this notion of reduction, we are ready to give several equivalent characterizations of Gröbner bases. The proof of the equivalences is given in [4, p. 206–207].

Theorem 2.9. *Let* G *be a finite subset of* $\mathbb{F}[\underline{X}] \setminus \{0\}$. *Then* G *is a* Gröbner basis *if and only if one of the following equivalent conditions is satisfied.*

(i) *Every* $f \in \langle G \rangle$ *has a unique normal form with respect to* $\to_G$.

(ii) *For every* $f \in \langle G \rangle$, *there is a reduction* $f \to_G^* 0$.

(iii) *Every nonzero* $f \in \langle G \rangle$ *is reducible modulo* G.

(iv) *Every nonzero* $f \in \langle G \rangle$ *is top-reducible modulo* G.

(v) *For every* $s \in \mathrm{HT}(\langle G \rangle)$ *there exists* $t \in \mathrm{HT}(G)$ *with* $t \mid s$.

(vi) $\mathrm{HT}(\langle G \rangle) \subseteq \mathcal{T}\,\mathrm{HT}(G)$

(vii) *The polynomials* $h \in \mathbb{F}[\underline{X}]$ *that are in normal form with respect to* $\to_G$ *form a system of representatives for the partition* $\{f + \langle G \rangle \mid f \in \mathbb{F}[\underline{X}]\}$ *of* $\mathbb{F}[\underline{X}]$. ■

Albeit useful in many situations, the characterizations given in Theorem 2.9 do not suggest an algorithm to decide whether a given set of polynomials G is a Gröbner basis, since all conditions in Theorem 2.9 seem to require an infinite number of checks, such as testing whether all polynomials in $\langle G \rangle$ reduce to zero modulo G. Fortunately, this can be reduced to a finite number of polynomials, namely the *S-polynomials* generated by G.

Definition 2.10. Let $g_1, g_2 \in \mathbb{F}[\underline{X}] \setminus \{0\}$. The *S-polynomial* of g_1 and g_2 is the polynomial

$$\mathrm{spol}(g_1, g_2) = \mathrm{HC}(g_2)u_1 g_1 - \mathrm{HC}(g_1)u_2 g_2,$$

where $u_j = \frac{\mathrm{lcm}(\mathrm{HT}(g_1),\mathrm{HT}(g_2))}{\mathrm{HT}(g_j)}$ for $j = 1, 2$. ■

Notice that by construction of the S-polynomial, the common head term $u_1\,\mathrm{HT}(g_1) = u_2\,\mathrm{HT}(g_2) = \mathrm{lcm}\,(\mathrm{HT}(g_1), \mathrm{HT}(g_2))$ of the left and right summand cancels.

The following theorem is a major step towards an algorithmic construction of Gröbner bases, as it concludes in particular that a finite number of computations suffices to check whether a given set of polynomials is a Gröbner basis. A proof is contained in [4, Theorem 5.48].

Theorem 2.11. *Let $I \neq \{0\}$ be an ideal of $\mathbb{F}[\underline{X}]$. A subset $G \subseteq I \setminus \{0\}$ is a Gröbner basis if and only if for all distinct $g_1, g_2 \in G$, we have* $\mathrm{spol}(g_1, g_2) \to_G^* 0$. □

Definition 2.12. A polynomial $f \in \mathbb{F}[\underline{X}]$ is said to be *homogeneous of degree d* if all terms in $\mathcal{T}(f)$ have the total degree d. Suppose Z is a fresh variable, i. e., $Z \notin \underline{X}$. If $g = \sum_{j=1}^{k} a_j t_j$ is some polynomial in the terms $t_1, \ldots, t_k$ such that $d = \max\{\deg(t) \mid t \in \mathcal{T}(g)\}$, then the polynomial

$$f^h = \sum_{j=1}^{k} a_j t_j Z^{d-d_j},$$

where $d_j = \deg(t_j)$ for $j = 1, \ldots, k$, is called the *homogenization* of f. Clearly, f_h is homogeneous of degree d. The function $\mathbb{F}[\underline{X}] \to \mathbb{F}[\underline{X}, Z]$ mapping f to f^h is called the *homogenization morphism.* ■

Although the property of a polynomial being homogeneous is independent of the term order used, the indeterminate Z is often chosen to be the smallest variable in $\mathbb{F}[\underline{X}, Z]$, so as to minimize its impact on the head term of f when forming f^h.

One of the most useful features of homogeneous polynomials is the validity of the following (easy) proposition.

Proposition 2.13. *Suppose $h \in \langle F \rangle$, where $F = \{f_1, \ldots, f_n\}$ is a set of homogeneous polynomials. Then for all terms t of h, we have* $\deg(t) \geq \min_{f \in F} \deg(f)$.

Proof. In the situation of the proposition, there is a polynomial combination $h = \sum_{j=1}^{n} h_j f_j$. Now any term t of h must appear in some product $h_j f_j$, hence $\deg(t) \geq \deg(h_j f_j) \geq \deg(f_j)$. □

Note that not every ideal $\mathbb{F}[\underline{X}]$ has a basis consisting of homogeneous polynomials only. For instance, if g is an inhomogeneous polynomial, then every generator of the principal ideal $g\mathbb{F}[\underline{X}]$ is a nonzero scalar multiple of g and therefore inhomogeneous.

Equipped with the terminology we just introduced, we can now weaken the definition of a Gröbner basis, introducing the notion of a Gröbner basis up to some degree d. Basically, a Gröbner basis of the ideal I up to degree d has the nice reduction properties of a Gröbner basis of I with respect to polynomials of degree at most d. This is made precise in the following theorem, which is proved in a generalized form in [4, Theorem 10.39].

Theorem 2.14. *Suppose G is a subset of nonzero homogeneous polynomials in $\mathbb{F}[\underline{X}]$ and $I = \langle G \rangle$. Let $d \in \mathbb{N}$, and let $\to_d^*$ be the restriction of $\to_G^*$ to $\mathbb{F}[\underline{X}]_d = \{p \in \mathbb{F}[\underline{X}] \mid \deg(p) \leq d\}$. Let $I_d = \langle G \rangle \cap \mathbb{F}[\underline{X}]_d$. Then G is a* Gröbner basis up to degree d, *or a d-*Gröbner basis*, if the following equivalent statements are satisfied.*

(i) Every $f \in I_d$ has a unique normal form with respect to $\to_d$.

(ii) For every $f \in I_d$, there is a reduction $f \to_d^* 0$.

(iii) Every nonzero $f \in I_d$ is reducible modulo G.

(iv) *Every nonzero* $f \in I_d$ *is top-reducible modulo* G.

(v) *For every* $s \in \mathrm{HT}(I_d)$ *there exists* $t \in \mathrm{HT}(G)$ *with* $t \mid s$.

(vi) $\mathrm{HT}(I_d) \subseteq \mathcal{T}\,\mathrm{HT}(G)$

(vii) *The polynomials* $h \in \mathbb{F}[\underline{X}]_d$ *that are in normal form with respect to* $\to_G$ *form a system of representatives for the partition* $\{(f+I) \cap \mathbb{F}[\underline{X}]_d \mid f \in \mathbb{F}[\underline{X}]_d\}$ *of* $\mathbb{F}[\underline{X}]_d$.

■

2.4 The Buchberger algorithm

We recall the basic algorithm for computing Gröbner bases, given by Buchberger [6] in 1965. The discovery of this algorithm, named after him, enabled a computational treatment of problems in commutative algebra, such as deciding whether a polynomial belongs to the ideal generated by some sequence of polynomials. A proof of this center piece of Gröbner bases theory is contained in every text book on Gröbner bases, for instance in [4, Theorem 5.53].

Algorithm 1 Buchberger Algorithm

Input: $F = \{f_1, \ldots, f_m\} \subseteq \mathbb{F}[\underline{X}] \setminus \{0\}$
Output: a Gröbner basis of $\langle F \rangle$

```
1:  function BUCHBERGER(F)
2:      n ← m
3:      g_i ← f_i for 1 ≤ i ≤ n
4:      P ← {(g_i, g_j) | 1 ≤ i < j ≤ n}
5:      while P ≠ ∅ do
6:          select some (g_i, g_j) ∈ P
7:          P ← P \ {(g_i, g_j)}
8:          s ← spol(g_i, g_j)
9:          h ← some normal form of s modulo {g_1, ..., g_n}
10:         if h ≠ 0 then
11:             n ← n + 1
12:             g_n ← h
13:             P ← P ∪ {(g_i, g_n) | 1 ≤ i < n}
14:         end if
15:     end while
16:     return {g_1, ..., g_n}
17: end function
```

The Buchberger algorithm forms the basis for many Gröbner basis algorithms; we comment on some of its aspects. Recall that the elements of P are called *critical pairs.*

Experiments show that the computationally most intense part of the algorithm is the reduction of a polynomial s to a normal form h. In the naïve version of the Buchberger algorithm described above, and for many examples, h will be zero almost all the time. This is a huge waste of time, for if, given some polynomial s, one could find out – without much work, in particular without reducing s – whether it reduces to zero modulo $\{g_1, \ldots, g_n\}$, then the corresponding critical pair could be disposed of right away.

The way we described the Buchberger algorithm, programmers of concrete implementations will have to specify two strategies, namely, how to choose pairs, and how to choose reductors when computing a normal form. Both choices can dramatically affect the overall performance of the algorithm. For instance, the pair selection can be used to eliminate S-polynomials that would reduce to zero. To this end, numerous strategies have been suggested. We name Buchberger's first and second criterion [7], the Gebauer-Möller installation [17], the sugar strategy [18], and pair minimization [9]. Although they are not instances of Buchberger's method, the algorithm from [21] and Faugère's F_5 algorithm have a similar structure and aim to lower the number of reductions to zero using certain strategies as well.

Another approach to improve the performance of Buchberger's algorithm is to use less naïve reduction techniques. The most notable algorithm pursuing this approach is Faugère's F_4 algorithm, whose good performance has drawn considerable interest in the community. For details, we refer the reader to Faugère's original paper [12], as well as the publications [5, 20, 23, 24, 25].

Reducing as few polynomials to zero as possible is certainly a desirable property of a Gröbner basis algorithm. The caveat, however, is that this goal often conflicts with other objectives. For instance, the computational burden for a more sophisticated selection strategy can simply outweigh its performance gain (as reported by Möller et al. for their sample implementation in [21]), or the resulting polynomials might be fewer in number, but harder to reduce (e. g., since they are of a much higher degree or tend to have more terms). As we shall see later, the F5 algorithm is aimed to prevent a lot of reductions to zero in an efficient way.

As the already huge memory requirements of Gröbner basis algorithms are often a limiting factor, an additional objective of new algorithms is to improve the memory efficiency. This will not be the topic of the present thesis, and neither will be the study of the complexity of the Buchberger or F_5 algorithms.

2.5 Applications of Gröbner bases

The reduction of polynomials is the analogue of division in the multivariate setting. In view of the generality of this concept, it is not surprising that Gröbner bases are applied in numerous areas of mathematics, computer science, and engineering. We kindly refer the reader to [8] for an overview, and highlight just two examples in this section.

An easy, but nevertheless important problem that can be solved using Gröbner bases is the *ideal membership problem*, which can be phrased as follows: Given a finite sequence $g, f_1, \ldots, f_m$ in a multivariate polynomial ring $\mathbb{F}[\underline{X}]$ over a field $\mathbb{F}$, give an algorithm that

decides whether $g \in \langle f_1, \ldots, f_m \rangle$. Equipped with the Buchberger algorithm, the ideal membership problem can be reduced to the case where $f_1, \ldots, f_m$ is a Gröbner basis: Any ideal of $\mathbb{F}[\underline{X}]$ but the zero ideal possesses a Gröbner basis, and $\langle f_1, \ldots, f_m \rangle = \{0\}$ implies that all f_i are zero, which can be checked. Recursive application of Theorem 2.9 gives the desired algorithm: We just compute a normal form of g modulo $f_1, \ldots, f_m$, which is 0 if and only if $g \in \langle f_1, \ldots, f_m \rangle$.

Algorithm 2 Ideal Membership Algorithm

Input: a Gröbner basis $G \subseteq \mathbb{F}[\underline{X}]$, $f \in \mathbb{F}[\underline{X}]$
Output: **true** if $f \in \langle G \rangle$, **false** if not

```
function IdealMembership(f, G)
    if f = 0 then
return true
    else
        if (∃g ∈ G) HT(g) | HT(f) then
return IdealMembership(f − (HM(f)/HM(g)) g, G)
        else
return false
        end if
    end if
end function
```

One of the highlights of Gröbner basis theory is that it can be used to solve systems of multivariate equations over a perfect field $\mathbb{F}$ (i. e., a finite field or an extension of $\mathbb{Q}$), a result due to Trinks [27]. Suppose $\underline{X} = x_1, \ldots, x_n$ is a sequence of indeterminates, and $f_1, \ldots, f_m \in \mathbb{F}[\underline{X}]$. Let L be an extension field of $\mathbb{F}$. A *solution* of the system $F = (f_1, \ldots, f_m)$ in L^n is a sequence $(a_1, \ldots, a_m)$ in L^n such that $f(a_i) = 0$ for $1 \leq i \leq m$. The set of solutions of F in L^n is denoted $\mathcal{V}_L(F)$. An ideal I is called *zero-dimensional* if, for L algebraically closed, the set $\mathcal{V}_L(F)$ is finite. Suppose the ideal $I = \langle f_1, \ldots, f_m \rangle$ is zero-dimensional, and the x_n-coordinates of distinct points in $\mathcal{V}_{\mathbb{F}}(F)$ are distinct. Then there is a Gröbner basis G of I with respect to a lexicographic order that is of the form

$$\{x_1 - g_1, \ldots, x_{n-1} - g_{n-1}, g_n\},$$

where $g_i \in \mathbb{F}[x_n]$ for $1 \leq i \leq n$. By factoring the univariate polynomial g_n over $L[\underline{X}]$ and substituting the roots, we obtain

$$\mathcal{V}_L(F) = \{(g_1(l), \ldots, g_{n-1}(l), l) \mid l \in L,\ g_n(l) = 0\}.$$

If we are interested in solutions in $\mathbb{F}^n$ only, and $\mathbb{F}$ is finite, we can adjoin the so-called *field equations* $x_i^q - x_i$ to the ideal, forcing $\mathcal{V}_L(F) = \mathcal{V}_{\mathbb{F}}(F)$. For a somewhat more detailed account of this process, we refer the reader to [24].

The main application we are interested in is cryptography. For instance, the family of multivariate quadratic public key schemes relies on the reference problem that the

solution of certain types of multivariate systems of equations, where the polynomials involved have at most total degree two, is intractable, cf. [28, 29]. In fact, the so-called $\mathcal{MQ}$ problem is known to be $\mathcal{NP}$-complete. As computing a Gröbner basis is usually the computationally most intense step when solving a system of multivariate equations by means of the process sketched above, fast Gröbner basis algorithms are of immediate practical relevance when estimating secure key sizes for $\mathcal{MQ}$ schemes. Other situations where systems of multivariate equations arise and have been used for cryptanalytic efforts include stream ciphers [15].

3 Faugère's F5 Algorithm

As mentioned in the previous chapter, there have been several suggestions how to improve the performance of Gröbner basis algorithms. Becker, Weispfenning, and Kredel [4, p. 291] comment on the algorithm proposed in by Möller, Mora, and Traverso [21] as follows:

> "This version is indeed capable of detecting more superfluous critical pairs than any other known implementation; the cost of testing submodule membership, however, has thus far turned out to be too high to translate the deletion of more pairs into a computational gain."

This is where F_5 enters. When devising F_5, Faugère's aim was to overcome the deficiencies of the algorithm by Möller et al., making their optimizations practical. Announced already in the 1999 paper [12], "number 5" was described in 2002 in the paper [14].

We suggest to consult the paper [21] before reading [14], as it contains some of the main ideas of F_5, written in a much more accessible, if concise, style.

3.1 Syzygies and reductions to zero

In a nutshell, syzygies are solutions to a system of $\mathbb{F}[\underline{X}]$-linear equations in a finitely generated module of polynomials. They are also the key to several optimizations of the Buchberger algorithm that help to detect S-polynomials that reduce to zero. The term *syzygy* originates in celestial mechanics, where it describes a constellation of three celestial bodies on a straight line.

Definition 3.1. An element $s \in \mathbb{F}[\underline{X}]^m$ is called a *syzygy* with respect to a sequence $F = (f_1, \ldots, f_m)$ in $\mathbb{F}[\underline{X}]$ if $\sum_{i=1}^m s_i f_i = 0$. Often the sequence $f_1, \ldots, f_m$ is clear from the context and thus not explicitly mentioned. If $n \geq 2$, then for each pair f_i, f_j with $1 \leq i < j \leq m$, we have a trivial relation $f_i f_j - f_j f_i = 0$, giving rise to the *trivial* or *principal syzygies* of the form $\pi_{ij} = f_j \mathbf{e}_i - f_j \mathbf{e}_j$, where $\mathbf{e}_k$ is the k-th standard basis vector in $\mathbb{F}[\underline{X}]^m$. ■

The following exact sequence of $\mathbb{F}[\underline{X}]$-modules illustrates the situation. Here v_F denotes the *evaluation homomorphism* with respect to F, defined by $v_F(h_1, \ldots, h_m) = \sum_{j=1}^m h_j f_j$.

$$0 \longrightarrow \mathrm{Syz} \hookrightarrow \mathbb{F}[\underline{X}]^m \xrightarrow[v_F]{} \langle F \rangle \longrightarrow 0$$

The set of all syzygies of a given system $F = (f_1, \ldots, f_m)$ is denoted $\mathrm{Syz}(f_1, \ldots, f_m)$. Since it is the kernel of the evaluation homomorphism v_F, the set $\mathrm{Syz}(f_1, \ldots, f_m)$ forms

an $\mathbb{F}[\underline{X}]$-module under component-wise multiplication and addition. The submodule generated by all *principal* syzygies of $f_1, \ldots, f_m$ will be denoted $\mathrm{PSyz}(f_1, \ldots, f_m)$. Clearly, since v_F is a homomorphism of $\mathbb{F}[\underline{X}]$-modules, two module elements $a, b \in \mathbb{F}[\underline{X}]^m$ satisfy $v_F(a) = v_F(b)$ if and only if $a - b$ is a syzygy.

It is not hard to see that a reduction to zero during the Buchberger algorithm corresponds to a syzygy. Indeed, if $s = \mathrm{spol}(g_k, g_l) = h_1 g_k - h_2 g_l$ reduces to 0 modulo a subset $G \subset \mathbb{F}[\underline{X}] \setminus \{0\}$, then there is a chain of reductions

$$s \;\to\; s - m_1 g_{i_1} \;\to\; s - m_1 g_{i_1} - m_2 g_{i_2} \;\to\; \ldots \;\to\; s - \sum_{j=1}^{k} m_j g_{i_j} = 0, \tag{3.1}$$

where the m_j are monomials (or 0) and the g_{i_j} are (not necessarily distinct) elements of G, satisfying $\mathrm{HT}(m_j g_{i_j}) \leq \mathrm{HT}(s) < \mathrm{HT}(h_1 g_k) = \mathrm{HT}(h_2 g_l)$. Without losing generality, we can assume that $g_i \neq g_j$ whenever $i \neq j$. Summing up monomials corresponding to the same g_i, we obtain a polynomial p_i for every g_i that occurs in the reduction of s. For each g_i that does not occur in the reduction, we set $p_i = 0$. By the definition of s, there are polynomials h_1, h_2 such that $s = h_1 g_k - h_2 g_l$, and as an element of $\langle F \rangle$, every g_j has a representation

$$g_j = \sum_{i=1}^{m} g_{i,j} f_i.$$

Together with (3.1), this gives a relation

$$\begin{aligned} 0 = s - \sum_{j=1}^{k} m_j g_{i_j} &= h_1 g_k - h_2 g_l - \sum_{j=1}^{|G|} p_j g_j \\ &= h_1 \sum_{i=1}^{m} g_{i,k}\, f_i - h_2 \sum_{i=1}^{m} g_{i,l}\, f_i - \sum_{j=1}^{|G|} p_j \sum_{i=1}^{m} g_{i,j}\, f_i \\ &= \sum_{i=1}^{m} \left(h_1 g_{i,k} - h_2 g_{i,l} - \sum_{j=1}^{|G|} p_j g_{i,j} \right) f_i, \end{aligned}$$

and thus a syzygy of F.

Note that in general not every syzygy corresponds to a reduction to zero during Buchberger's algorithm, since only a finite number of reductions are performed, but, as every $\mathbb{F}[\underline{X}]$-module, the module of syzygies is infinite if it is nontrivial.

A notion closely related to the module of syzygies is the property of a polynomial system F being *regular*.

Definition 3.2. A sequence $f_1, \ldots, f_m \in \mathbb{F}[\underline{X}] \setminus \{0\}$ is called a *regular* sequence if $\langle f_1, \ldots, f_m \rangle$ is proper and for every i such that $1 \leq i < m$, the polynomial f_i is not a zero divisor in $\mathbb{F}[\underline{X}]/\langle f_{i+1}, \ldots, f_m \rangle$, that is,

$$(\forall g \in \mathbb{F}[\underline{X}]) \quad g f_i \in \langle f_{i+1}, \ldots, f_m \rangle \;\Rightarrow\; g \in \langle f_{i+1}, \ldots, f_m \rangle.$$

■

Remark 3.3. Bardet's dissertation [3] contains an in-depth treatment of regular sequences, and their connection to the complexity of F_5 with a view on applications to cryptography and error-correcting codes (some of the results are also available in the preprint [2]). She explains, citing a result from [11], that if $F = (f_1, \ldots, f_m)$ is a regular sequence of homogeneous polynomials in n variables, then the ideal $\langle F \rangle$ has dimension $n - m$. In particular, *no overdetermined sequence can be regular.* As explained in section 2.5, multivariate polynomial systems arising in cryptographic applications are usually overdetermined, because they consist of m equations in n variables, plus the n field equations of the form $x_i^q - x_i$. This is bad news for cryptographers hoping to use F_5, since its termination relies on the fact that no reductions to zero occur, which in turn is only ensured by the input being a regular sequence. (Designers of crypto systems should not rejoice to early, though – Faugère writes [14] that it is possible to modify the algorithm "slightly" so that it terminates for non-regular sequences as well.)

Theorem 3.4. *Let $f_1, \ldots, f_m \in \mathbb{F}[\underline{X}] \setminus \{0\}$. Then these are equivalent:*

(i) All syzygies of F are generated by principal syzygies: $\mathrm{PSyz}(F) = \mathrm{Syz}(F)$

(ii) F is a regular sequence.

(iii) For every i such that $1 \leq i \leq m$, the sequence $f_i, \ldots, f_m$ is regular.

Proof. (ii) $\Leftrightarrow$ (iii) is obvious from the definition of a regular sequence.

(i) $\Rightarrow$ (iii): Let $g \in \mathbb{F}[\underline{X}]$ such that $gf_k \in \langle f_{k+1}, \ldots, f_m \rangle$. We want to show $g \in \langle f_{k+1}, \ldots, f_m \rangle$. Since (ii) $\Leftrightarrow$ (iii), we can assume without loss of generality that $k = 1$. By hypothesis, we have a syzygy $s \in \mathrm{PSyz}(F)$ such that $s_1 = g$. By assumption, the principal syzygies π_{ij} generate $\mathrm{Syz}(F)$, so there are polynomials α_{ij}, $1 \leq i < j \leq m$, such that

$$s = \sum_{i=1}^{m} \sum_{j=i+1}^{m} \alpha_{ij} \pi_{ij}.$$

Since the first component $\pi_{1j,1}$ of π_{1j} is f_j, we have $g = s_1 = \sum_{j=2}^{m} \alpha_{1j} \pi_{1j,1} = \sum_{i=2}^{m} \alpha_{1j} f_j$, so $g \in \langle f_2, \ldots, f_m \rangle$.

(iii) $\Rightarrow$ (i): Suppose $f_1, \ldots, f_m$ is a regular sequence. We proceed by induction on m. Trivially, we have $\mathrm{Syz}(f_m) = 0 = \mathrm{PSyz}(f_m)$. For the inductive step, it suffices to show that if $\mathrm{PSyz}(f_i, \ldots, f_m) = \mathrm{Syz}(f_i, \ldots, f_m)$ holds for $i = 2$, then it holds for $i = 1$. So let $s \in \mathrm{Syz}(F)$ be a syzygy. Then $s_1 f_1 = -\sum_{i=2}^{m} s_i f_i$, and since $f_1, \ldots, f_m$ is a regular sequence, $s_1 \in \langle f_2, \ldots, f_m \rangle$. Therefore $s_1 = \sum_{i=2}^{m} g_i f_i$ for some $g_2, \ldots, g_m \in \mathbb{F}[\underline{X}]$. But then

$$s_1 f_1 = \sum_{i=2}^{m} f_1 g_i f_i = -\sum_{i=2}^{m} s_i f_i,$$

so $s' = (f_1 g_2 + s_2, f_1 g_3 + s_3, \ldots, f_1 g_m + s_m) \in \mathrm{Syz}(f_2, \ldots, f_m) = \mathrm{PSyz}(f_2, \ldots, f_m)$. Lifting s' to $(0, s'_1, \ldots, s'_{m-1}) \in \mathrm{PSyz}(F)$ and subtracting principal syzygies, we get

$$(0, s') - g_2 \pi_{12} - \cdots - g_m \pi_{1m} = \left(\sum_{i=2}^{m} g_i f_i,\ s_2,\ \ldots,\ s_m \right) = s,$$

whence $s \in \mathrm{PSyz}(F)$.

□

3.2 Foundations of F_5

This section aims to elucidate the theory behind Faugère's F_5 algorithm. After introducing some new terminology, we state and prove the principal theorem that forms the basis of the F_5 algorithm as formulated in [14].

Definition 3.5. A *module term* or *positional term* is an element of $\mathbb{F}[\underline{X}]^m$ that has the form $t\mathbf{e}_k$, where t is a term and $\mathbf{e}_k$ is the k-th standard basis vector of $\mathbb{F}[\underline{X}]^m$. We denote the set of module terms by

$$\mathbf{T} = \{t\mathbf{e}_i \mid t \in \mathcal{T},\ 1 \leq i \leq m\}.$$

■

We will now extend the total order $<$ to the $\mathbb{F}[\underline{X}]$-module $\mathbb{F}[\underline{X}]^m$.

Definition 3.6. Suppose $g = (g_1, \ldots, g_m)$ and $h = (h_1, \ldots, h_m)$ are in $\mathbb{F}[\underline{X}]^m \setminus \{0\}$. The *index* of g, written $\mathrm{index}(g)$, is the smallest $i \in \mathbb{N}$ such that g_i is nonzero.

Suppose $\mathrm{index}(g) = i, \mathrm{index}(h) = j$. We write $g \preceq h$ if and only if either

(i) $i > j$, or

(ii) $i = j$ and $\mathrm{HT}(g_i) \leq \mathrm{HT}(h_i)$.

Furthermore, we set $0 \preceq g$ for the null vector $0 \in \mathbb{F}[\underline{X}]^m$. We write $\prec$ for the strict part of $\preceq$, i. e., $g \prec h$ if and only if $g \preceq h$ and $g \neq h$. We will also freely use the dual relations $\succeq$ and $\succ$, and say g is *smaller*, respectively, *larger* than h if $g \preceq h$ or $g \succeq h$, respectively.

■

Let $g \in \mathbb{F}[\underline{X}]^m \setminus \{0\}$ be a module element with $\mathrm{index}(g) = i$. Extending the notion of a head term from polynomials to elements of $\mathbb{F}[\underline{X}]^m$, we call $\mathrm{HT}(g_i)\mathbf{e}_i$ the *module head term* of g, and denote it by $\mathrm{MHT}(g)$.

Lemma 3.7. *The relation $\prec$ is a strict well-founded quasi-order on $\mathbb{F}[\underline{X}]^m$. In particular, $\preceq$ is a partial order on the set of module terms. The module head terms of any two $\prec$-minimal elements of a subset of $\mathbb{F}[\underline{X}]^m$ agree.*

Proof. It is straightforward to verify that $\prec$ is irreflexive, (trivially) antisymmetric and connex. It remains to show that $\prec$ is transitive and well-founded. Suppose that $\emptyset \neq G \subseteq \mathbb{F}[\underline{X}]$. Since the index of all polynomials in G is bounded by m and $\leq$ is a well-order on the head terms of G, both $k := \max\{\mathrm{index}(g) \mid g \in G\}$ and $t := \min\{\mathrm{HT}(g_k) \mid g \in G, \mathrm{index}(g) = k\}$ are well-defined. Then $M := \{g \in G \mid \mathrm{index}(g) = k \text{ and } \mathrm{HT}(g_i) = t\}$ is the set of minimal elements of G. To see that $\prec$ is transitive, let $f \prec g$ and $g \prec h$. If $\mathrm{index}(f) = \mathrm{index}(h) = k$, then $\mathrm{HT}(f_k) < \mathrm{HT}(g_k) < \mathrm{HT}(h_k)$. Otherwise $\mathrm{index}(f) > \mathrm{index}(h)$, so again $f \prec h$.

The claim about $\preceq$ is a straightforward consequence. □

Definition 3.8. A *labeled polynomial* is an element $(t\mathbf{e}_i, p)$, where $t \in \mathcal{T}$, $p \in \mathbb{F}[\underline{X}]$, and $\mathbf{e}_i$ is some standard basis vector in $\mathbb{F}[\underline{X}]^m$.

For a labeled polynomial $r = (u\mathbf{e}_k, p)$, we define the *polynomial of* r by $\operatorname{poly}(r) = p$, the *signature of* r by $\mathcal{S}(r) = u\mathbf{e}_k$, the *head term of* r by $\operatorname{HT}(r) = \operatorname{HT}(p)$, the *head coefficient of* r by $\operatorname{HC}(r) = \operatorname{HC}(p)$, and the *head monomial of* r by $\operatorname{HM}(r) = \operatorname{HM}(p)$.

We also define the following operations on labeled polynomials. If t is a term, then $tr = (tu\mathbf{e}_k, tp)$. If $\lambda \in \mathbb{F}$, then $\lambda r = (u\mathbf{e}_k, \lambda p)$. Note that these operations are not induced by $\mathbb{F}[\underline{X}]^{m+1}$, and that the product gr for a polynomial $g \in \mathbb{F}[\underline{X}] \setminus (\mathbb{F} \cup \mathcal{T})$ remains undefined.

A labeled polynomial r is called *admissible* with respect to a sequence $F = (f_1, \ldots, f_m)$ of polynomials in $\mathbb{F}[X] \setminus \{0\}$ if there exists a $g \in \mathbb{F}[\underline{X}]^m \setminus \{0\}$ such that $v_F(g) = \operatorname{poly}(r)$ and $\operatorname{MHT}(g) = \mathcal{S}(r)$.

A labeled polynomial $r = (u\mathbf{e}_k, p)$ is called *normalized with respect to* F if $u \notin \operatorname{HT}(\langle f_{k+1}, \ldots, f_m\rangle)$. A pair (v, r), $v \in \mathcal{T}$, is called *normalized* if the labeled polynomial vr is normalized. A pair (r_1, r_2) of labeled polynomials is called a *normalized pair* if, for $w = \operatorname{lcm}(\operatorname{HT}(r_1), \operatorname{HT}(r_2))$, $u_i = \frac{w}{\operatorname{HT}(r_i)}$, the pairs (u_1, r_1), (u_2, r_2) are normalized and $u_2\,\mathcal{S}(r_2) \prec u_1\,\mathcal{S}(r_1)$. For such a normalized pair of labeled polynomials (r_1, r_2), let $\operatorname{spol}(r_1, r_2)$ be the labeled polynomial

$$(u_1\,\mathcal{S}(r_1),\ \operatorname{HC}(r_2)u_1 \operatorname{poly}(r_1) - \operatorname{HC}(r_1)u_2 \operatorname{poly}(r_2)).$$

■

What we call a labeled polynomial is called a *rule* by Faugère. As this might lead to confusion with *simplification rules* that are often, especially in verbal discussions, shortened to *rules* as well, we chose a different terminology.

Convention. Sometimes, we apply terminology to labeled polynomials which is, strictly speaking, only defined for elements of the polynomial $\mathbb{F}[\underline{X}]$. For instance, we might say that a set G of labeled polynomials is a Gröbner basis.

We kindly ask the reader to gracefully interpret such statements about labeled polynomials as statements about the polynomials with the labels removed, that is, substituting every labeled polynomial $\rho = (v\mathbf{e}_i, p)$ by its polynomial $\operatorname{poly}(\rho) = p$. For instance, in the above example, we actually mean that the set $\{\operatorname{poly}(g) \mid g \in G\}$ is a Gröbner basis.

Lemma 3.9. *Let $r = (u\mathbf{e}_k, p)$ be a labeled polynomial and let $F = (f_1, \ldots, f_m)$ be a sequence of nonzero polynomials. Suppose the module element $g \in \mathbb{F}[\underline{X}]^m$ satisfies $\mathcal{S}(r) = \operatorname{MHT}(g)$.*

(i) r is not normalized if and only if there is an $s \in \operatorname{PSyz}(F)$ such that $\operatorname{index}(s) = k$ and $\operatorname{MHT}(g - s) \prec \operatorname{MHT}(g)$.

(ii) If F is a regular sequence and r is admissible and normalized, then $\mathcal{S}(r)$ is minimal among

$$\{\operatorname{MHT}(g') \mid g' \in \mathbb{F}[\underline{X}]^m,\ v_F(g') = p\}. \tag{3.2}$$

Proof. (i) Suppose r is not normalized. Then there exists an $i > k$ and a term v such that $v\,\mathrm{HT}(f_i) = u$. Set $s = v\pi_{ki} \in \mathrm{PSyz}$. The index of s is k, and $\mathrm{MHT}(s) = v\,\mathrm{HT}(f_i)\mathbf{e}_k = u\mathbf{e}_k$. Cancelling the head term u, we get $\mathrm{HT}(g_k - s_k) < \mathrm{HT}(g_k)$, so $\mathrm{MHT}(g - s) \prec \mathrm{MHT}(g)$. Conversely, if there exists an $s \in \mathrm{PSyz}(F)$ such that $\mathrm{index}(s) = k$ and $\mathrm{MHT}(g - s) \prec \mathrm{MHT}(g)$, then $\mathrm{HT}(g_k - s_k) < \mathrm{HT}(g_k)$, so that $\mathrm{HT}(g_k) = \mathrm{HT}(s_k)$. Since $s \in \mathrm{PSyz}$ and $\mathrm{index}(s) = k$, $\mathrm{HT}(s_k)$ is a multiple of a finite product $\prod_j \mathrm{HT}(f_{i_j})$, where all $i_j > k$. In particular, g_k is top-reducible by some f_i with $i > k$.

(ii) Suppose F is a regular sequence. If r is admissible, then there exists a $g \in \mathbb{F}[\underline{X}]^m$ such that $\mathcal{S}(r) = \mathrm{MHT}(g)$ and $v_F(g) = p$. Assume g is not minimal among the set in (3.2), say $g' \in \mathbb{F}[\underline{X}]^m$ satisfies $v_F(g') = p$ and $\mathrm{MHT}(g') \prec \mathrm{MHT}(g)$. Then the difference $g - g'$ has index k and is a syzygy since v_F is a module homomorphism. If F is a regular sequence, $\mathrm{Syz}(F) = \mathrm{PSyz}(F)$ by Theorem 3.4. But then $g - g' \in \mathrm{PSyz}$, so by (i), r cannot be normalized. □

Closely connected to polynomial reductions are *standard representations*, as treated in, e. g., [4, p. 218–219]. Note that we are using the *polynomial* instead of the (in some sense equivalent) *monomial* version of the definition, cf. [4] for a discussion.

Definition 3.10. Let $0 \neq f \in \mathbb{F}[\underline{X}]$, P a finite subset of $\mathbb{F}[\underline{X}] \setminus \{0\}$, and $t \in \mathcal{T}$. A representation

$$f = \sum_{p \in P} q_p p,$$

where for every $p \in P$, q_p is a nonzero polynomial is called a *t-representation* of f with respect to P if for all $p \in P$ such that $q_p \neq 0$, we have $\mathrm{HT}(q_p p) \leq t$. If $t = \mathrm{HT}(f)$, we say this is a *standard representation* of f. ■

We now extend this well-known definition of a t-representation to labeled polynomials, imposing additional conditions on the signatures involved.

Definition 3.11. Let P be a finite set of labeled polynomials, and s, t labeled polynomials with $\mathrm{poly}(s) = f$, $\mathrm{poly}(t) = g$, where $f, g \neq 0$. We say that

$$f = \sum_{p \in P} \mu_p \,\mathrm{poly}(p)$$

is a *t-representation* of s with respect to P if for all $p \in P$ such that $\mathrm{poly}(p) \neq 0$

$$\mathrm{HT}(\mu_p)\,\mathrm{HT}(p) \leq \mathrm{HT}(t) \quad \text{and} \quad \mathrm{HT}(\mu_p)\,\mathcal{S}(p) \preceq \mathcal{S}(s).$$

If this is the case, we write $s = \mathcal{O}_P(t)$. Similarly, we write $s = o_P(t)$ if there exists a labeled polynomial t' satisfying $\mathcal{S}(t') \preceq \mathcal{S}(t)$ and $\mathrm{HT}(t') < \mathrm{HT}(t)$ such that $s = \mathcal{O}_P(t')$. ■

Theorem 3.12. *Let G be a finite subset of $\mathbb{F}[\underline{X}]$ with $0 \notin G$. Then G is a Gröbner basis if and only if every $f \in \langle G \rangle$ has a standard representation with respect to G.* □

The new criterion introduced by Faugère is inspired by the following theorem due to Lazard. A proof is contained in the standard reference [4, Theorem 5.64].

Theorem 3.13. *Let G be a finite subset of $\mathbb{F}[x_1, \ldots, x_n]$ with $0 \notin G$. Assume that for all $g_1, g_2 \in G$, $\operatorname{spol}(g_1, g_2)$ either equals zero or it has a t-representation with respect to G for some $t < \operatorname{lcm}(\operatorname{HT}(g_1), \operatorname{HT}(g_2))$. Then G is a Gröbner basis.* □

In 2002, Faugère published [14] a result that basically says that it suffices to check the hypothesis of Theorem 3.13 for all *normalized* pairs only. This theorem, which forms the basis of his F_5 algorithm, is proved in the remainder of the section.

The general structure of Faugère's original (sketched) proof in [14] is similar to the proof of Theorem 3.13 presented in [4, p. 220–221], but Faugère's presentation is not easy to comprehend and contains some errors. We will try to make the following more accessible. First we need a straightforward (and purely order-theoretical) lemma that we shall use several times in the proof.

Lemma 3.14. *Let $a_1 \succeq a_2 \succeq \ldots$ be a sequence in $\mathbb{F}[\underline{X}]^m$, and suppose $b \in \mathbb{F}[\underline{X}]^m$ satisfies $a_1 \succ b$. Let $n \in \mathbb{N}_{>0}$, and suppose $a' \in (\mathbb{F}[\underline{X}]^m)^n$ is a descending sequence consisting of b and $n-1$ members (with distinct indices) of the sequence $a_2, a_3, \ldots$. Then $a' \prec_{\text{lex}} (a_1, \ldots, a_n)$.*

Proof. Suppose $a'_k = b$. By construction, we have $a'_i \preceq a_{i+1}$ for $1 \leq i < k$. If $b \succ a_k$, then by transitivity $a'_{k-1} \prec b$, a contradiction. Let l be minimal such that $a'_l \prec a_1$. If $b = a'_k$, then $a'_i \preceq a_i$ for $k < i \leq n$ and $a'_l \prec a_l$ imply $a' \prec_{\text{lex}} a$. □

Note that we can still conclude $a' \preceq_{\text{lex}} a$ if we just require the weaker condition $a_1 \succeq b$.

Assumption 3.15. For the remainder of this section, assume the following. Suppose $F = (f_1, \ldots, f_m)$ be a sequence of monic polynomials in $\mathbb{F}[\underline{X}]$. Let $G = \{r_1, \ldots, r_n\}$ be a set of labeled polynomials such that $r_i \neq r_j$ whenever $1 \leq i < j \leq n$. For $1 \leq i \leq n$, set $g_i = \operatorname{poly}(r_i)$, and $G_1 = \{g_i \mid 1 \leq i \leq n\}$. Suppose the following criteria are satisfied:

(i) $\{(\mathbf{e}_1, f_1), \ldots, (\mathbf{e}_m, f_m)\} \subseteq G$

(ii) $G_1 \subseteq \langle F \rangle$

(iii) All labeled polynomials in G are admissible and monic.

Moreover, for two labeled polynomials $r_i, r_j \in G$, define the terms

$$\tau_{ij} = \operatorname{lcm}(\operatorname{HT}(r_i), \operatorname{HT}(r_j)) \quad \text{and} \quad u_{ij} = \frac{\tau_{ij}}{\operatorname{HT}(r_i)}.$$

Definition 3.16. Let $I = \langle F \rangle = \langle G_1 \rangle$ and $0 \neq f \in I$. Denote the group of permutations on $\{1, \ldots, n\}$ by Sym_n, its neutral element by id, and let us define the set $\mathcal{D}_f$ of *ordered representations of f with respect to G* by

$$\left\{ (s, \sigma) \in \mathbb{F}[\underline{X}]^m \times \operatorname{Sym}_n \mid \sum_{i=1}^{n} s_i g_{\sigma(i)} = f,\ \operatorname{HT}(s_1)\mathcal{S}(r_{\sigma(1)}) \succeq \operatorname{HT}(s_2)\mathcal{S}(r_{\sigma(2)}) \succeq \ldots \right\},$$

where we ease our notation by employing the convention $\operatorname{HT}(0) = 0$. (Recall that for the zero vector $0 \in \mathbb{F}[\underline{X}]^m$, we have by definition $0 \prec g$ for all nonzero $g \in \mathbb{F}[\underline{X}]^m$.)

The set $\mathcal{D}_f$ is not empty since $F \subseteq G_1$, ensuring that f has a representation with respect to the basis F of I. For an element $(s,\sigma) \in \mathcal{D}_{f,G}$, define

$$\mathcal{V}(s,\sigma) = \big(\mathrm{HT}(s_1)\mathcal{S}(r_{\sigma(1)}),,\ldots,\mathrm{HT}(s_n)\mathcal{S}(r_{\sigma(n)})\big)\,,$$

and let $\mathcal{V} = \{\mathcal{V}(s,\sigma) \mid (s,\sigma) \in \mathcal{D}_{f,G}\}$. We define a relation $\sqsubset$ on $\mathcal{D}_{f,G}$. Let $(s,\sigma),(s,\sigma') \in \mathcal{D}_{f,G}$, and let $\bar{v} = \mathcal{V}(s,\sigma)$, $\bar{v}' = \mathcal{V}(s',\sigma')$. We write $(s,\sigma) \sqsubset (s',\sigma')$ if and only if one of the following conditions is true:

(i) $\bar{v} \prec_{\mathrm{lex}} \bar{v}'$, where $\prec_{\mathrm{lex}}$ is the lexicographic order on $(\mathbb{F}[\underline{X}]^m)^n$ induced by $\prec$, i. e., $\bar{v} \prec_{\mathrm{lex}} \bar{v}'$ iff there exists i such that $\bar{v}_i \prec \bar{v}'_i$ and $\bar{v}_j = \bar{v}'_j$ whenever $1 \leq j < i \leq n$.

(ii) $\bar{v} = \bar{v}'$ and $\max_{i=1,\ldots,n} \mathrm{HT}(s_i g_{\sigma(i)}) < \max_{i=1,\ldots,n} \mathrm{HT}(s'_i g_{\sigma'(i)})$

(iii) $\bar{v} = \bar{v}'$ and $t := \max_i \mathrm{HT}(s_i g_{\sigma(i)}) = \max_i \mathrm{HT}(s'_i g_{\sigma'(i)})$ and

$$\big|\{i \mid \mathrm{HT}(s_i g_{\sigma(i)}) = t\}\big| < \big|\{i \mid \mathrm{HT}(s'_i g_{\sigma'(i)}) = t\}\big|$$

■

The proof of the main result in this section, Corollary 3.23, proceeds by Noetherian induction using the relation $\sqsubset$. Hence we need the following lemma.

Lemma 3.17. *The relation $\sqsubset$ is well-founded.*

Proof. Let $\emptyset \neq A \subseteq \mathcal{D}_{f,G}$, and set $A_0 = A$. Denote by $\pi_i \colon \mathcal{V} \to \mathbb{F}[\underline{X}]^m$ the projection on the i-th component. Then for $1 \leq i \leq n$, the set

$$A_i = \big\{(s,\sigma) \in A_{i-1} \mid \pi_i(\mathcal{V}(s,\sigma)) \text{ is } \prec\text{-minimal in } \{\pi_i(\mathcal{V}(s',\sigma')) \mid (s',\sigma') \in A_{i-1}\}\big\}$$

is not empty since $\prec$ is well-founded by Lemma 3.7. Thus A_n, which is the set of all $(s,\sigma) \in \mathcal{D}_{f,G}$ such that $\mathcal{V}(s,\sigma)$ is a $\prec_{\mathrm{lex}}$-minimal element of A, is not empty. As $<$ is a well-founded relation on $\mathbb{N}$, there is a nonempty set $S \subseteq A_n$ whose elements are minimal with respect to conditions (ii) and (iii). So $\sqsubset$ is a well-founded relation on $\mathcal{D}_{f,G}$. □

As we want to proceed by Noetherian induction, it will be helpful to know some more properties of $\sqsubset$-minimal elements. We will investigate this in the following lemmata.

Lemma 3.18. *Let $(s,\eta) \in \mathcal{D}_{f,G}$ be a $\sqsubset$-minimal element of $\mathcal{D}_{f,G}$. Then there is a permutation $\sigma \in \mathrm{Sym}_n$ such that $(s_{\sigma(1)},\ldots,s_{\sigma(n)},\sigma)$ is still $\sqsubset$-minimal and satisfies*

$$\mathrm{HT}(s_i)\,\mathcal{S}(r_{\sigma(i)}) = \mathrm{HT}(s_j)\,\mathcal{S}(r_{\sigma(j)}) \text{ and } \mathrm{HT}(s_i g_{\sigma(i)}) < \mathrm{HT}(s_j g_{\sigma(j)}) \;\Rightarrow\; j < i \qquad (3.3)$$

for all $i,j \in \{1,\ldots,n\}$.

Proof. By renumbering G and G_1, we may assume that η is the identity on $\{1,\ldots,n\}$. The claim is trivial if $i = j$. If $i < j$ satisfy the assumption of implication (3.3) and $\tau \in \mathrm{Sym}_n$ denotes the transposition that swaps i and j, then $(s',\sigma') = (s_{\tau(1)},\ldots,s_{\tau(n)},\tau)$ satisfies (3.3) for i,j as well. Moreover, it is still a $\sqsubset$-minimal element of $\mathcal{D}_{f,G}$. The claim follows by induction. □

Lemma 3.19. *Let (s, σ) be a $\sqsubset$-minimal element of $\mathcal{D}_{f,G}$. Then for all $i \in \{1, \dots, n\}$ such that $s_i \neq 0$, the pair (s_i, r_i) is normalized.*

Proof. By renumbering G and G_1, we may assume that η is the identity on $\{1, \dots, n\}$. Say $r_i = (v\mathbf{e}_k, g_i)$. As r_i is admissible by hypothesis (iii), there exists a $\hat{g} \in \mathbb{F}[\underline{X}]^m \setminus \{0\}$ such that $v_F(\hat{g}) = g_i$ and $\mathcal{S}(r_i) = v\mathbf{e}_k = \mathrm{MHT}(\hat{g})$. By definition, the pair $(\mathrm{HT}(s_i), r_i)$ is not normalized if and only if $\mathrm{HT}(s_i)r_i$ is not normalized, i. e., $\mathrm{HT}(s_i)v$ is top-reducible by $\langle f_{k+1}, \dots, f_m \rangle$. Define a set A of labeled polynomials by $A = \{g \in G \mid \mathcal{S}(g) \prec \mathbf{e}_k\}$. By hypothesis (i), A contains all labeled polynomials of the form $(\mathbf{e}_l, f_l)$, where $k < l \leq m$. Since $\mathrm{index}(\hat{g}) = k$ and $\mathrm{HT}(\hat{g}_k) = v$, there is a representation

$$s_i \hat{g}_k = r + \sum_{a \in A} \lambda_a \,\mathrm{poly}(a) \quad \text{where } r, \lambda_a \in \mathbb{F}[\underline{X}],\ \mathrm{HT}(r) < \mathrm{HT}(s_i \hat{g}_k). \tag{3.4}$$

Substituting this in the representation of f by s, we obtain a new representation of f:

$$\begin{aligned} f = \sum_{l=1}^{m} s_l g_l &= \sum_{l=1, l\neq i}^{m} s_l g_l + s_i \sum_{l=k}^{m} \hat{g}_l f_l = \sum_{l=1, l\neq i}^{m} s_l g_l + s_i \hat{g}_k f_k + \sum_{l=k+1}^{m} s_i \hat{g}_l f_l \\ &= \sum_{l=1, l\neq i}^{m} s_l g_l + r f_k + \sum_{a \in A} f_k \lambda_a \,\mathrm{poly}(a) + \sum_{l=k+1}^{m} s_i \hat{g}_l f_l. \end{aligned} \tag{3.5}$$

We can regard the representation (3.5) as an element (s', σ') of $\mathcal{D}_f$ by grouping together summands that belong to the same labeled polynomial (the polynomial of which we have always written as the rightmost factor in each product) and applying a suitable permutation to ensure that the entries are sorted by decreasing signature. Let $\bar{v}' = \mathcal{V}(s', \sigma')$. We aim to show $\bar{v} \prec_{\mathrm{lex}} \bar{v}'$, so that $(s', \sigma') \sqsubset (s, \mathrm{id})$, but $(s', \sigma') \neq (s, \mathrm{id})$.

To see this, first we note that the last two sums in (3.5) contribute only to module terms $\mathrm{HT}(s'_l)\,\mathcal{S}(r_l)$ of index $> k$, so $\mathrm{HT}(s'_l)\,\mathcal{S}(r_l) \prec \mathrm{HT}(s_i)\,\mathcal{S}(r_i)$. Hence the first entry where $\bar{v}$ and $\bar{v}'$ might differ is one of i, k', where $r_{k'} = (\mathbf{e}_k, f_k)$. Let us disregard the monotonicity of $\bar{v}, \bar{v}'$ for a moment. When passing from $\bar{v}$ to $\bar{v}'$, all entries of index k remain the same, except $\bar{v}_i$ and $\bar{v}_{k'}$. The entry $\bar{v}_i = \mathrm{HT}(s_i)\,\mathcal{S}(r_i)$ is omitted, and $\bar{v}_{k'} = \mathrm{HT}(s_{k'})\mathbf{e}_k$ is replaced by $\mathrm{HT}(s_{k'} + r)\,\mathcal{S}(r_{k'})$. We distinguish three cases.

- If $k' < i$, then $\mathrm{HT}(s_i)\,\mathcal{S}(r_i) \preceq \mathrm{HT}(s_{k'})\mathbf{e}_k$, and so $\mathrm{HT}(s_{k'}) \geq \mathrm{HT}(s_i v) > \mathrm{HT}(r)$. Therefore $\mathrm{HT}(s_{k'} + r)\mathbf{e}_k = \mathrm{HT}(s_{k'})\mathbf{e}_k$. As $i \neq k'$ and $\bar{v}_i$ is omitted, we have $\bar{v}'_j = \bar{v}_{j+1}$ for all $j \geq i$. Since

  $$\left|\{j \mid \mathrm{index}(\bar{v}'_j) = k\}\right| < \left|\{j \mid \mathrm{index}(\bar{v}_j) = k\}\right|,$$

  there exists an index $l \geq i$ such that $\mathrm{index}(\bar{v}'_l) > k = \mathrm{index}(\bar{v}_l)$, so that in particular $\bar{v}'_l \prec \bar{v}_1$. Lemma 3.14 yields $\bar{v}' \prec_{\mathrm{lex}} \bar{v}$.

- If $k' = i$, then $v = 1$, and $\mathrm{HT}(s_{k'}) = \mathrm{HT}(s_i) > \mathrm{HT}(r)$, so $\bar{v}_i$ is replaced by the smaller entry $\mathrm{HT}(r)\mathbf{e}_k$. The claim follows again by Lemma 3.14.

- If $k' > i$, then $\mathrm{HT}(s_{k'})\mathbf{e}_k \prec \mathrm{HT}(s_i)\,\mathcal{S}(r_i)$. This means $\mathrm{HT}(s_{k'}) < \mathrm{HT}(s_i v)$, so either $s_{k'} + r = 0$ or $\mathrm{HT}(s_{k'} + r)\mathbf{e}_k \prec \mathrm{HT}(s_i)\,\mathcal{S}(r_i)$. In either case, $\bar{v}_i$ is omitted, and $\bar{v}_{k'}$ is replaced by an entry that is not $\prec$-larger. By a similar reasoning as in Lemma 3.14 (only that now we have two $b's$), we see that $\bar{v}' \prec_{\mathrm{lex}} \bar{v}$. □

The lemma we just proved might seem remarkable at first, since we did not hypothesize the r_i to be normalized, and a multiple tr_i cannot be normalized if r_i is not. However, G always contains at least one normalized labeled polynomial, namely $(\mathbf{e}_m, f_m)$. For instance, consider the (extreme) case that $f_m = 1$. Since 1 is reducible modulo $\langle f_m \rangle$, not even the standard labeled polynomials $(\mathbf{e}_i, f_i)$, where $1 \leq i < m$, are normalized. A minimal representation of a nonzero polynomial $f \in \mathbb{F}[\underline{X}]$ will therefore be just $f = ff_m$, or (s, σ) with $s = (f, 0, \ldots, 0)$ and $r_{\sigma(1)} = (\mathbf{e}_m, f_m)$.

Lemma 3.20. *Suppose (s, σ) is a $\sqsubset$-minimal element of $\mathcal{D}_{f,G}$, and let*

$$t = \max\left\{\mathrm{HT}(s_i g_{\sigma(i)}) \mid s_i \neq 0\right\}.$$

(i) Among all pairs $(s_i, g_{\sigma(i)})$ that satisfy $\mathrm{HT}(s_i g_{\sigma(i)}) = t$, there is exactly one pair $(s_j, g_{\sigma(j)})$ such that $\mathrm{HT}(s_j)\,\mathcal{S}(r_{\sigma(j)})$ is maximal.

(ii) There is no pair $(s_k, r_{\sigma(k)}) \neq (s_j, r_{\sigma(j)})$ such that $\mathrm{index}(\mathcal{S}(r_{\sigma(k)})) = \mathrm{index}(\mathcal{S}(r_{\sigma(j)}))$ and $\mathrm{HT}(s_j g_{\sigma(k)}) = t$.

Proof. Again, we may assume that $\sigma = \mathrm{id}$. The existence of (s_j, r_j) is clear since the representations are finite. For the uniqueness in (i), it suffices to show (ii). Let $I = \{i \mid \mathrm{HT}(s_i g_i) = t\}$, $w = \max\{\mathrm{HT}(s_i)\,\mathcal{S}(r_i) \mid i \in I\}$ and $J = \{i \in I \mid \mathrm{index}(\mathcal{S}(r_i)) = \mathrm{index}(w)\}$. It suffices to show $|J| = 1$. Suppose for a contradiction that $|J| > 1$. Let $j_0 = \mathrm{index}(w)$. Then, as for every $i \in J$ the labeled polynomial r_i is admissible, there is a representation $g_i = \sum_{j=j_0}^{m} w_{ij} f_j$ with $\mathrm{HT}(s_i w_{ij_0})\mathbf{e}_{j_0} = \mathrm{HT}(s_i)\,\mathcal{S}(r_i) \preceq w$. Let $J^c = \{1, \ldots, n\} \setminus J$. We get a new representation of f:

$$\begin{aligned} f &= \sum_{i \in J} s_i g_i + \sum_{i \in J^c} s_i g_i = \sum_{i \in J} s_i w_{ij_0} f_{j_0} + \sum_{i \in J} s_i \sum_{j=j_0+1}^{m} w_{ij} f_j + \sum_{i \in J^c} s_i g_i \\ &= \left(\sum_{i \in J} s_i w_{ij_0}\right) f_{j_0} + \sum_{j=j_0+1}^{m} \left(\sum_{i \in J} s_i w_{ij}\right) f_j + \sum_{i \in J^c} s_i g_i \end{aligned} \tag{3.6}$$

Similarly as for (3.5), we obtain an element $(s', \sigma') \in \mathcal{D}_{f,G}$ from the representation (3.6). Let $\bar{v}' = \mathcal{V}(s', \sigma')$. Again, we want to show $\bar{v}' \prec_{\mathrm{lex}} \bar{v}$.

Let k be minimal such that $\bar{v}_k$ has index j_0. When passing from $\bar{v}$ to $\bar{v}'$, all entries of index $< j_0$ remain unchanged. Therefore to prove $\bar{v}' \prec_{\mathrm{lex}} \bar{v}$, it suffices to show that we have $(\bar{v}'_k, \ldots, \bar{v}'_n) \prec_{\mathrm{lex}} (\bar{v}_k, \ldots, \bar{v}_n)$.

Suppose $r_{j'_0} = (\mathbf{e}_{j_0}, f_{j_0})$. Define $u = s_{j'_0}$ if $j'_0 \in J^c$ and $u = 0$ if $j'_0 \in J$. In $\bar{v}'$, the entries $\bar{v}_l$, where $l \in J \cup \{j'_0\}$, are omitted and replaced by the entry

$$\nu = \mathrm{HT}\left(u + \sum_{i \in J} s_i w_{ij_0}\right) \mathbf{e}_{j_0}.$$

We see that $\nu \preceq \max\{\bar{v}_l \mid l \in J \cup \{j_0'\}\} \preceq \bar{v}_k$. By hypothesis, $|J| \geq 2$, so that $\bar{v}'$ has less entries of index j_0 than $\bar{v}$. Hence there exists an $l > k$ such that $\bar{v}_l' = 0$ or $\operatorname{index}(\bar{v}_l') > j_0$. In either case, $\bar{v}_l' \prec \bar{v}_k$, so we can apply Lemma 3.14 to conclude $\bar{v}' \prec_{\text{lex}} \bar{v}$. This contradicts the minimality of (s, id), whence the claim is proved.

□

Let $r = (t\mathbf{e}_i, p)$, $s = (u\mathbf{e}_j, q)$ be labeled polynomials. We say that r *$\mathcal{S}$-reduces to* $r' = (t\mathbf{e}_i, p')$ *modulo* s, written $r \to^{\mathcal{S}} r'$, if p top-reduces to p' modulo q, and, for $v \in \mathcal{T}$ such that $v\,\mathrm{HT}(q) = \mathrm{HT}(p)$, we have $uv\mathbf{e}_j \preceq t\mathbf{e}_i$. If $p' = 0$, we say that r *$\mathcal{S}$-reduces to 0 modulo* s.

We are now ready to prove the main theorem behind F_5. A sketch of the proof, with an overall structure remarkably similar to the proof in [4, p. 220–221], is contained in Faugère's paper [14]; the idea of using principal syzygies already appears in [21].

Theorem 3.21. *Let $w \in \mathcal{T}$, and $k^* \in \{1, \ldots, m\}$. Suppose the S-polynomial of every pair $r_i, r_j \in G$ such that (r_i, r_j) is normalized and $\mathcal{S}(\operatorname{spol}(r_i, r_j)) \prec w\mathbf{e}_{k^*}$ is either zero or $\operatorname{spol}(r_i, r_j) = o_G(u_{ij} r_i)$.*

Then for every nonzero admissible labeled polynomial $\tilde{f}$ such that $\operatorname{poly}(\tilde{f}) \in \langle F \rangle$ and $\mathcal{S}(\tilde{f}) \prec w\mathbf{e}_{k^*}$, the polynomial f has a standard representation with respect to G_1. Moreover, $\tilde{f}$ $\mathcal{S}$-reduces to 0 modulo G.

Proof. Let $I = \langle F \rangle = \langle G_1 \rangle$ and suppose the labeled polynomial $\tilde{f} = (v\mathbf{e}_\alpha, f)$ with $0 \neq f \in I$ is admissible with respect to F and satisfies $v\mathbf{e}_\alpha \prec w\mathbf{e}_{k^*}$. Since $\mathcal{D}_f \neq 0$, there is a $\sqsubset$-minimal element $(s', \sigma') \in \mathcal{D}_f$, necessarily with $\mathrm{HT}(s_i')\,\mathcal{S}(r_{\sigma'(i)}) \prec w\mathbf{e}_{k^*}$ for $1 \leq i \leq n$. Renumbering the elements of G and G_1 does not affect this bound, and so, together with Lemma 3.18, we can assume without loss of generality that there exists (s, id) such that $(s, \mathrm{id}) \in \mathcal{D}_f$ is a $\sqsubset$-minimal element satisfying (3.3). Moreover, with $\bar{v} = \mathcal{V}(s, \mathrm{id})$, we still have $\bar{v}_i \prec w\mathbf{e}_{k^*}$ for $1 \leq i \leq n$.

Our goal is to show that f has a standard representation with respect to G_1. Suppose the contrary, and define t, I, J as in the proof of Lemma 3.20. Let $i \in J$ and $j \in I \setminus \{i\}$. Then $t = \mathrm{HT}(s_i)\,\mathrm{HT}(g_i) = \mathrm{HT}(s_j)\,\mathrm{HT}(g_j)$, so τ_{ij} divides t. Consequently $u_{ij} \mid \mathrm{HT}(s_i)$ and $u_{ji} \mid \mathrm{HT}(s_j)$. Lemma 3.19 implies that $u_{ij} r_i$ and $u_{ji} r_j$ are normalized. By Lemma 3.20, we have $\operatorname{index}(\mathcal{S}(r_i)) < \operatorname{index}(\mathcal{S}(r_j))$, so that $u_{ji} r_j \prec u_{ij} r_i$. Thus the pair (r_i, r_j) is normalized.

Now define $m_i = \mathrm{HM}(s_i)$ and $m_j = \frac{\mathrm{HC}(s_i)}{\mathrm{HC}(s_j)}\,\mathrm{HM}(s_j)$. We calculate

$$\begin{aligned} m_i g_i - m_j g_j &= \mathrm{HC}(s_i)\,\mathrm{HT}(s_i) g_i - \mathrm{HC}(s_i)\,\mathrm{HT}(s_j) g_j \\ &= \mathrm{HC}(s_i) \left(\frac{u_{ij} t}{\tau_{ij}} g_i - \frac{u_{ji} t}{\tau_{ij}} g_j \right) \\ &= \mathrm{HC}(s_i) \frac{t}{\tau_{ij}} \operatorname{spol}(g_i, g_j). \end{aligned} \tag{3.7}$$

Since (r_i, r_j) is normalized and $\mathcal{S}(\operatorname{spol}(r_i, r_j)) \preceq \mathrm{HT}(s_i)\,\mathcal{S}(r_i) \prec w\mathbf{e}_{k^*}$, we have by hypothesis $\operatorname{spol}(r_i, r_j) = o_G(u_{ij} r_i)$ (the S-polynomial cannot be zero since otherwise replacing s_i by $s_i - \mathrm{HM}(s_i)$ and s_j by $s_j + \mathrm{HC}(s_i)\,\mathrm{HT}(s_j)$ gave a smaller representation).

Furthermore, $\mathrm{HT}(s_i)\,\mathrm{HT}(g_i) = \mathrm{HT}(s_j)\,\mathrm{HT}(g_j) = t$, hence $\mathrm{lcm}(\mathrm{HT}(g_i), \mathrm{HT}(g_j)) \mid t$. As $\mathrm{HT}(u_{ij} r_i) = \mathrm{lcm}(\mathrm{HT}(g_i), \mathrm{HT}(g_j))$, we find $\mathrm{spol}(r_i, r_j) = o_G(\mathrm{HT}(s_i) r_i)$. Together with (3.7), this gives

$$m_i g_i - m_j g_j = \mathrm{HC}(s_i) \frac{t}{\tau_{ij}} \mathrm{spol}(g_i, g_j) = o_G(\mathrm{HT}(s_i) r_i).$$

This yields a new representation of f:

$$f = o_G(\mathrm{HT}(s_i) r_i) + s_i' g_i - \frac{\mathrm{HC}(s_i)}{\mathrm{HC}(s_j)} s_j' g_j + \alpha s_j g_j + \sum_{k=1, k \neq i,j}^{n} s_k g_k, \tag{3.8}$$

where, for $k \in \{i, j\}$, $s_k' = s_k - \mathrm{HM}(s_k)$ and $\alpha = 1 + \frac{\mathrm{HC}(s_i)}{\mathrm{HC}(s_j)}$. Similar to the steps before, we obtain $(s', \sigma') \in \mathcal{D}_{f,G}$ from (3.8). Let $\bar{v}' = \mathcal{V}(s', \sigma')$. We claim that $\bar{v}' \prec_{\mathrm{lex}} \bar{v}$, contradicting the minimality of (s, id) once again.

By Definition 3.11, the expression $o_G(\mathrm{HT}(s_i) r_i)$ involves only summands of the form $m_r \,\mathrm{poly}(r)$, $r \in G$, such that $\mathrm{HT}(m_r)\,\mathrm{HT}(r) < \mathrm{HT}(s_i)\,\mathrm{HT}(r_i)$ and

$$\mathrm{HT}(m_r)\,\mathcal{S}(r) \leq \mathrm{HT}(s_i)\,\mathcal{S}(r_i).$$

Using (3.3), we conclude that $\bar{v}_l' = \bar{v}_l$ whenever $1 \leq l < i$. As $i \neq j$, we have $\bar{v}_i' \prec \bar{v}_i$, and thus $\bar{v}' \prec_{\mathrm{lex}} \bar{v}$.

This contradicts our initial assumption that $\mathrm{HT}(f) < t$. Therefore (s, id) is a standard representation of f with respect to G. Now the second claim is straightforward: we know there is at least one $i \in \{1, \ldots, n\}$ for which $\mathrm{HT}(s_i g_i) = \mathrm{HT}(f)$, and so $\mathrm{HT}(f - c s_i g_i) < \mathrm{HT}(f)$, where $c \in \mathbb{F}^\times$ such that $c\,\mathrm{HC}(s_i) = \mathrm{HC}(f)$. This is an $\mathcal{S}$-reduction, and we may continue recursively until we have reduced f to zero.

□

Definition 3.22. In the situation of the preceding theorem, we say that G is a *Gröbner basis up to signature* $w\mathbf{e}_{k^*}$ *of* $\langle F \rangle$, or simply that G *is a* $w\mathbf{e}_{k^*}$*-Gröbner basis of* $\langle F \rangle$.

Now we are ready to prove the central result of this section.

Corollary 3.23 (F5 Criterion)**.** *Suppose the S-polynomial of every pair* $r_i, r_j \in G$ *such that* (r_i, r_j) *is normalized is either zero or* $\mathrm{spol}(r_i, r_j) = o_G(u_{ij} r_i)$*. Then* G_1 *is a Gröbner basis of the ideal generated by* F*.*

Proof. Let $f \in \langle G_1 \rangle$. Then there is a representation $(s, \sigma) \in \mathcal{D}_{f,G}$. Let $w\mathbf{e}_{k^*}$ be some signature larger than $\max\{\mathrm{HT}(s_i)\,\mathcal{S}(r_{\sigma(i)}) \mid i \in \{1, \ldots, n\}\}$. By the hypothesis on the S-polynomials, we can apply Theorem 3.21, concluding that f has a standard representation with respect to G_1. Thus G is a Gröbner basis of $\langle F \rangle$ by Theorem 3.12. □

Let us point out that while Corollary 3.23 and Theorem 3.13 are quite similar, neither is a generalization of the other. Theorem 3.13 treats only polynomials and hypothesizes t-representations for *all* non-zero S-polynomials, whereas Corollary 3.23 uses labeled

polynomials and supposes only that *some* nonzero S-polynomials have a t-representation, taking the signatures of the involved polynomials into account.

The following result can be found as a remark in [14], except that the existence of G' is not postulated. This additional hypothesis is in fact always satisfied, *given* an algorithm that only treats S-polynomials of normalized pairs (such as F5 for regular sequences, as we shall see). But at this stage, the requirement seems indispensable.

Corollary 3.24. *If the hypothesis of Corollary 3.23 is satisfied by all S-polynomials of degree less or equal to d for some $d \in \mathbb{N}$, and there is a Gröbner basis G' of homogeneous polynomials such that $G_1 \supseteq G'$ and for all $g \in G' \setminus G_1$, $\deg(g) > d$, then G_1 is a Gröbner basis up to degree d of the ideal generated by F.*

Proof. Let f be a nonzero polynomial in $\langle F \rangle = \langle G_1 \rangle$ with $\deg(f) \leq d$. Since G' is a Gröbner basis, there is a reductor g of f in G'. In fact we even have $g \in G_1$, since all elements of G' are homogeneous of degree greater d. Thus G_1 is a d-Gröbner basis by [4, Theorem 10.39]. □

3.3 Pseudo code

In this section, we describe the F_5 algorithm in pseudo code. We emphasize that we have corrected several smaller mistakes in the pseudo code from [14] (for instance, the check for normalized pairs was erroneous).

The notation used is hopefully natural: $x \leftarrow v$ assigns the value v to the variable x, $(\,)$ denotes the empty sequence, and a sequence $(a_1, \ldots, a_n)$ is enlarged by simply defining a_{n+1}. The variables r and Rules will be considered global to the algorithm, and are shared by most routines. The term *r-index* refers to an index $i \in \mathbb{N}$ specifying a member r_i of the global sequence r. If A is a set, respectively, a sequence, we denote its cardinality, respectively, length, by $|A|$. If two finite sequences $a = (a_1, \ldots, a_k)$ and $b = (b_1, \ldots, b_l)$ are given, we denote their concatenation $c = (a_1, \ldots, a_k, b_1, \ldots, b_l)$ by $c \leftarrow a$ concat b.

Algorithm 3 F5 – Main loop

```
global r          // array of labeled polynomials
global Rules      // array of simplification rules
```

Input: A sequence $F = (f_1, \ldots, f_m)$ of nonzero homogeneous polynomials in $\mathbb{F}[\underline{X}]$
Output: a Gröbner basis of $\langle F \rangle$

1: **function** F5(F)
2: $m \leftarrow |F|$
3: Rules $\leftarrow ((\,))_{j=1}^{m}$
4: $r \leftarrow (\,)$
5: $r_m \leftarrow (\mathbf{e}_m, \mathrm{HC}(f_m)^{-1} f_m)$
6: $G \leftarrow (\emptyset)_{i=1}^{m}$
7: $G_m \leftarrow \{m\}$
8: **for** $i \leftarrow m-1, \ldots, 1$ **do**
9: $G_i \leftarrow$ AlgorithmF5(f_i, i, G)
10: **if** $(\exists k \in G_i)$ $\mathrm{poly}(r_k) = 1$ **then**
11: **return** $\{1\}$
12: **end if**
13: **end for**
14: **return** $\{\mathrm{poly}(r_k) \mid k \in G_1\}$
15: **end function**

Algorithm 4 F5 – Core routine

Input: $f \in \mathbb{F}[\underline{X}]$, $i \in \mathbb{N}$, G a sequence of sets of r-indices
Output: set G' of r-indices with $i \in G'$

1: **function** AlgorithmF5(f, i, G)
2: $r_i \leftarrow (\mathbf{e}_i, \mathrm{HC}(f)^{-1} f)$
3: $G' \leftarrow G_{i+1} \cup \{i\}$
4: $P \leftarrow \emptyset$
5: **for** $j \in G_{i+1}$ **do**
6: $P \leftarrow P \cup \textsc{CritPair}(i, j, i, G)$
7: **end for**
8: **while** $P \neq \emptyset$ **do**
9: $d \leftarrow \min\{\deg(p) \mid p \in P\}$
10: $P_d \leftarrow \{p \in P \mid \deg(p) = d\}$
11: $P \leftarrow P \setminus P_d$
12: $S_d \leftarrow \textsc{SPols}(P_d)$
13: $R_d \leftarrow \textsc{Reduction}(S_d, G', G)$
14: **for** $k \in R_d$ **do**
15: $P \leftarrow P \ \cup \bigcup\{\textsc{CritPair}(k, l, i, G) \mid l \in G'\}$
16: $G' \leftarrow G' \cup \{k\}$
17: **end for**
18: **end while**
19: **return** G'
20: **end function**

Algorithm 5 F5 – CritPair

Input: r_k, r_l are defined, i is the current iteration in F5
Output: corresponding critical pair if (r_k, r_l) or (r_l, r_k) normalized

function CritPair(k, l, i, G)
  $t \leftarrow \operatorname{lcm}(\operatorname{HT}(r_k), \operatorname{HT}(r_l))$
  $u_1 \leftarrow t/\operatorname{HT}(r_k)$
  $u_2 \leftarrow t/\operatorname{HT}(r_l)$
  if $\mathcal{S}(u_1 r_k) \prec \mathcal{S}(u_2 r_l)$ **then**
    // swap k and l
    $k' \leftarrow k$
    $k \leftarrow l$
    $l \leftarrow k'$
    $u_1 \leftarrow t/\operatorname{HT}(r_k)$
    $u_2 \leftarrow t/\operatorname{HT}(r_l)$
  end if
  Let $(t_1, \mathbf{e}_{k_1}) = \mathcal{S}(r_k)$
  Let $(t_2, \mathbf{e}_{k_2}) = \mathcal{S}(r_l)$
  if $u_1 t_1$ is top-reducible by G_{k_1+1} **then**
    return $\emptyset$
  end if
  if $u_2 t_2$ is top-reducible by G_{k_2+1} **then**
    return $\emptyset$
  end if
  return $\{(t, u_1, k, u_2, l)\}$
end function

Algorithm 6 F5 – SPols

Input: B a set of critical pairs
Output: F a sequence of r-indices

1: **function** SPols(B)
2: $F \leftarrow (\,)$
3: **for** $(t, u, k, v, l) \in B$ **do**
4: $c_1 \leftarrow \mathrm{HC}(r_k)$
5: $c_2 \leftarrow \mathrm{HC}(r_l)$
6: $s \leftarrow u\,\mathrm{poly}(r_k) - c_1/c_2 v\,\mathrm{poly}(r_l)$
7: **if** $s \neq 0$ and $\neg$IsRewritable(u, k) and $\neg$IsRewritable(v, l) **then**
8: $N \leftarrow |r| + 1$
9: $r_N \leftarrow (u\,\mathcal{S}(r_l), s)$
10: AddRule(N)
11: $F_{|F|+1} \leftarrow N$
12: **end if**
13: **end for**
14: sort F s.t. $i < j \Rightarrow \mathcal{S}(r_{F_i}) \prec \mathcal{S}(r_{F_j})$
15: **return** F
16: **end function**

Algorithm 7 F5 – Reduction

Input: T, G' sets of r-indices, $G = (G_i)_{i=1}^m$ sequence of sets of r-indices
Output: set of r-indices D

1: **function** Reduction(T, G', G)
2: $D \leftarrow \emptyset$
3: **while** $T \neq \emptyset$ **do**
4: choose $k \in T$ such that $\mathcal{S}(r_k)$ is $\prec$-minimal among $\{\mathcal{S}(r_{k'}) \mid k' \in T\}$
5: $T \leftarrow T \setminus \{k\}$
6: $h \leftarrow$ a normal form of $\mathrm{poly}(r_k)$ w.r.t. $\{\mathrm{poly}(r_j) \mid j \in G'\}$
7: $r_k \leftarrow (\mathcal{S}(r_k), h)$
8: $K, T' \leftarrow$ TopReduction$(k, G' \cup D, G)$
9: $D \leftarrow D \cup K$
10: $T \leftarrow T \cup T'$
11: **end while**
12: **return** D
13: **end function**

Algorithm 8 F5 – FindReductor

Input: k an r-index, G' a set of r-indices, $G = (G_i)_{i=1}^m$ sequence of sets of r-indices
Output: a set R of r-indices, possibly empty

1: **function** FindReductor(k, G', G)
2: $t \leftarrow \mathrm{HT}(r_k)$
3: **for** $j \in G'$ **do**
4: $t' \leftarrow \mathrm{HT}(r_j)$
5: Let $v_j \mathbf{e}_{k_j} = \mathcal{S}(r_j)$
6: **if** $t' \mid t$ **then**
7: $u \leftarrow t/t'$
8: **if** $u\,\mathcal{S}(r_j) = \mathcal{S}(r_k)$ or IsRewritable(u, j, k) or uv_j is top-reducible with respect to G_{k_j+1} **then**
9: **continue**
10: **else**
11: **return** $\{j\}$
12: **end if**
13: **end if**
14: **end for**
15: **return** $\emptyset$
16: **end function**

Algorithm 9 F5 – TopReduction

Input: k an r-index, G' a set of r-indices, $G = (G_i)_{i=1}^m$ sequence of sets of r-indices
Output: D, T two (possibly empty) sets of r-indices

1: **function** TopReduction(k, G', G)
2: **if** $\text{poly}(r_k) = 0$ **then**
3: print "Warning, reduction to zero – F_5 may not terminate"
4: **return** $\emptyset, \emptyset$
5: **end if**
6: $p \leftarrow \text{poly}(r_k)$
7: $J \leftarrow$ FindReductor(k, G', G)
8: **if** $J = \emptyset$ **then**
9: $p \leftarrow p / \operatorname{HC}(p)$
10: $r_k \leftarrow (\mathcal{S}(r_k), p)$
11: **return** $\{k\}, \emptyset$
12: **else**
13: Let j be the single element of J
14: $q \leftarrow \text{poly}(r_j)$
15: $u \leftarrow \operatorname{HT}(p) / \operatorname{HT}(q)$
16: $p \leftarrow p - \operatorname{HC}(p) / \operatorname{HC}(q) uq$
17: **if** $u\,\mathcal{S}(r_j) \prec \mathcal{S}(r_k)$ **then**
18: $r_k \leftarrow (\mathcal{S}(r_k), p)$
19: **return** $\emptyset, \{k\}$
20: **else**
21: $N \leftarrow |r| + 1$
22: $r_N \leftarrow (u\,\mathcal{S}(r_j), p)$
23: AddRule(N)
24: **return** $\emptyset, \{k, N\}$
25: **end if**
26: **end if**
27: **end function**

Algorithm 10 F5 – Simplification rules

Input: j, an r-index
1: **procedure** AddRule(j)
2: Let $(t, \mathbf{e}_i) = \mathcal{S}(r_j)$
3: $\text{Rules}_i \leftarrow ((t, j))$ concat Rules_i
4: **end procedure**

Input: $u \in \mathcal{T}$, k an r-index
Output: true or false
1: **function** IsRewritable(u, k)
2: $k' \leftarrow \text{Rewrite}(u, k)$
3: **return** $k \neq k'$
4: **end function**

Input: $u \in \mathcal{T}$, k an r-index
Output: an r-index
1: **function** Rewrite(u, k)
2: Let $(v, \mathbf{e}_l) = \mathcal{S}(r_k)$
3: **for** $j \leftarrow 1, \ldots, |\text{Rules}_l|$ **do**
4: Let $(t, j') = \text{Rules}_{l,j}$
5: **if** $t \mid uv$ **then**
6: **return** j'
7: **end if**
8: **end for**
9: **return** k
10: **end function**

3.4 Proof of F_5

We provide proof of the correctness, termination and, in some sense, efficiency, of F_5. Unfortunately, we were unable to free F_5 from the hypothesis that F is a regular sequence in order to show the termination of $\text{F5}(F)$, and some gaps remain.

Notation. During the algorithm presented in the previous section, sets of labeled polynomials are stored as indices referring to the global variable r. To ease the notational pain, we establish a notation for dereferencing these indices. Suppose S is a variable holding indices referring to elements of r, and suppose at a certain step of the algorithm, $\text{val}(r)$ and $\text{val}(S)$ are the values of r and S, respectively. When discussing this step, we will identify variables with their values, and we denote by $\hat{S}$ the set, resp., sequence, obtained from $\text{val}(S)$ by replacing every index j by the labeled polynomial $\text{val}(r)_j$.

Proposition 3.25 (Faugère's Proposition 2)**.** *Let $\widetilde{R}$ be the set of all labeled polynomials r_k occurring during the execution of* F5*. Then every element of $\widetilde{R}$ is admissible.*

Proof. Every labeled polynomial except for the standard polynomials $(\mathbf{e}_i, f_i)$ is built from others in one of four steps in TopReduction or SPols. Thus we proceed by induction: First we show that every standard labeled polynomial satisfies the claim, then we argue that any statement in the algorithm that alters or creates labeled polynomials produces only admissible ones.

Let $(\mathbf{e}_j, f_j)$ be a standard labeled polynomial. It is admissible since $f_j = v_F(\mathbf{e}_j)$.

It remains to show that step 9 in SPols and steps 10, 18, 22 in TopReduction create only admissible labeled polynomials. Consider line 10 in TopReduction. If $r = (t\mathbf{e}_i, p)$ is an admissible labeled polynomial with $p \neq 0$, then $r' = (t\mathbf{e}_i, \frac{1}{\mathrm{HC}(p)}p)$ is admissible, since only $\mathrm{HT}(p)$, not $\mathrm{HC}(p)$ matters for r' to be admissible.

We claim that the three remaining steps which introduce or change labeled polynomials all share the same form: Given two admissible labeled polynomials r_{j_1}, r_{j_2} and two terms $u_1, u_2 \in \mathcal{T}$ such that

$$u_2\, \mathcal{S}(r_{j_2}) \prec u_1\, \mathcal{S}(r_{j_1}), \tag{3.9}$$

a new labeled polynomial $r_{j_3} = (u_1\, \mathcal{S}(r_{j_1}), u_1 \operatorname{poly}(r_{j_1}) - u_2 \operatorname{poly}(r_{j_2}))$ is formed. Indeed, the signatures in (3.9) cannot be equal in TopReduction since FindReductor would have discarded such a reductor. In SPols, the relation (3.9) is ensured by the procedure CritPair which swaps the components of a critical pair if necessary.

We show that in this situation r is admissible. Since r_{j_1}, r_{j_2} are admissible, there exist $a, b \in \mathbb{F}[\underline{X}]^m$ such that $r_{j_1} = \sum_{j=k}^m a_j f_j$ and $r_{j_2} = \sum_{j=l}^m b_j f_j$, where $1 \leq k \leq l \leq m$, $\mathrm{index}(a) = k$, $\mathrm{index}(b) = l$, and $\mathrm{HT}(a_k)\mathbf{e}_k = \mathcal{S}(r_{j_1})$. This gives a representation

$$\operatorname{poly}(r_{j_3}) = \sum_{j=k}^{m} (u_1 a_j - u_2 b_j) f_j.$$

The index of $u_1 a - u_2 b$ is k, since $k \leq l$. If $k < l$, then $u_2 b_k = 0$; if $k = l$, then $u_2 \,\mathrm{HT}(b_k) < \mathrm{HT}(u_1)\,\mathrm{HT}(a_k)$ by (3.9). In either case, we have $\mathrm{HT}(\operatorname{poly}(r_{j_2})) = u_1\,\mathrm{HT}(a_k)$, so the new labeled polynomial r_{j_3} is admissible.

□

The next proposition shows that F_5 treats only normalized pairs, and that the routine CritPair does not reject any normalized pair.

Proposition 3.26. *Suppose that for every $j \in \{i+1, \ldots, m\}$, the set G_j is a Gröbner basis of the ideal generated by $f_j, \ldots, f_m$.*

(i) All labeled polynomials in G_i created during the execution of F5(f_i, i, G) *are normalized.*

(ii) Every tuple (t, u_1, k, u_2, l) returned by CritPair(k, l, i, G) *represents a normalized critical pair (r_k, r_l).*

(iii) If (r_k, r_l) is normalized, then CritPair(k, l, i, G) *returns a tuple (t, u_1, k, u_2, l).*

(iv) If (r_k, r_l) *is normalized, then* CritPair(l, k, i, G) *returns a tuple* (t, u_1, k, u_2, l).

Proof. (i) We proceed by induction on the creation of labeled polynomials: We argue that the standard labeled polynomials of the form $(\mathbf{e}_i, f_i)$ created during F_5 are normalized, and that, given normalized labeled polynomials, line 9 of SPols and lines 22 of TopReduction result only in normalized labeled polynomials.

F5, line 5 and AlgorithmF5, line 2: If a standard labeled polynomial $(\mathbf{e}_i, f_i)$, where $1 \leq i < m$, is not normalized, then the constant 1 is top-reducible by $\langle f_{i+1}, \ldots, f_m \rangle$. But then $G_{i+1} \cap \mathbb{F}^\times \neq \emptyset$, i. e., $\langle f_{i+1}, \ldots, f_m \rangle = \mathbb{F}[\underline{X}]$, in which case F5 had terminated already after step $i+1$ due to line 11. Hence $(\mathbf{e}_i, f_i)$ is normalized.

SPols, line 9: If a labeled polynomial r_N is created in line 9 of SPols from a normalized critical pair (t, u_1, k, u_2, l) produced by CritPair, then in particular $u_1 r_k$ is normalized. Since $\mathcal{S}(r_N) = u_1 \, \mathcal{S}(r_k)$, this implies r_N is normalized.

TopReduction, line 22: Due to the check in line 8 of FindReductor, line 22 of TopReduction yields a normalized labeled polynomial.

(ii) Regarding (ii), simply note that in lines 6–11, CritPair swaps the components of a given critical pair if necessary and that it ensures in lines 16 and 19 that both $u_1 r_k$ and $u_2 r_l$ are normalized. Hence any pair produced by CritPair is normalized.

(iii) Conversely, any normalized pair passes the checks in lines 15 and 18, proving (iii).

(iv) follows from (iii) and the swap performed in lines 6–11 of CritPair. □

Unfortunately, our proof of F_5 is not complete; it seemed within reach, but we were unable to show that certain optimizations were correct. One of these gaps concerns the so-called *simplification rules* where certain products ur_k of a term u and a labeled polynomial r_k are called *rewritable*.

We say that a module term $t\mathbf{e}_k$ *divides* another module term $v\mathbf{e}_l$, written $t\mathbf{e}_k \mid v\mathbf{e}_l$, if $k = l$ and there is a term u such that $ut = v$, i. e., $ut\mathbf{e}_k = v\mathbf{e}_l$.

Definition 3.27. Let r_k, r_l be labeled polynomials occurring during the execution of F5(F), and let u, v be terms. We say that the pair (u, r_k) is *rewritable by* (v, r_l) if $l > k$ and $t \, \mathcal{S}(r_l) = u \, \mathcal{S}(r_k)$. If $p = (r_i, r_j)$ is a critical pair, then we call p rewritable if one of (u_{ij}, r_i), (u_{ji}, r_j) is rewritable.

We state some conjectures regarding optimizations included in F_5. All of them are implicitly used by Faugère in his paper [14], but unfortunately he does not provide a proof, just examples in simplified cases. Although not having succeeded in proving them, after studying the paper [21], the author of the present thesis is convinced they – or similar statements ensuring the correctness of F_5 – can be proven using the linear algebra viewpoint described in [12, 21]. (Faugère also gives a linear algebra example in [14].) It is should therefore be a theoretically (and practically) rewarding task to find an "F_4-like" variant of F_5.

Conjecture 3.28. *Let* ur_k *be a product of a term* u *and a labeled polynomial* r_k *occurring during step* i *of* F5(F). *Let* $F_i = (f_i, \ldots, f_m, g_1, \ldots, g_{n-m})$, *where*

$$\{f_i, \ldots, f_m, g_1, \ldots, g_{n-m}\} = \hat{G}_i \setminus \{\mathrm{poly}(r_k)\}.$$

If there is a nonzero syzygy $s \in \mathrm{Syz}(F')$ *such that* $\mathrm{MHT}(s) \mid u\,\mathcal{S}(r_k)$*, then* ur_k *is redundant. In particular, if* ur_k *is a component of a critical pair, then the critical pair is redundant, and if* ur_k *is used as a reductor, then the reductor need not be used. More specifically,*

(i) if no labeled polynomial is reduced to zero in F5, then it suffices to treat critical pairs that are not rewritable;

(ii) if no labeled polynomial is reduced to zero in F5, then it suffices to employ only reductors ur_k *such that* (u, r_k) *is not rewritable; and*

(iii) it suffices to employ only reductors ur_k *such that* (u, r_k) *is normalized.*

We briefly describe how to derive the more specific statements in the above conjecture from the general case. Statements (i) and (ii) are both based on an idea that we first found in the paper [21] by Möller, Mora and Traverso: Suppose ur_k is rewritable by (v, r_l). Since r_l was not reduced to zero, there is a labeled polynomial $r_j \in \hat{G}_i \setminus \{r_k\}$ such that $\mathcal{S}(r_j) = \mathcal{S}(r_l)$. As r_l is admissible by Proposition 3.25, there exists a representation $h = (h_1, \ldots, h_m) \in \mathbb{F}[\underline{X}]$ such that $v_F(h_1, \ldots, h_m) = g_j$ and $\mathrm{MHT}(h_1, \ldots, h_m) = \mathcal{S}(r_l)$ divides $u\,\mathcal{S}(r_k)$. Thus $s = (h_1, \ldots, h_m, 0, \ldots, 0) - \mathbf{e}_j$ is a nonzero syzygy in $\mathrm{Syz}(F_i)$ such that $\mathrm{MHT}(s) \mid u\,\mathcal{S}(r_k)$. Concerning statement (iii), assume that $\hat{G}_{i+1}$ is a Gröbner basis of $\langle f_{i+1}, \ldots, f_m \rangle$, and that $\mathcal{S}(r_k) = t\mathbf{e}_i$. Since ur_k is not normalized, there is an $r_j \in \hat{G}_{i+1}$ such that $\mathrm{HT}(r_j) \mid ut$. Since $g_j \in \langle f_{i+1}, \ldots, f_m \rangle$, there is a representation $g_j = \sum_{j=i+1}^{m} h_j f_j$. Therefore $g_j f_i - \sum_{j=i+1}^{m} h_j f_i f_j = 0$, and so

$$s = g\mathbf{e}_i - \sum_{j=i+1}^{m} h_j \mathbf{e}_j = \sum_{j=i+1}^{m} h_j \pi_{ij} \in \mathrm{PSyz}(F) \subseteq \mathrm{Syz}(F_i)$$

is a syzygy with $\mathrm{MHT}(s) \mid u\,\mathcal{S}(r_k)$.

The reduction in F_5 is similar, but not identical to the one used in the homogeneous version of Buchberger's algorithm described in [23, 24]. Most notably, it deviates in the following points.

(i) Only top-reductions are performed.

(ii) Some reductors are discarded. Hence, polynomials might not necessarily be reduced to a normal form, as all reductors might have been rejected.

(iii) New polynomials might be created during the reduction.

(iv) Polynomials are treated in increasing order by signature.

Despite these differences, the following lemma shows that the function Reduction still does what one would expect.

Lemma 3.29. *If* $F = (f_1, \ldots, f_m)$ *is a regular sequence of homogeneous polynomials in* $\mathbb{F}[\underline{X}] \setminus \{0\}$*, then the following holds for any call* $\textsc{Reduction}(T, G)$ *made during the execution of* $\mathrm{F5}(F)$.

(i) The procedure Reduction *terminates after finitely many steps.*

(ii) Assume the S-polynomial corresponding to a normalized critical pair (t, u, r_j, v, r_k) *is*

$$r_l = (u\,\mathcal{S}(r_j), \mathrm{HC}(r_k)u\,\mathrm{poly}(r_j) - \mathrm{HC}(r_j)v\,\mathrm{poly}(r_k)). \tag{3.10}$$

Suppose D *is the result of a call* Reduction(T, G)*. Let* $\hat{D} = \{r_\alpha \mid \alpha \in D\}$*, and* $\hat{G} = \{r_\alpha \mid \alpha \in G\}$*. Then for all* $r_l \in \hat{T}$ *defined as in* (3.10)*,*

$$r_l = \mathcal{O}_{\hat{G}\cup\hat{D}}(r_l) = o_{\hat{G}\cup\hat{D}}(ur_j). \tag{3.11}$$

Proof. (i) *Termination.* Suppose s is the value of the variable k assigned in line 4 of Reduction(T, G). In fact, s is uniquely determined in this situation, because no two S-polynomials have the same signature (otherwise the left component of the second S-polynomial would be rewritable). Then s is passed to TopReduction, resulting in one of four possible return values:

(i) $\emptyset, \emptyset$: r_s is discarded because $\mathrm{poly}(r_s) = 0$

(ii) $\{s\}, \emptyset$: $\mathrm{poly}(r_s)$ is in normal form with respect to A

(iii) $\emptyset, \{s\}$: r_s is reduced by an element of A

(iv) $\emptyset, \{s, N\}$: a copy of r_s, reduced by A, is introduced

In cases (i) and (ii), s will not be treated again in Reduction, whereas in cases (iii) and (iv), the variable k will be assigned the value s again. In case (iii), this is obvious due to the uniqueness of s. In the situation of (iv), the new labeled polynomial r_N has a larger signature than r_s, hence $\mathcal{S}(r_s)$ will still be minimal.

Therefore every polynomial r_s is reduced until it is either discarded, as in case (i), or included in D, as in case (ii).

Say $t = \mathrm{HT}(r_s)$, and a polynomial is created in case (iv). Then in subsequent calls, FindReductor will not yield the same reductor for polynomials of head term t as before, since a new simplification rule was added. By construction, the new polynomial is homogeneous of the same degree and has a smaller head term than t, therefore it cannot serve to reduce t. So for every polynomial that is initially in $\hat{R}$, case (iv) is entered only a finite number of times. Furthermore, there cannot be an infinite number of polynomials created this way, since their head terms are strictly decreasing, and there are only a finite number of possible reductors.

Clearly, case (iii) cannot occur infinitely often, so we conclude that the function Reduction terminates after a finite number of steps.

(ii) *Representation of S-polynomials.* Suppose we are in the situation of the lemma. As argued above, r_l is repeatedly top-reduced by $\hat{G} \cup \hat{D}$. If r_l is eventually reduced to zero, there is nothing to show. Otherwise, suppose $(\mathcal{S}(r_l), p)$ is the reduced version of r_l whose r-index is included in D. Due to the top-reductions performed, there is a standard representation of $\mathrm{poly}(r_l)$

$$\mathrm{poly}(r_l) = cp + \sum_{j\in M} p_j\,\mathrm{poly}(r_j), \tag{3.12}$$

in the sense of Definition 3.10, where $M \subseteq G_2 \cup D$ is the set of r-indices of the reductors used, the polynomial p is monic, $c \in \mathbb{F}^\times$, and the p_j are nonzero polynomials.

The last **if**-clause in TopReduction ensures that for every $j \in M$ and every term $t \in p_j$, we have $t\,\mathcal{S}(r_j) \prec \mathcal{S}(r_l)$, so in particular $\mathrm{HT}(p_j)\,\mathcal{S}(r_j) \prec \mathcal{S}(r_l)$.

Now we see that (3.12) is an r_l-representation of r_l with respect to $\hat{G} \cup \hat{D}$ in the sense of Definition 3.11. This gives the first equation in (3.11). For the second, we just notice that $\mathcal{S}(r_l) \preceq \mathcal{S}(ur)$ and $\mathrm{HT}(r_l) = \mathrm{HT}(\mathrm{spol}(r_j, r_k)) < \tau jk = u\,\mathrm{HT}(r_j)$.

□

Theorem 3.30 (Faugère's Theorem 2)**.** *Let $F = (f_1, \ldots, f_m)$ be a regular sequence of homogeneous polynomials in $\mathbb{F}[\underline{X}] \setminus \{0\}$. Then the result of* F5(F) *is a Gröbner basis of the ideal generated by F.*

Proof. Aiming to apply Corollary 3.23, we verify that its hypotheses are satisfied.

Without loss of generality, we may assume that the polynomials $f_1, \ldots, f_m$ are monic, as this does not change the ideal and every f_i is divided by its head coefficient in line 5 of F5, respectively, line 2 of AlgorithmF5.

F5 may terminate "early" if during some stage i of the algorithm, the ideal $\langle F \rangle$ turns out to be the trivial ideal $\mathbb{F}[\underline{X}]$ in line 11. In this case, $\hat{G}_i$ certainly is a (probably highly redundant) Gröbner basis of $\langle F \rangle$. Hence we can assume that $\langle F \rangle \neq \mathbb{F}[\underline{X}]$.

Assumption 3.15 (i). By the argument above, we can assume every f_i is monic. Each labeled polynomial of the form $(\mathbf{e}_i, f_i)$ is included in the set G_i, and so eventually $(\mathbf{e}_i, f_i) \in \hat{G}_1$ for every $i = 1, \ldots, m$, since $\hat{G}_m \subseteq \cdots \subseteq \hat{G}_1$.

Assumption 3.15 (ii). The algorithm constructs the polynomials in $\hat{G}_1$ in four ways, which we can describe recursively:

- The algorithm includes input polynomials f_i,
- it creates S-polynomials from polynomials in $\langle F \rangle$,
- it divides polynomials in $\langle F \rangle$ by a constant, and
- it subtracts polynomials in $\langle F \rangle$ from others in $\langle F \rangle$.

As these operations do not leave the ideal, we get $\hat{G}_1 \subset \langle F \rangle$.

Assumption 3.15 (iii). This hypothesis is a consequence of Proposition 3.25, and of line 10 in TopReduction.

Hypothesis of Corollary 3.23. Suppose (r_k, r_l) is a normalized pair with $k, l \in G_1$. Then by Proposition 3.26 (iii)–(iv), the S-polynomial $\mathrm{spol}(r_k, r_l)$ is treated in some step i of the algorithm. Let D be the result of the call to Reduction(T, G) with $\mathrm{spol}(r_k, r_l) \in \hat{T}$. As every element of D is eventually included in the intermediate Gröbner basis in line 16 of AlgorithmF5, we have $D \subseteq G_i$ by the end of stage i. By Lemma 3.29, it follows that $\mathrm{poly}(r_l)$ was either reduced to zero modulo $\hat{G}_i$, or $\mathrm{spol}(r_k, r_l) = o_{\hat{G}_i}(u_{kl} r_k) = o_{\hat{G}_1}(u_{kl} r_k)$.

Therefore, all hypotheses of Corollary 3.23 are satisfied, and so $\hat{G}_1$ is a Gröbner basis of the ideal $\langle F \rangle$. □

It should be noted that in [14], Faugère sketches a proof of the above theorem that argues using truncated Gröbner bases, cf. Corollary 3.34.

The following conjecture states that the algorithm F5 terminates if there is no reduction to zero. Unfortunately, we were unable to complete Faugère's sketch given in [14], and so we can only give a partial proof of what otherwise would have been a theorem. Experimental evidence the author has collected with his implementation of F_5 seems to support the claim.

Conjecture 3.31. *Suppose $F = (f_1, \ldots, f_m)$ is a sequence of homogeneous monic polynomials such that no reduction to zero occurs during the algorithm* F5(F). *Then* F5(F) *terminates.*

Partial proof. Let R be the result of a call of the function Reduction and $\hat{G}_i$ the intermediate Gröbner basis prior to including the elements of $\hat{R}$. It suffices to show that if $R \neq \emptyset$, then $\langle \mathrm{HT}(\hat{G}_i) \rangle \neq \langle \mathrm{HT}(\hat{G}_i \cup \hat{R}) \rangle$, for then, as $\mathbb{F}[\underline{X}]$ is noetherian, eventually R will be the empty set and remain empty.

Suppose $\hat{R}$ is nonempty. Without loss of generality, we may furthermore assume that R is finite and $i = 1$. Choose $r_k \in \hat{R}$ such that $\mathcal{S}(r_k)$ is $\prec$-maximal. We show that r_k is not top-reducible by any other element of $\hat{G}_1 \cup \hat{R}$ and thus in particular $\langle \mathrm{HT}(\hat{G}_1) \rangle \neq \langle \mathrm{HT}(\hat{G}_1 \cup \{r_k\}) \rangle$.

Suppose by way of contradiction that there exists an $r_l \in (\hat{G}_1 \cup \hat{R}) \setminus \{r_k\}$ such that $\mathrm{HT}(r_l) \mid \mathrm{HT}(r_k)$, say $u\,\mathrm{HT}(r_l) = \mathrm{HT}(r_k)$ with $u \in \mathcal{T}$. Clearly $\mathrm{poly}(r_l) \notin \langle \hat{G}_2 \rangle$, otherwise r_k could have been reduced by $\hat{G}_2$ in line 6 of Reduction.

We distinguish four cases.

Case 1. ur_l is normalized. Note that $u\,\mathcal{S}(r_l) \neq \mathcal{S}(r_k)$, otherwise r_k would have been discarded due to the rewriting rules.

Case 1.1. $\mathcal{S}(r_k) \prec u\,\mathcal{S}(r_l)$. We claim that the critical pair $(\mathrm{HT}(r_k), u, r_l, 1, r_k)$ was introduced in the list. Indeed, (u, r_l) is normalized since $u\,\mathcal{S}(r_l)$ is not top-reducible by G_2 and by Proposition 3.26, r_k and therefore $(1, r_k)$ is normalized. Together with the assumption that $\mathcal{S}(r_k) \prec u\,\mathcal{S}(r_l)$, this implies that the critical pair is normalized, hence it was created by CritPair by Proposition 3.26 (iii)–(iv). The corresponding S-polynomial produced by the procedure SPols is $(u\,\mathcal{S}(r_l), ur_l - r_k)$. As the signature of a labeled polynomial is not changed during the reduction and by assumption no reduction to zero occurs, $\hat{R}$ contains some labeled polynomial with signature $u\,\mathcal{S}(r_l)$. But $\mathcal{S}(r_k) \prec u\,\mathcal{S}(r_l)$, contradicting the assumption that $\mathcal{S}(r_k)$ is maximal.

Case 1.2. $u\,\mathcal{S}(r_l) \prec \mathcal{S}(r_k)$. Since $\mathcal{S}(r_k)$ is maximal, r_k was the last labeled polynomial to be included in $\hat{D}$. In particular, when TopReduction is passed r_k, the labeled polynomial r_l is already in $\hat{D}$. But then $u\,\mathcal{S}(r_l) \prec \mathcal{S}(r_k)$ implies that r_k can be reduced by r_l, a contradiction.

Case 2. ur_l is not normalized. Since r_l is admissible by Proposition 3.25, there exists $s' \in \mathbb{F}[\underline{X}]^m$ such that $s'_1 \neq 0$, $v_F(s') = \mathrm{poly}(r_l)$, and $\mathrm{HT}(s'_1)\mathbf{e}_1 = \mathcal{S}(r_l)$. Let

$us'_1 = v + \sum_{i=2}^{m} \lambda_i f_i$ be a reduced representation, i. e., with $\lambda_2, \ldots, \lambda_m \in \mathbb{F}[\underline{X}]$ and $v \in \mathbb{F}[\underline{X}]$ not reducible by $\hat{G}_2$. Since us'_1 is reducible by $\hat{G}_2$, either $v = 0$ or $\mathrm{HT}(v) < \mathrm{HT}(us'_1)$. If $v = 0$, then $u\,\mathrm{poly}(r_l) \in \langle \hat{G}_2 \rangle$, so $\mathrm{poly}(r_k)$ would have been reduced by $\hat{G}_2$. Therefore $v \neq 0$. We have

$$u\,\mathrm{poly}(r_l) = us'_1 f_1 + \sum_{i=2}^{m} us'_i f_i = vf_1 + \sum_{i=2}^{m} (\lambda_i f_1 + us'_i) f_i. \tag{3.13}$$

Now define a set $T = \{t \in \mathcal{T}(v) \mid \mathrm{HT}(tf_1) > \mathrm{HT}(r_k)\}$. From the algorithm, recall the notation $r_1 = (\mathbf{e}_1, f_1)$. Since v is in normal form with respect to $\langle f_2, \ldots, f_m \rangle$, the labeled polynomial tr_1 is normalized for every $t \in T$.

Case 2.1. $T = \emptyset$. In this case there is a representation $\sum_{i=2}^{m} (\lambda_i f_1 + us'_i) f_i = \sum_{g \in \hat{G}_2} \mu_g g$ where the μ_g are polynomials that satisfy either $\mu_g = 0$ or $\mathrm{HT}(\mu_g g) \leq \mathrm{HT}(r_k)$. As the head term of $u\,\mathrm{poly}(r_l)$ is $\mathrm{HT}(r_k)$, it must occur in the righthand expression in (3.13) somewhere. We conclude that $\mathrm{HT}(vf_1) = \mathrm{HT}(r_k)$, in which case r_k would have been reduced by the normalized labeled polynomial $\mathrm{HT}(v)r_1$, or some $g \in \hat{G}_2$ satisfies $\mathrm{HT}(\mu_g g) = \mathrm{HT}(r_k)$, in which case r_k would have been reduced by g.

Case 2.2. $T \neq \emptyset$. This case is what makes the claim a conjecture rather than a theorem. Since v is in normal form with respect to $\hat{G}_2$, any labeled polynomial tr_1, where $t \in T$, is normalized. Faugère claims that all these labeled polynomials were included in some critical pair, and that therefore there exists a labeled polynomial $r_j \in \hat{G}_2$ such that $\mathcal{S}(r_j) = \mathrm{HT}(v)\mathbf{e}_1$ and $\mathrm{HT}(r_j) = \mathrm{HT}(r_k)$. If so, then r_k could have been reduced by r_j, a contradiction.

□

We could settle case 2.2 in the proof of Conjecture 3.31 if we knew that for any value of $\hat{G}_1$ during the algorithm, it was always true that $\hat{G}_1$ is a $w\mathbf{e}_{k^*}$-Gröbner basis of $\langle f_1, \ldots, f_m \rangle$, where $w\mathbf{e}_{k^*} = \max\{\mathcal{S}(r_k) \mid r_k \in \hat{G}_1\}$. If this was the case, then from the proof of Theorem 3.21, we would conclude that ur_l could have been $\mathcal{S}$-reduced by a *normalized* labeled polynomial $u'r_{l'}$ with $r_{l'} \in \hat{G}_1$. This would the reductor for r_k we were looking for.

The next theorem confirms that, indeed, principal syzygies are avoided by F_5. Corollary 3.33 concludes what we were striving for: There are no reductions to zero during F_5 if the input is a regular sequence.

Theorem 3.32. *Let $F = (f_1, \ldots, f_m)$ be a sequence of nonzero monic homogeneous polynomials. If a labeled polynomial r_k reduces to zero during the algorithm* F5(F)*, then there exists a syzygy*

$$s \in \mathrm{Syz}(F) \setminus \mathrm{PSyz}(F) \quad \textit{such that} \quad \mathrm{MHT}(s) = \mathcal{S}(r_k).$$

Proof. Suppose the labeled polynomial r_k reduces to zero. Then by Proposition 3.25, there exists $s \in \mathbb{F}[\underline{X}]^m$ such that $v_F(s) = 0$ and $\mathrm{MHT}(s) = \mathcal{S}(r_k)$. We see that $s \in \mathrm{Syz}(F)$. Now suppose $s \in \mathrm{PSyz}(F)$, and let $i = \mathrm{index}(s)$. Then s is a sum of multiples of principal syzygies π_{ij} with $j > i$, so $\mathrm{HT}(s_i) \in \mathrm{HT}(\langle f_{i+1}, \ldots, f_m \rangle)$. This contradicts the fact that r_k is normalized by Proposition 3.26. Hence $s \in \mathrm{Syz}(F) \setminus \mathrm{PSyz}(F)$. □

Corollary 3.33. *Suppose $F = (f_1, \ldots, f_m)$ is a regular sequence of nonzero homogeneous polynomials. Then $\mathrm{F5}(F)$ terminates, and no labeled polynomial is reduced to zero by the algorithm.*

Proof. If F is a regular sequence, then $\mathrm{Syz}(F) = \mathrm{PSyz}(F)$ by Theorem 3.4. The assertions follow from Theorem 3.32 and Conjecture 3.31. □

As we have promised on page 25, we can now prove an alternative version of Corollary 3.24. It is unfortunate that we have to require that F is a regular sequence (so that $\mathrm{F5}(F)$ terminates).

Corollary 3.34. *If the hypothesis of Corollary 3.23 is satisfied by all S-polynomials of degree less or equal to d for some $d \in \mathbb{N}$, and G_1 is a regular sequence, then G_1 is a Gröbner basis up to degree d of the ideal generated by F.*

Proof. $G' = \mathrm{F5}(G_1)$ satisfies the requirement that $G_1 \subseteq G'$, all $g \in G'$ are homogenous and either $g \in G_1$ or $\deg(g) > d$. The claim follows from Corollary 3.24. □

Finally, we give an overview between the theorems in this text and corresponding statements from Faugère's article [14]. The reader should be aware, however, that while this gives a general overview, the precise hypotheses and conclusions vary.

Faugère's F5 Paper	**This Thesis**
n/a	Theorem 3.21
Theorem 1	Corollary 3.23
Proposition 2	Proposition 3.25
Proposition 3	Proposition 3.26
Theorem 2	Theorem 3.30
Theorem 3	Conjecture 3.31
Theorem 4	Theorem 3.32
Corollary 3	Corollary 3.33
Remark 1	Corollary 3.34
partly in Theorem 2	Lemma 3.29
implicit	Conjecture 3.28

3.5 Implementing F_5

While Faugère maintains in [14] that it is "very easy" to modify the Buchberger algorithm to yield F_5, it was a considerable effort for the author to implement the algorithm from the pseudo code given in [14].

At the time of writing, we are aware of four other implementations, by Jean-Charles Faugère (C), Robert Pearce (Maple), A. J. M. Segers (Magma), and Allan Steel (C),

respectively. Probably efforts are being made in other research teams as well. Of these authors, only Pearce [22] and Segers [24] have published their code. However, neither implementation seemed to be stable in our tests. Therefore, we decided to implement F_5 ourselves. Our language of choice was the computer algebra system Magma [10] developed at the University of Sydney.

In the following, we share our experiences from this project.

Language While Magma has an easy syntax and a very nice structured approach to algebraic computation, its lack of certain language features turned out to be inconvenient when implementing F_5. In particular, we missed the support for global variables (after all, all labeled polynomials are stored in the global r!), pointers, objects, as well as more features to manipulate sparse matrices. Regarding an *efficient* implementation, the interpreted Magma code would of course not be able to keep up with an implementation in a compiled language such as C or C++. As one will certainly be experimenting with different variants of an algorithm, support for modularity is also a highly desirable feature of the programming language used. We therefore recommend implementors of F_5 to use an object-oriented language such as C++. This will also allow a specialized memory management, a feature that is desirable as Gröbner basis computations can quickly require huge amounts of memory.

Stages interdependent Unlike in the classic Buchberger algorithm, and also in F_4, pair selection and polynomial reduction in F_5 strongly depend on each other. For instance, rendering a labeled polynomial r_k inadmissible during the reduction would destroy the relationship between the signature $\mathcal{S}(r_k)$ and the polynomial $\mathrm{poly}(r_k)$. Consequently, decisions such as whether a pair involving r_k is normalized or whether a given multiple of r_k is rewritable might be affected. This interdependency makes it also harder to combine F_5 with other optimizations of the Buchberger algorithm.

Pre-reduce input Inspired by the option `ReduceInitial` for Magma's Buchberger algorithm, our implementation first tries to mutually reduce the input polynomials to a normal form. For most examples we tried, this resulted in a small speed-up.

Unnecessary reductions In our description, line 6 of Reduction computes a normal form of r_k with respect to G'. However, if TopReduction just returned via line 24, then r_k is already in normal form with respect to G'. Therefore it is suggested to execute line 6 of Reduction only if the preceding call to TopReduction returned a set T' of cardinality ≥ 1. On a similar note, line 8 passes the set $G' \cup D$ to TopReduction. However, since the polynomial r_k reduced in TopReduction is in normal form with respect to G, it is sufficient to pass the new polynomials $(G' \cup D) \setminus G$.

Keep basis reduced We experienced a speed-up by computing the (unique) *reduced* Gröbner basis G_i^{red} of the Gröbner basis $\{\mathrm{poly}(r_k) \mid k \in G_i\}$ after completing step i, and subsequently checking if polynomials are top-reducible with respect to G_i^{red} instead of

$\hat{G}_i$. It is not suggested to *replace* the polynomials in $\hat{G}_i$ completely, as this would almost certainly result in inadmissible polynomials.

F5 $\neq$ F5 The reader should carefully note that there are different descriptions of what actually constitutes the F5 algorithm. There is the original description given by Faugère in [14], in which he also mentions the variants $F_5/2$, F_5', and F_5''; Bardet [3] describes "F5-matriciel" and "F5-matriciel/2"; in section 3.3, we have given a description slightly different from the original suggestion. See section 3.7 for ideas how to improve the reduction in F_5.

Data structures To increase the performance and memory efficiency of the algorithm, more sophisticated data structures than those employed in out implementation are likely to be quite beneficial in practice. For instance, organizing the simplification rules using a tree structure rather than a list seems promising. If the input is not a regular sequence and a large number of polynomials is reduced to zero, then the labeled polynomials should be stored in a more dynamic data structure than an array.

Sorting the input Due to its incremental structure, F_5 does not treat every input polynomial equally. Therefore the performance of the algorithm can be considerably affected by the *order* of the input. It appears that this is the reason why Bardet suggests in [3] to order the input such that $\deg(f_m) \leq \cdots \leq \deg(f_1)$. (Actually, she suggests to order the sequence the other way around, but her algorithms treat f_i before f_{i+1}, contrary to our notation.)

Earlier pair elimination Critical pairs that turn out to be rewritable are eliminated in the function SPols, but until then they are stored in the queue. It would be worth investigating if it pays off to add to CritPair a check whether a given critical pair is rewritable. If this is the case, the pair can be disposed of right then, shortening the length of the queue. If not, the simplification rules that the pair was compared against should be memorized (e. g., by storing the indices of the last rules checked) to prevent SPols from looking at them a second time.

Termination As many examples, in particular those arising from cryptographic problems, are *not* regular sequences, it is very desirable to improve F_5 so that termination is guaranteed even if there are reductions to zero. Probably one has to store which pairs, respectively, reductors, were discarded as they were rewritable by a labeled polynomial which was reduced to zero later on, and treat those again.

3.6 Performance

We do not claim that our implementation of F_5 is particularly efficient, as already the choice of an interpreted language indicates. Rather, its primary goal is correctness – after all, we do not know of any other public implementation that terminates for all regular

sequences; it is meant to serve as a base for further research (cf. section 4.2). Nevertheless, we report some performance results and compare it to the F_4 implementation by Segers [24]. We also list the timings of Magma's highly efficient F_4 implementation (due to Allan Steel), indicating that – not surprisingly – it is playing in a completely different league than the other two. (To be fair, it should be noted that Segers' implementation does not employ sparse matrix techniques.)

We share some observations made during our experiments.

- For many examples, there are considerably less reductions to zero. At the same time, however, the *total* number of polynomials considered is sometimes higher (examples Weispfenning 94, Segers). Confirming Corollary 3.33, we did not observe any reduction to zero for regular sequences.

- The reduction used in F_4 is very fast. Although F_4 has to reduce considerably more polynomials in the examples f744, Cyclic 7, and Katsura 8, it is faster or not much slower than F_5. (The degree and weight of the polynomials is also a factor in this, though.) It is therefore natural to ask how to speed up the reduction of F_5 using the ideas from F_4, cf. section 3.7.

- The maximal degree of the polynomials considered by F_5 is usually slightly higher than the maximal degree of the polynomials considered by F_4, probably causing a higher reduction workload per polynomial.

- The examples arising from multivariate cryptographic schemes (such as C^* and HFE, while only one HFE example is listed) that include the so-called field equations are never regular sequences (cf. Remark 3.3). Not surprisingly, F_5 fails to terminate quite often for this class of polynomial systems.

- To compare the memory consumption of F_4 and F_5, efficient implementations of comparable sophistication are needed.

Further in-depth analyses using equally sophisticated implementations is needed to call the race between F_4 and F_5, in particular the draw conclusions about the amount of memory needed. It would be very desirable if F_5 could be modified – probably the simplification rules have to be used more carefully – to ensure termination of the algorithm even for non-regular sequences.

The polynomial systems and the parameters for the examples (such as the fields and orderings used) are found in the appendix. All computations were done on an Athlon XP 2500 with 512 MB RAM running Magma 2.11-14.

The first column gives the name of the algorithm; the second column lists the total number of polynomials reduced by the algorithm, including the input as well as S-polynomials; the third column shows how many of these polynomials were in fact reduced to zero; the fourth column lists the CPU time consumed by the algorithm, measured using Magma's `Cputime()` command; the last column shows the largest degree of a polynomial treated.

The exact systems, term orders and fields used can be found in the file `examples.mag` in the distribution of the F_5 sample implementaton available at

http://www.cdc.informatik.tu-darmstadt.de/~stegers/.

Weispfenning94

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	45	18	0.509	14
F5	68	4	0.49	16
F4 Magma	46	?	0	15

Buchberger 87

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	13	7	0.02	6
F5	11	3	0.01	7
F4 Magma	15	?	0	7

Eco 6

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	92	60	0.56	9
F5	56	16	0.279	9
F4 Magma	92	?	0.011	10

Segers' HFE system

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	156	129	0.5	4
F5	380	218	3.65	10
F4 Magma	174	?	0.01	5

Möller-Mora-Traverso 1992

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	12	4	0.019	7
F5	10	0	0.011	8
F4 Magma	12	?	0	9

Uteshev-Bikker

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	35	27	0.36	7
F5	20	0	0.049	7
F4 Magma	43	?	0	8

Lichtblau

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	76	29	14.76	29
F5	76	0	5.25	34
F4 Magma	76	?	0.779	30

Liu

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	41	18	0.329	9
F5	19	0	0.031	9
F4 Magma	42	?	0	10

Gerdt93

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	16	3	0.14	6
F5	10	1	0.009	6
F4 Magma	15	?	0	7

Sym3-3

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	16	4	0.109	10
F5	11	0	0.029	10
F4 Magma	17	?	0	11

Trinks

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	46	37	0.181	5
F5	18	1	0.03	6
F4 Magma	26	?	0	7

Hairer1

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	34	24	0.31	8
F5	16	0	0.03	8
F4 Magma	28	?	0	9

f633

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	295	213	14.6	6
F5	75	2	0.46	8
F4 Magma	209	?	0.009	7

Katsura 5

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	77	62	0.801	6
F5	57	0	0.269	7
F4 Magma	68	?	0.01	7

Katsura 6

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	159	100	3.43	7
F5	74	0	0.731	7
F4 Magma	164	?	0.01	8

Katsura 7

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	371	195	25.7	8
F5	185	0	9.911	9
F4 Magma	377	?	0.05	9

Katsura 8

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	883	422	182.53	9
F5	423	0	135.129	11
F4 Magma	881	?	0.3	10

Cyclic 5

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	104	80	0.501	13
F5	39	0	0.179	13
F4 Magma	112	?	0	14

Cyclic 6

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	350	305	6.419	15
F5	218	16	7.07	17
F4 Magma	388	?	0.021	17

Cyclic 7

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	1909	1851	268.699	19
F5	1328	101	858.41	20
F4 Magma	2205	?	0.5	20

Gonnet 83

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	*aborted in degree 6 due to memory requirements*			
F5	7336	5089	3123.480	13
F4 Magma	8224	?	0.72	10

f744

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	1817	1602	365.180	8
F5	1346	477	2326.680	11
F4 Magma	1464	?	0.220	

Schrans-Troost 1990

Algorithm	Polynomials	Reds. to 0	CPU Time	Degree
F4 Segers	701	629	65.25	9
F5	≈ 1228	*aborted in step $i = 3$ after 12+ hours*		
F4 Magma	774	?	0.169	10

3.7 Combining F_4 and F_5: An attempt

For efficiency reasons, Faugère suggests in [14] to "translate" F_5 to an algorithm in "F_4 fashion." As current records [12, 25] in computing Gröbner bases were obtained using (presumably highly optimized) variants of the F_4 algorithm, we tried to develop an F_5 variant based on linear algebra techniques similar as those employed in F_4. Such a "hybrid" version specialized for fields of characteristic two, called $F_5/2$, was also used by Faugère to break Patarin's first HFE challenge [16]. Although we did not succeed in completing a working algorithm, we hope our experiences might be of some use to the reader when tackling the problem himself. We hope she or he will do better, and let the author know!

It should be noted that Bardet's dissertation [3] contains a description of an "F5-matriciel," however in a very concise and simplified version. Allan Steel, developer of the Gröbner basis package of Magma, reported in private communication [26] to have an

"F_4-like" implementation of F_5 as well, but it is unpublished. The most advanced matrix-based implementation is to our knowledge still Faugère's, included in his software FGb.

The F_4 algorithm benefits from transforming the reduction of polynomials to the problem of reducing a (potentially very large, and very sparse) matrix over the coefficient ring to row echelon form. Efficiently solving large systems of linear equations over finite fields is a difficult, but well-studied problem, which arises for instance in index-calculus methods for computing discrete logarithms. This translation enables the use of efficient sparse linear algebra techniques to speed up the reduction (a discussion of some of the techniques in use can be found in [16], for instance). Apart from the reduction, another advantages of F_4 is its flexibility: the pair selection strategy can be chosen freely, and it is suitable for non-homogeneous systems. However, even with a well-established pair selection criterion such as the Gebauer-Möller installation, still quite some polynomials reduce to zero – for instance, in the example Cyclic 7 above, Segers' F_4 implementation still reduces about 97% of the polynomials to zero!

The major feature of F_5, on the other hand, is that it prevents all reductions to zero caused by principal syzygies and by the syzygies caused by new polynomials (using the simplification rules). In many examples, its selection is optimal in the sense that there are no reductions to zero. But there is still a lot of work to complete on F_5: termination of the algorithm is not guaranteed in many important cases; no multireduction technique is employed; pair selection and reduction are interdependent due to the requirements that all polynomials treated be admissible. In general, F_5 does not seem to be understood very well, but the success of [16] makes it look promising.

In our approach to a hybrid version, we extended a basic version of F_4 (without the Simplify routine) to proceed incrementally and treat only normalized pairs. To this end, we assigned all polynomials a signature as in F_5, and prevented the addition or subtraction of two polynomials with the same signature, as this might have caused a cancellation in the respective representations, rendering the labeled polynomials inadmissible. (A phenonemon we refer to as as *signature corruption*.) This lead us to use a variant of the restricted Gaussian algorithm suggested by Bardet [3]: The polynomials are stored in the Macaulay matrix, with lower rows having smaller signatures, the signature of every one of them is stored separately, and is referenced by the row index. Moreover, the rows are sorted by increasing signature, so that the first row has the least signature. The signatures are not changed during the reduction, and the only operation permitted during the algorithm is to add a scalar multiple of row i to row j and store the result in row j, provided that $i < j$.

We also tried to use the criteria employed by F_5 in the routine FindReductor by including them in what corresponds to the symbolic preprocessing step in F_4. However, we encountered the difficulty that, while F_4 selects reductors solely based on the term they reduce, F_5 discards a reductor if its signature agrees with the reductee's, preventing a possible signature corruption. Just as in line 6, we imposed no restrictions on a reductor r_k if $\mathrm{index}(\mathcal{S}(r_k)) > i$, where i is the current step of the algorithm.

It is unfortunate that our hybrid implementation turned out not to be correct. Due to the limited scope of this thesis project, we did not have time to continue this work.

4 Conclusion

This chapter concludes the present report by briefly listing the contributions made and suggesting some topics for further work.

4.1 Contributions

The contributions we made are both theoretical and practical. On the practical side, we corrected minor errors in Faugère's pseudo code, and completed the – to our knowledge – first working public implementation of F_5, and one of only three working implementations at all (again, to our knowledge). While not designed for efficiency, it will doubtless be useful to anybody tackling the task of implementing F_5, and others who wish to experiment with the algorithm, for instance to check what kind of input systems are easily solved by F_5. In addition, we listed some experimental results, hinting that the version of F_5 presented in [14] can be considered as more or less naïve, and that Faugère's actual implementations are *a lot* more sophisticated. In sections 3.5 and 3.7, we suggested further improvements the F_5 algorithm and pointed out some problems encountered when attempting to merge F_4 and F_5 to an "$F_{4.5}$" algorithm.

On the theoretical side, we have slightly refined Faugère's main theorem in Theorem 3.21 and Corollary 3.23, and given the first full proof, completing the sketches published in [14]. Furthermore, we gave a more accessible account of the termination and correctness proofs of F_5, while supposing the correctness of certain optimizations (Conjecture 3.28). We also hope to have provided some insight into the inner workings of F_5, and stimulated further research on efficient Gröbner basis algorithms and their applications, in particular in cryptography.

4.2 Future work

Devising, implementing and analyzing Gröbner basis efficient algorithms is a non-trivial taks. While we made some contributions, a lot more work needs to be done.

First, a complete proof of F_5, if possible even in a less technical style providing some more insight, is needed. In view of applications, it is imperative to extend F_5 to accept non-regular – in particular, overdetermined! – sequences.

The author is convinced that an "F_4-like" (Faugère) version of F_5 would not only improve the efficiency of the algorithm, but would also be easier to understand and prove (and hence, improve). In this context, one should first go back to the paper by Möller et al. [21] and compare their algorithm to F_5. Their algorithm also has the nice

feature of enabling the use of syzygies known in addition to those in $\mathrm{PSyz}(F)$ and those caused by the reduction (i. e., the simplification rules).

Using the sample implementation by the author, an efficient public implementation would enable a fairer performance comparison of F_4 vs. F_5. Furthermore, the memory requirements could be analyzed, a task which cannot be fulfilled using an implementation in a high level language such as Magma.

To compare the efficiency of F_5, a more extensive theoretical and practical comparison with the pair minimization technique of Caboara, Kreuzer and Robbiano [9] or at least the criteria by Gebauer and Möller seems in order.

Bibliography

[1] Ars, G., J.-C. Faugère, H. Imai, M. Kawazoe and M. Sugita, *Comparison between XL and Gröbner basis algorithms*, in: P. J. Lee, editor, *ASIACRYPT 2004*, Lecture Notes in Computer Science **3329** (2004), pp. 338–353.

[2] Bardet, M., J.-C. Faugère and B. Salvy, *Complexity of Gröbner basis computation for semi-regular overdetermined sequences over* $\mathbb{F}_2$ *with solutions in* $\mathbb{F}_2$, rapport de recherche 5049, Institut National de Recherche en Informatique et en Automatique, Lorraine (2003).

[3] Bardet, M. T., "Etude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie," Ph.D. thesis, Université Paris 6 (2004), in French.
http://fgbrs.lip6.fr/~bardet/

[4] Becker, T., V. Weispfenning and H. Kredel, "Gröbner Bases," Springer-Verlag, New York, 1993.

[5] Brickenstein, M., "Neue Varianten zur Berechnung von Gröbnerbasen," Diplom thesis, Technische Universität Kaiserslautern, in German.
http://www-user.rhrk.uni-kl.de/~bricken/mathe.html

[6] Buchberger, B., "Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal (An algorithm for finding a basis for the residue class ring of a zero-dimensional polynomial ideal)," Ph.D. thesis, Universität Innsbruck (1965), in German.

[7] Buchberger, B., *A criterion for detecting unnecessary reductions in the construction of Gröbner bases*, in: *EUROSAM '79: Proceedings of the International Symposium on Symbolic and Algebraic Computation* (1979), pp. 3–21.

[8] Buchberger, B., *Gröbner bases: A short introduction for systems theorists*, in: R. Moreno-Diaz, B. Buchberger and J. Freire, editors, *Proceedings of EUROCAST 2001 (8th International Conference on Computer Aided Systems Theory – Formal Methods and Tools for Computer Science)*, Lecture Notes in Computer Science **2178** (2001).

[9] Caboara, M., M. Kreuzer and L. Robbiano, *Efficiently computing minimal sets of critical pairs*, Journal of Symbolic Computation **38** (2004), pp. 1169–1190.

[10] Computational Algebra Group, School of Mathematics and Statistics, University of Sydney, *Magma*, computational algebra software, version 2.11-14.
http://magma.maths.usyd.edu.au/

[11] Cox, D. R., J. Little and D. O'Shea, "Ideals, varieties, and algorithms," Undergraduate texts in mathematics, Springer-Verlag, 2000, second edition.

[12] Faugère, J.-C., *A new efficient algorithm for computing Gröbner bases (F_4)*, Journal of Pure and Applied Algebra **139** (1999), pp. 61–88.

[13] Faugère, J.-C., *Hfe challenge 1 broken in 96 hours*, announcement on `sci.crypt` (2002).

[14] Faugère, J.-C., *A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5)*, in: *ISSAC '02: Proceedings from the International Symposium on Symbolic and Algebraic Computation* (2002), pp. 75–83, revised version 1.2 available at the URL below.
http://www-calfor.lip6.fr/~jcf/Papers/@papers/f5.pdf

[15] Faugère, J.-C. and G. Ars, *An algebraic cryptanalysis of nonlinear filter generators using Gröbner bases*, Research report 4739, Institut National de Recherche en Informatique et en Automatique, Lorraine (2003).

[16] Faugère, J.-C. and A. Joux, *Algebraic cryptanalysis of Hidden Field Equation (HFE) cryptosystems using Gröbner bases*, in: D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, Lecture Notes in Computer Science **2729** (2003), pp. 44–60.

[17] Gebauer, R. and H. M. Möller, *On an installation of Buchberger's algorithm*, J. Symb. Comput. **6** (1988), pp. 275–286.

[18] Giovini, A., T. Mora, G. Niesi, L. Robbiano and C. Traverso, *"One sugar cube, please" or selection strategies in the Buchberger algorithm*, in: *ISSAC '91: Proceedings of the International Symposium on Symbolic and Algebraic Computation* (1991), pp. 49–54.
http://doi.acm.org/10.1145/120694.120701

[19] Lazard, D., *Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations*, in: *EUROCAL '83: Proceedings of the European Computer Algebra Conference on Computer Algebra* (1983), pp. 146–156.

[20] McKay, C. E., "An Analysis of Improvements to Buchberger's Algorithm for Gröbner Basis Computation," Master's thesis, University of Maryland, College Park (2004).
http://hdl.handle.net/1903/2119

[21] Möller, H. M., T. Mora and C. Traverso, *Gröbner bases computation using syzygies*, in: *ISSAC '92: Papers from the International Symposium on Symbolic and Algebraic Computation* (1992), pp. 320–328.
http://www.disi.unige.it/person/MoraF/publications.html

[22] Pearce, R., private communication (2005).
http://www.cecm.sfu.ca/~rpearcea/

[23] Schmidbauer, J., "Optimierung des Buchberger-Algorithmus im homogenen Fall," Diplom thesis, Universität Regensburg (2002), in German.
http://www.matha.mathematik.uni-dortmund.de/~kreuzer/projects.html

[24] Segers, A., "Algebraic Attacks from a Gröbner Basis Perspective," Master's thesis, Technische Universiteit Eindhoven (2004).
http://www.win.tue.nl/~bdeweger/students.html

[25] Steel, A., *Allan Steel's Gröbner basis timings page*, website (2004).
http://magma.maths.usyd.edu.au/users/allan/gb/

[26] Steel, A., private communication (2005).
http://magma.maths.usyd.edu.au/users/allan/

[27] Trinks, W., *Über B. Buchbergers Verfahren, Systeme algebraischer Gleichungen zu lösen*, Journal of Number Theory **10** (1978), pp. 475–488, in German.

[28] Wolf, C., "Hidden Field Equations (HFE) – Variations and Attacks," Diplom thesis, Universität Ulm (2002).
http://www.christopher-wolf.de/dpl/

[29] Wolf, C. and B. Preneel, *Taxonomy of public key schemes based on the problem of multivariate quadratic equations*, Technical Report 2005/077, Cryptology ePrint Archive (2005).
http://eprint.iacr.org/2005/077/

A Source code: F5

This chapter contains a sample implementation of Faugère's F_5 algorithm in the language of Magma v2.11-14 [10]. Due to space limitations, this listing contains only the central functions, and is intended to help the understanding of the algorithm. The full source code with all helper functions is available on the author's homepage:

http://www.cdc.informatik.tu-darmstadt.de/~stegers/

Readers new to Magma should note that, following the terminology of [11], Magma calls *terms* what we call *monomials* in this thesis and vice versa. For the sake of (local) consistency, this terminology is used in the source code as well.

```
/*
    For a procedure name, the identifier in brackets
    refers to the names J.-C. Faugere chose in his paper:

        "A new efficient algorithm for computing Gr"obner bases
        without reduction to zero ($F_5$)", ISSAC '02: Proceedings
        from the International Symposium on Symbolic and Algebraic
        Computation, ACM Press, 2002.
        Version 1.2 of the paper available at
        http://www-calfor.lip6.fr/~jcf/Papers/@papers/f5.pdf


During the algorithm, polynomials are augmented using so-called
signatures. These "labeled polynomials", or "rules", as Faug'ere calls
them, are implemented as tuples in Magma as follows: a labeled
polynomial f is a tuple  < t, i, p >, where
    t is a term,
    i the index of the i-th canonical basisvector F_i of the module P^m,
    p the polynomal,
    such that the signature of the labeled polynomial is S(p) = t*e_i.

Most of the time, these labeled polynomials will be elements of a unique
global array r, and be referenced by their indices in r, or r-indices for short.

Critical pairs are stored as tuples <l, u, j, v, k>, where
   l is the LCM of the critical pair
   u,v are terms, and
   j,k are (distinct) r-indices of labeled polynomials.

The variable R contains a sequence sequences of simplification rules,
similar to the array Rule in Faug'ere's paper. A simplification rule
in R[i] is a tupel <t,k>, where t is a term and k is an r-index of a
labeled polynomial r_k that has the signature t*e_i.
*/

import "critpairs.mag": NUMBER_OF_CRITERIA;

/*-------------------------------------------------+
|                                                  |
|                                                  |
|                 Main Algorithm                   |
|                                                  |
|                                                  |
+-------------------------------------------------*/
```

```
/*********************************************************

    AlgorithmF5

    returns a (not necessarily reduced) Groebner basis
    for the ideal generated by {f_i} union G_iplus1.

    Eventually, G_iplus1 is set to the resulting basis.


    Input:

        m_i_fi = <m,i,f_i>

        m           total number of input polynomials

        i           # of current global iteration

        f_i         polynomial to be added in this
                    iteration

        G_iplus1    sequence of r-indices of the labeled
                    polynomials forming the (non-reduced)
                    Gr"obner basis of the ideal generated
                    by f_{i+1}, ..., f_m

        G_iplus1Reduced: the unique reduced Gr"obner basis
                    of the ideal  generated by
                    f_{i+1}, ..., f_m

        rules       sequence of simplificationr rules

        statsPerCall: receives statistics generated during
                    this call, conforms with stats.mag



*********************************************************/
intrinsic AlgorithmF5(~m_i_fi::Tup,
    ~r::[],
    ~G_iplus1::[],
    ~G_iplus1Reduced::[],
    ~rules::[],
    ~statsPerCall::[])
{Compute a Groebner basis of the ideal <G_iplus1 cat [f_i]>.}

    m   := m_i_fi[1];
    i   := m_i_fi[2];
    f_i := m_i_fi[3];


    pol_ring := Parent(f_i);
    pair_univ := PairUniverse(pol_ring);
    rule_univ := RuleUniverse(pol_ring);


    // Statistics per call of F5
    // (data structure discussed in IncrementalF5)

    // Statistics per degree
    statsPerDeg := Stats_Init(m);

    r[i]:= <1, i, f_i / LeadingCoefficient(f_i)>;
    G_i := Append(G_iplus1,i);

    Stats_PolCreate(~statsPerCall,[i],~r);


    newPolIndices := [i];

    // Given G_{i+1}, compute the unique Groebner basis of the
    // ideal generated by G_{i+1}.
    procedure ReduceIntermediateBasis(~reduced,~r,~newPols)
```

```
    // Inefficient variant as in the paper (no reduction):
    //reduced cat:= [pol_ring| r[1][3]: l in newPols];

    // Efficient variant (using Magma's Reduce()):
    reduced := ReduceGroebnerBasis(reduced cat [pol_ring| r[1][3]: l in newPols]);

    // Honest variant (using interpreted reduction code, not Magma's):
    //reduced := ReduceGroebnerNaive(reduced cat [pol_ring| r[1][3]: l in newPols]);
end procedure;


// Collect head terms of G_{i+1}, ..., G_m
// to speed up reduction (phi) and reducibility test (psi)
heads := [ [car<Integers(),pol_ring,pol_ring>|
            <l,LM(G_iplus1Reduced[comp][l]),LC(G_iplus1Reduced[comp][l])>:
            l in [1..#G_iplus1Reduced[comp]]]: comp in [1..m]];

// Hand-written top reduction of a
// labeled polynomial x w.r.t. G_{i+1}
function phi(x)
    if x[3] ne Parent(x[3])!0 then
        LMx := LM(x[3]);
        LCx := LC(x[3]);
    end if;

    while (x[3] ne 0) and exists(g){h: h in heads[i+1] | IsDivisibleBy(LMx,h[2])} do
        y := x[3];
        x[3] -:= LCx/g[3] * (LMx div g[2]) * G_iplus1Reduced[i+1][g[1]];

        if x[3] ne Parent(x[3])!0 then
            LMx := LM(x[3]);
            LCx := LC(x[3]);
        end if;

        if GetAssertions() and not AssertReducedTo(y,x[3]) then
            print "x:",y;
            print "g:",g;
            print "x':",x[3];
            error  "not reduced! stopping.";
        end if;
    end while;

    return x;

    // very expensive test:
    assert x[3] eq NormalForm(x[3],G_iplus1Reduced[i+1]);
end function;



// For a labeled polynomial x = <t, k, p>, psi(x) is true iff the term t
// is top-reducible mod G_{k+1} and k<m, false otherwise (i.e. if k=m).
// In other words, psi(x) is true iff x is not normalized.
function psi(x)
    if x[2] eq m then
        return false;
    end if;

    return exists{h: h in heads[x[2]+1] | IsDivisibleBy(x[1],h[2])};
end function;

P := [ pair_univ | ];

for j in G_iplus1 do
    result := <-42>;
    CritPair(i,j,~r,i,~psi,~result);
    pair, is_pair,critUsed := Explode(result);

    assert TestCritPair(pair,is_pair,r,pol_ring);

    if is_pair then // check if the pair wasn't redundant
        Append(~P,pair);
        if GetAssertions() and not AssertPairNormalized(pair,r,m,pol_ring) then
```

```
            print "";
            print "critUsed:",critUsed;
            printf "Accepted non-normalized pair %o at spot 1\n", pair;
            PrintPairInfo(pair,~r,~G_iplus1Reduced,~heads,~psi);
            assert false;
        end if;
        Stats_PairCreate(~statsPerCall,pair);
    else
         if GetAssertions() and AssertPairNormalized(pair,r,m,pol_ring) then
            printf "Missed normalized pair %o at spot 1\n", pair;
            PrintPairInfo(pair,~r,~G_iplus1Reduced,~heads,~psi);
            assert false;
         end if;
         Stats_PairDiscard(~statsPerCall,critUsed);
    end if;
end for;

Sort(~P, ~HasLowerDegreeCritPair);

while not IsEmpty(P) do
    statsPerDeg := Stats_Init(m);

    d := DegreeCritPair(P[1]); // smallest occurring degree

    D := DegreeCritPair(P[#P]);
    P_d := [ pair_univ | pair: pair in P | DegreeCritPair(pair) eq d ];

    ChangeUniverse(~P_d,pair_univ);

    printf "\n----- Selecting new P_d...\n";
    printf "Size of G_%o so far: %o\n", i, #G_i;
    printf "Size of G_%o:        %o\n", i+1, #G_iplus1;
    printf "Size of G_%o reduced:%o\n", i+1, #G_iplus1Reduced[i+1];
    printf "Number of remaining critical pairs: %o\n",#P;
    printf "Minimal degree: %3o\n",d;
    printf "Maximal degree: %3o\n",D;
    printf "Selecting %o pair(s) of degree d=%o...\n\n",#P_d,d;


    P := P[#P_d+1..#P];
    assert #SequenceToSet(P) eq #P;

    //printf "P after removing P_%o: %o\n", d, P;
    //printf "P_%o: %o\n",d,P_d;


    print "Computing S-polynomials...";

    R_d := [];


    spols_redPairs := [* *];

    Spols(~G_i, ~P_d, i, ~r, ~rules,~spols_redPairs);

    spols := spols_redPairs[1];
    redundantPairs := spols_redPairs[2];


    printf "Number of S-polynomials before reduction: %o\n",#spols;
    Stats_PolCreate(~statsPerDeg,spols,~r);
    assert #P_d ge #spols;

    Stats_PolDiscard(~statsPerDeg,#P_d-#spols);

    phi_psi_stats := <phi, psi,statsPerDeg>;

    //ReductionF5(~spols, G_i, i, ~r, ~phi_psi_stats, ~rules);
    ReductionF5(~spols, newPolIndices, i, ~r, ~phi_psi_stats, ~rules);

    // Update statsPerDeg from the tuple
    statsPerDeg := phi_psi_stats[3];

    printf "Number of S-polynomials after reduction: %o\n\n", #spols;
```

```
    print "#R_d:",#R_d;

    R_d := R_d cat spols;

    printf "Adding critical pairs for reduced polynomials...\n", #R_d;

    nNewPairs := 0;


    for k in R_d do
        new_pairs := [pair_univ | ];

        for l in G_i do
            result := <-42>; // initialize variable receiving result
            CritPair(k, l, ~r, i, ~psi, ~result);
            pair, is_pair,critUsed := Explode(result);

            assert TestCritPair(pair,is_pair,r,pol_ring);

            if is_pair then
                Include(~new_pairs,pair_univ!pair);
                Stats_PairCreate(~statsPerDeg,pair);
                if GetAssertions() and not AssertPairNormalized(pair,r,m,pol_ring) then
                    printf "Accepted non-normalized pair %o\n", pair;
                    PrintPairInfo(pair,~r,~G_iplus1Reduced,~heads,~psi);
                    assert false;
                end if;
            else
                Stats_PairDiscard(~statsPerDeg,critUsed);

                if GetAssertions() and AssertPairNormalized(pair,r,m,pol_ring) then
                    printf "Missed normalized pair %o\n", pair;
                    PrintPairInfo(pair,~r,~G_iplus1Reduced,~heads,~psi);
                    assert false;
                end if;
            end if;
        end for;

        nNewPairs +:= #new_pairs;
        P cat:= new_pairs;

        // Add rule k to intermediate basis:
        Append(~G_i, k);
        Append(~newPolIndices,k);

    end for;

    printf "Number of new critical pairs:      %6o\n", nNewPairs;



    //printf "#G_i after adding s-polynomials: %o\n",#G_i;
    //printf "G_i after adding s-polynomials: %o\n",G_i;
    //print "r after one d-loop:",r;
    print "Sorting critical pairs...";
    Sort(~P, ~HasLowerDegreeCritPair);

    //printf "Statistics for degree %o:\n",d;
    //Stats_Print(statsPerDeg);
    Stats_Update(~statsPerCall,statsPerDeg);

end while;


G_iplus1 cat:= newPolIndices;

assert #SequenceToSet(G_iplus1) eq #G_iplus1;

// initalize for reduction
G_iplus1Reduced[i] := G_iplus1Reduced[i+1];


//Rereduce intermediate basis
ReduceIntermediateBasis(~G_iplus1Reduced[i],~r,~newPolIndices);
```

```
    print "Size of reduced basis:", #G_iplus1Reduced[i];

    //printf "\n\n\nStatistics for stage %o:\n\n",i;
    //Stats_Print(statsPerCall);

end intrinsic;



/***********************************************************

    Prereduce [no equivalent in Faug'ere's text]

    Mutual reduction of the input polynomials before
    starting F5. Corresponds to Magma's option ReduceInitial
    for the Buchberger algorithm.

    Input:

        F           sequence of polynomials

    Returns (in ~F):

        F           the reduced polynomials

***********************************************************/
intrinsic Prereduce(~F::[])
{Reduce the input before starting F5. Corresponds to Magma's option
 ReduceInitial for the Buchberger algorithm.}
    // Faster in most examples (exceptions: Cyclic n, Segers HFE)
   F := Reduce(F);
end intrinsic;



/***********************************************************

    IncrementalF5 [IncrementalF5]



    Input:

        F           sequence of nonzero homogeneous
                    polynomials (if F is not regular,
                    termination is not guaranteed)

        ensure_gb   If true, IncrementalF5 will check
                    using Magma's IsGroebner() that after
                    each step i, G_i is a Groebner basis.
                    Slows down the algorithm considerably.

    Returns:

        Gred        the unique reduced Groebner basis
                    of the ideal generated by F, identical
                    to the result of the call
                        GroebnerBasis(F);
                    in Magma

        G1          the non-reduced Groebner basis of F
                    as computed by F5. Consists of labeled
                    polynomials.

        is_gb       (for testing purposes) true if G1 is
                    a Groebner basis, false otherwise

        stats       a statistics object (see stats.mag)
                    summarizing this run of IncrementalF5


***********************************************************/
```

```
intrinsic IncrementalF5(F,ensure_gb) -> [],[],BoolElt,[]
{Outer routine of F5.}

    error if IsEmpty(F), "IncrementalF5: F must not be empty!";

    P := Parent(F[1]);
    rule_univ := RuleUniverse(P);

    Prereduce(~F);

    m := #F;

    // Initialize statistics
    stats := Stats_Init(m);



    rules := ResetRules(m,P);

    // Define the Macaulay matrix
    r := [rule_univ| ];

    r[m] := < 1, m, F[m] / LeadingCoefficient(F[m])>;

    printf "====================== Stage %o/%o ====================== \n",m,m;
    printf "Adding polynomial r[1].\n\n";

    G_iplus1 := [ m ];

    G_iplus1Reduced := [ [P| ]: i in [1..m+1] ];
    G_iplus1Reduced[m] := [ F[m] ];

    Stats_PolCreate(~stats,[m],~r);

    for i := m-1 to 1 by -1 do
        statsPerCall := Stats_Init(m);
        printf "====================== Stage %o/%o ====================== \n",i,m;

        m_i_fi := <m,i,F[i]>;
        AlgorithmF5(~m_i_fi,~r,~G_iplus1,~G_iplus1Reduced,~rules,~statsPerCall);

        G_rules := [r[l]: l in Sort(G_iplus1)];

        print "";


        print "";

        if ensure_gb then
            is_gb := IsGroebner([r[l][3]: l in G_iplus1]);

            if is_gb then
                printf "G_%o *is* a Groebner basis\n",i;
                printf "G_%o generates <f_%o, ..., f_m>: %o\n",i,i,
                Ideal([r[l][3]: l in G_iplus1]) eq Ideal([F[l]: l in [i..m]]);
                printf "Size of basis is %o.\n",#G_iplus1;
                printf "Size of reduced basis is %o.\n", #G_iplus1Reduced[i];
            else
                printf "G_%o is *not* a Groebner basis. Size is %o.\n",i,#G_iplus1;

                if ensure_gb then
                printf "Aborted after step i = %o and returning G_i\n",i;
                return G_iplus1Reduced[i],G_rules,is_gb;
                end if;
            end if;
            print "";
        else
            is_gb := true;
        end if;
        //printf "G_%o = %o\n\n",i,G_iplus1;

        Stats_Update(~stats,statsPerCall);
    end for;
```

```
    print "Statistics for all stages:";
    Stats_Print(stats);

    return G_iplus1Reduced[1],G_rules,is_gb,stats;
end intrinsic;



/*********************************************************

    F5opt

    Wrapper for the F5 algorithm, e.g. for use in a
    comparison with other algorithms.


    Input:

        F           sequence of nonzero homogeneous
                    polynomials (if F is not regular,
                    termination is not guaranteed)

        ensure_gb   If true, IncrementalF5 will check
                    using Magma's IsGroebner() that after
                    each step i, G_i is a Groebner basis.
                    Slows down the algorithm considerably.

    Returns:

        Gred        the unique reduced Groebner basis
                    of the ideal generated by F, identical
                    to the result of the call
                        GroebnerBasis(F);
                    in Magma

        stats       a statistics object (see stats.mag)
                    summarizing this run of IncrementalF5


        name        a string identifying the algorithm,
                    currently simply "F5"

*********************************************************/
intrinsic F5opt(F::[],ensure_gb::BoolElt) -> [],[],MonStgElt
{Faugere's F5 algorithm}


    error if IsEmpty(F), "F must not be empty!";

    P := Parent(F[1]);

    homogenized := false;
    isHomogeneous := IsHomogeneousSystem(F);

    // Make sure we didn't forget to homogenize
    if not isHomogeneous[1]  then
        read answer,"System is not homogeneous! Homogenize first y/n? [y]";
        if answer ne "n" then
            homogenized := true;
            F,P := HomogenizeExample(F,P);
        end if;
    end if;

    Gpols,Grules,b,stats := IncrementalF5(F,ensure_gb);
    if isHomogeneous[1] then
        print "Input system was homogeneous.";
    else
        printf "Input system was *not* homogeneous,";

        if homogenized then
            print " it was homogenized first.";
        else
            print " but it was not homogenized. You're lucky it terminated at all!";
```

```
        end if;
    end if;

    if ensure_gb then
        same_ideal := Ideal(F) eq Ideal(Gpols);
        print "Generates same ideal:", same_ideal;
    else
        print "Did not check if G_1 is actually a Groebner basis.";
    end if;

    return Gpols,stats, "F5";
end intrinsic;



/***********************************************************

    F5


    Wrapper for the F5 algorithm, useful for calls from
    the Magma console.


    Input:

        F           sequence of nonzero homogeneous
                    polynomials (if F is not regular,
                    termination is not guaranteed)

    Returns:

        Gred        the unique reduced Groebner basis
                    of the ideal generated by F, identical
                    to the result of the call
                        GroebnerBasis(F);
                    in Magma


***********************************************************/
intrinsic F5(F::[]) -> [],[],MonStgElt
    {Faugere's F5 algorithm, not checking that G_i is a Groebner basis after each step i.}

    return F5opt(F,false);
end intrinsic;



/***********************************************************

    F5ensure


    Wrapper for the F5 algorithm, useful for calls from
    the Magma console when debugging the algorithm.


    Input:

        F           sequence of nonzero homogeneous
                    polynomials (if F is not regular,
                    termination is not guaranteed)

    Returns:

        Gred        the unique reduced Groebner basis
                    of the ideal generated by F, identical
                    to the result of the call
                        GroebnerBasis(F);
                    in Magma


***********************************************************/
```

```
intrinsic F5ensure(F::[]) -> [],[],MonStgElt
    {Faugere's F5 algorithm, checking that G_i is a Groebner basis after each step i.}

    return F5opt(F,true);
end intrinsic;
/*------------------------------------------------+
|                                                 |
|                                                 |
|            Computation of Critical Pairs        |
|                  and S-Polynomials              |
|                                                 |
|                                                 |
+------------------------------------------------*/



/*********************************************************

    Criteria used to detect non-normalized critical pairs

*********************************************************/

    // Both components have index <= i
    CRITERION_ONE   := 1; //

    // Larger component is not normalized
    CRITERION_TWO   := 2;

    // Smaller component is not normalized
    CRITERION_THREE := 3;


    // Number of criteria above, used for stats
    NUMBER_OF_CRITERIA := 3;



/*********************************************************

  Rewrite [Rewritten]


  Simplifies a given a pair (u,r[k]) using a list of
  simplification rules.

  Input:

        u:          term
        k:          r-index
        iter:       # of iteration in F5
        r:          global list of labeled polynomials
        rules:      global list of simplification rules
        result:     tupel to hold the result

  Returns (in ~result):
    A tupel of the form <u',k'>, where u is a term
    and k' the highest r-index with r[k'][2]=iter and
    r[k'][1] a divisor of u*r[k']. In particular
    k=k' if the given pair could not be simplified.

*********************************************************/

intrinsic Rewrite(u::RngMPolElt, k::RngIntElt, iter::RngIntElt, ~r::[], ~rules::[], ~result::Tup)
{Simplifies a given a pair (u,r) using the given list of rules.}

    assert LC(u) eq 1;

    t := LM(r[k][1]);
    i := r[k][2];

    assert LC(t) eq 1;
```

```
    for j in [1..#rules[i]] do

        t_j := rules[i,j][1];
        k_j := rules[i,j][2];
        assert LC(t_j) eq 1;

        // assert entry rules[i,j] is correct:
        assert LM(r[k_j][1]) eq t_j;
        assert r[k_j][2] eq i;


        if IsDivisibleBy(u*t,t_j) then
            result := < (u*t)div t_j, k_j >;
            return;
        end if;
    end for;

    result := < u, k >; // could not be simplified
end intrinsic;



/***********************************************************
  IsRewritable [Rewritten?]

  Checks if a term u can be rewritten using a given
  labeled polynomial r[k].

  Input:
        u:          term
        k:          r-index
        iter:       # of global iteration
        r:          global list of labeled polynomials
        rules:      global list of rules

  Returns (in ~result):
        True if the term u can be rewritten using the
        labeled polynomial r_k, false otherwise.

***********************************************************/
intrinsic IsRewritable(u::RngMPolElt,k::RngIntElt,iter::RngIntElt,~r::[],~rules::[],~result::BoolElt)
{Checks if a term u can be rewritten using a given labeled polynomial r[k].}

    subresult := <-42>;
    Rewrite(u,k,iter,~r,~rules,~subresult);
    term, l := Explode(subresult);

    assert LC(term) eq 1;

    result := l ne k;

end intrinsic;



/***********************************************************
    CritPair

    Given two r-indices i,j, check if one of the two
    critical pairs (r_i,r_j), (r_j,r_i) is normalized.
    If so, assemble and return it.


    Input:

        i:          r-index of a normalized labeled
                    polynomial
        j:          r-index of a normalized  labeled
                    polynomial
        r:          "global" array of labeled polynomials
        k:          current iteration in IncrementalF5
        psi:        function that returns true for a
                    labeled polynomial x iff x is
```

```
                top-reducible mod G_{l+1}, where
                l = x[2]

    Returns (in ~result):

        result      a list [* p, is_pair, crit *]

        is_pair:    is true if this critical pair is
                    normalized, false otherwise.

        p:          If is_pair is true, p is a critical
                    pair, i.e. a tuple <t,u,k,v,l>,
                    where t is the LCM of the HTs of
                    the polynomials indexed by k,l,
                    u is t div HT(r[k]), v is t div
                    HT(r[l]). {k,l} is {i,j} (not
                    necessarily [k,l]=[i,j], as the
                    components may be swapped (cf.
                    Faug'ere's  Def. 3). The pair is
                    guaranteed to be normalized.

        crit:       number of the criterion used to
                    identify the pair as redundant
                    (see constants above)

**********************************************************/
intrinsic CritPair(i::RngIntElt, j::RngIntElt, ~r::[], iter::RngIntElt, ~psi::Program, ~result::Tup)
{Compute critical pair for two rules,  if necessary.}

    pol_ring := Parent(r[i][3]);
    rule_univ := RuleUniverse(pol_ring);
    pair_univ := PairUniverse(pol_ring);

    t := LCMCritPair(r[i],r[j]);

    u_1 := pol_ring!t div pol_ring!LM(r[i][3]);
    u_2 := pol_ring!t div pol_ring!LM(r[j][3]);

    if SignatureLess(<u_1*r[i][1],r[i][2]>,<u_2*r[j][1],r[j][2]>) lt 0 then
        // swap rules
        tmp  := i;
        i := j;
        j := tmp;

        tmp := u_1;
        u_1 := u_2;
        u_2 := tmp;
    end if;

    t_1 := LM(r[i][1]);
    t_2 := LM(r[j][1]);


    k_1 := r[i][2];
    k_2 := r[j][2];


    function Criterion1()
        if k_1 gt iter then
            //TODO: can't happen!
            //printf "crit (1) ruled out pair %o.\n", [i,j];
            return false;
        else
            return true;
        end if;
    end function;


    function Criterion2()
        if psi(<u_1*t_1,k_1,1>) then
            //printf "crit (2) ruled out pair %o\n", [i,j];
            return false;
        else
            return true;
```

```
        end if;
    end function;


    function Criterion3()
        if psi(<u_2*t_2,k_2,1>) then  //DELTA_Faugere
            //printf "crit (3) ruled out pair %o\n", [i,j];
            return false;
        else
            return true;
        end if;
    end function;


    pair := pair_univ!<t, u_1, i, u_2, j>;

    if not Criterion1() then
        result := < pair, false, CRITERION_ONE >;
        return;
    elif not Criterion2() then
        result := < pair, false, CRITERION_TWO >;
        return;
    elif not Criterion3() then
        result := < pair, false, CRITERION_THREE >;
        return;
    else
        //printf "New crit pair: <%o, %o, r_%o, %o, r_%o>\n", t,u_1,i,u_2,j;

        result := < pair, true, 0 >;
        return;
    end if;

end intrinsic;



/***********************************************************

   Spols [Spol]

   Calculate S-polynomials for a sequence of
   critical pairs, eliminating redundant
   S-polynomials using simplification rules.

   Input:

        G_i:        current intermediate Groebner
                    basis of <f_i,...,f_m>

        critpairs:  sequence of critical pairs

        iter:       # of current global iteration

        r:          "global" list of labeled polynomials,
                    receives any S-polynomials created

        rules:      "global" list of simplification rules,
                    is updated if any S-polynomials are
                    created

        F_dropped:  tuple, receives the return value


   Returns (in ~F_dropped):

        F_dropped[1]: sequence of the r-indices of the
                    S-polynomials for the given critical
                    pairs except those that were detected
                    as unnecessary

        F_dropped[2]: sequence of those critical pairs
                    in critpairs for which the S-polynomial
                    was dropped, i.e., not inserted into r
```

```
**********************************************************/
intrinsic Spols(~G_i::[],~critpairs::[], iter::RngIntElt, ~r::[], ~rules::[], ~F_dropped)
{Calculate non-redundant S-polynomials for a sequence of critical pairs.}

    pol_ring := Parent(r[iter][3]);

    rule_univ := RuleUniverse(pol_ring);

    // # of S-polynomials that are 0
    nZero := 0;

    F := [ ];

    droppedL := [];

    print "Spols: Total number of pairs to examine:", #critpairs;
    for pair in critpairs do

        u := pair[2];
        v := pair[4];

        il := pair[3];
        jl := pair[5];

        assert il ne jl;

        // ** Determine if we need the S-polynomial **

        // Criterion 1: is none of the polys zero?
        needSpol := (r[il][3] ne 0) and (r[jl][3] ne 0);

        lc_il := LC(u*r[il][3]);
        lc_jl := LC(v*r[jl][3]);

        sp := u*r[il][3]-  lc_il / lc_jl * v*r[jl][3];

        if sp eq 0 then
            // Skip zeroes
            //printf "\nS-Polynomial of pair %o is 0, skipping\n",pair;
            //printf "r_%o = %o\n", il, r[il];
            //printf "r_%o = %o\n", jl, r[jl];
            nZero +:= 1;
            continue;
        end if;

        // Criterion 2: Can the left summand be rewritten?
        if needSpol then
            result := false;
            IsRewritable(u, il, iter, ~r, ~rules, ~result);
            needSpol := not result;
        end if;

        // Criterion 3: Can the right summand be rewritten?
        if needSpol then
            IsRewritable(v, jl, iter, ~r, ~rules, ~result);
            needSpol := not result;
        end if;


        if needSpol then

            N := #r + 1; // "increment" N (it's actually not global)

            assert AssertReducedTo(u*r[il][3],sp);
            assert AssertReducedTo(v*r[jl][3],sp);

            r[N] := rule_univ!<u * r[il][1], r[il][2], sp>;

            AddRule(~rules,~r,N,pol_ring);

            Append(~F,N);

        else
```

```
            droppedL cat:= [pair];


        end if;
    end for;

    cmpFunc := -42; //initialize with dummy value
    IndexSignatureLess(~r,~cmpFunc);
    Sort(~F, cmpFunc);

    if GetAssertions() then
        degs := {TotalDegree(r[1][3]): l in F};
        error if #degs gt 1, "S-polynomials have different degrees!:",degs;
    end if;

    F_dropped := [* F, droppedL *];
end intrinsic;
/*-------------------------------------------------+
|                                                  |
|                                                  |
|          Reduction of Polynomial Sequences       |
|                                                  |
|                                                  |
|                                                  |
+-------------------------------------------------*/



/**************************************************

    FindReductor [IsReducible]


    Input:

        k0_k:    a tuple <k0,k> of integers (see below)

        k0:     r-index of labeled polynomial to reduce

        k:      # of global iteration

        G:      sequence of r-indices of polynomials w.r.t.
                which r[k0] is to be reduced

        r:      "global" list of polynomials

        psi:    function that returns true iff a rule
                is top-reducible w.r.t. G_{i+1}

        rules:  "global" list of simplification rules

        result: variable to receive return value


    Returns (in ~result):

        A tupel < red, is_top_red >.

        is_top_red: false if the polynomial of r[k0] cannot
                be top-reduced by the polynomials specified
                by G (modulo optimizations)

        red:    a reductor of r[k0] if r[k0] can be
                top-reduced (modulo optimizations)


**************************************************/
intrinsic FindReductor(k0_k::Tup, ~G::[],  ~r::[], ~phi_psi_stats::Tup, ~rules::[], ~result)
{Find a reductor for a given labeled polynomial.}

    k0 := k0_k[1];
    k := k0_k[2];
```

```
    psi := phi_psi_stats[2];

    pol_ring := Parent(r[k0][3]);

    rule_univ := RuleUniverse(pol_ring);

    lt_0 := pol_ring!LeadingMonomial(r[k0][3]);

    for j in G do

        lt_i := pol_ring!LeadingMonomial(r[j][3]);
        k_j := r[j][2];

        if not IsDivisibleBy(lt_0,lt_i) then
            // discard reductor by criterion (d)
            continue;
        else
            u := lt_0 div lt_i;
            ut := u * r[j][1];

            if (ut eq r[k0][1]) and (r[j][2] eq r[k0][2]) then
                // discard reductor by criterion (d)
                continue;
            end if;

            canBeRewritten := false;

            IsRewritable(u,j,k,~r,~rules,~canBeRewritten);

            if canBeRewritten then
                // discard reductor by criterion (c)
                continue;

            elif psi(<ut,k_j,1>) then
                // discard reductor by criterion (b)
                continue;
            else
                // conditions (a) through (d) are satisfied

               result := < j,true >;
               return;
            end if;
        end if;
    end for;

    // no reductor found or all discarded
    result := < 0,false >;

end intrinsic;



 /*****************************************************

    TopReduction [TopReduction]


    Performs a top-reduction using a normalized
    reductor on a given labeled polynomial,
    if possible. If no normalized reductor can be
    found, the polynomial is divided by its leading
    coefficient. If the reductor has a larger signature
    than the reductee, the reductum is added as a new
    labeled polynomial, and the list of simpification
    rules is updated.

    Input:

        k0_k:       tuple <k0,k>, see below

        k0:         the index of a polynomial in r
                    that is to be top-reduced

        k:          # of global iteration
```

```
        G:          list of r-indices of labeled polynomials
                    w.r.t. which to reduce r[k0]

        r:          list of labeled polynomials,
                    might be updated

        phi_psi_stats: tuple <phi,psi,stats> (see below)


        phi:        function that, given a labeled
                    polynomial <t,j,p>, returns
                    <t,j,NormalForm(p,G_{k+1})>

        psi:        function that returns true iff a
                    labeled polynomial <t,j,p> is
                    top-reducible w.r.t. G_{j+1}

        stats       a statistics object (see stats.mag)
                    that might be updated

        R:          simplification rules, might be
                    updated

        result:     variable to receive return value



    Result (in ~result):

        result = [* no_red_to_zero, h, red_list *]

        no_red_to_zero: false iff the r[k0] was reduced
                    to zero

        red_list:   Sequence of r-indices of polynomials
                    to be reduced, possibly empty;
                    unspecified if no_red_to_zero is false


        h:          r-index of any rules for which no normalized
                    reductor could be found, and hence may be
                    declared done by Reduction.
                    If no_red_to_zero is false or red_list is
                    empty, h is unspecified.


**************************************************/
intrinsic TopReduction(k0_k::Tup, ~G::[], ~r::[], ~phi_psi_stats::Tup, ~R::[], ~result)
{Reduces a polynomial w.r.t. a list of polynomials. Adds new rules as it sees fit.}

    k0 := k0_k[1];
    k  := k0_k[2];

    error if k0 gt #r, "TopReduction: Rule index k0 is invalid!";

    pol_ring := Parent(r[k0][3]);

    old := r[k0];



    if r[k0,3] eq Zero(pol_ring) then
        printf "Warning, r[%4o] is 0! Input is not a regular sequence. Stage: %o\n",k0,k;

        Stats_RedToZero(~phi_psi_stats[3],k);

        result := [* false, 0,[ ] *];
        return;
    end if;


    // k1 corresponds to the index of r' in Faugere's paper
    FindReductor(<k0,k>, ~G, ~r, ~phi_psi_stats, ~R, ~result);
```

```
    k1, top_reducible := Explode(result);

    lc_0 := LeadingCoefficient(r[k0][3]);

    if (not top_reducible) then

        // lc_0 is nonzero (see if-clause above)
        normalized_rule := <r[k0][1], r[k0][2], r[k0][3] div lc_0>;
        r[k0] := normalized_rule;


        result := [*  true, k0, [ ] *];
        return;

    else

        lt_0 := LeadingMonomial(r[k0][3]);
        lt_1 := LeadingMonomial(r[k1][3]);

        lc_1 := LeadingCoefficient(r[k1][3]);

        u := pol_ring!lt_0 div pol_ring!lt_1;

        new_sig := <u * r[k1][1], r[k1][2]>;

        if SignatureLess(new_sig, <r[k0][1],r[k0][2]>) lt 0 then

            r[k0][3] := r[k0][3] - lc_0 / lc_1 * u * r[k1][3];

            assert AssertReducedTo(old[3],r[k0][3]);

            result := [*  true, 0, [k0] *];
            return;
        else

            N := #r + 1;

            r[N] := <new_sig[1], new_sig[2], u*r[k1][3] - lc_1/lc_0*r[k0][3]>;

            if r[N][3] eq 0 then //DEBUG
                //print "TopRed: Unexpected reduction to zero occurred!"; //TODO what's so unexp. here?
                Stats_RedToZero(~phi_psi_stats[3],k);
                result := [* false, 0, [] *];
                return;
            end if;


            AddRule(~R,~r,N,pol_ring);
            Stats_PolCreate(~phi_psi_stats[3],[N],~r);

            assert AssertReducedTo(r[k0][3],r[N][3]);

            result := [* true, 0, [N,k0] *];
            // (Segers says k0 is superfluous, but if left out F5 loops)

            return;

        end if;
    end if;

end intrinsic;


/*****************************************************

    Reduction [Reduction]


    Input:


        todo:       sequence of t-indices of representing
                    polynomials to reduce
```

```
        G:          SeqEnum of indices of r representing
                    polynomials w.r.t. which to reduce

        r           set of all labeled polynomials known
                    (will possibly be updated)

        k           # of global iteration

        phi_psi_stats: tuple <phi,psi,stats> (see below)


        phi:        function that, given a labeled
                    polynomial <t,j,p>, returns
                    <t,j,NormalForm(p,G_{k+1})>

        psi:        function that returns true iff a
                    labeled polynomial <t,j,p> is
                    top-reducible w.r.t. G_{j+1}

        stats       a statistics object (see stats.mag)
                    that might be updated

        rules:      simplification rules, might be
                    updated


   Returns (in ~todo):

            sequence of r-indices of labeled polynomials
            to be included in the Groebner basis


**************************************************/
intrinsic ReductionF5(~todo::[], G::[], k::RngIntElt, ~r::[], ~phi_psi_stats::Tup, ~rules::[])
{Reduction step in F5.}

        done := [];
        pol_ring := Parent(r[k][3]);

        // list of reduced S-polynomials known to be in NF w.r.t. G_i+1, currently
        inNFwrtG_iplus1 := [Integers()| ];
        nPolsCreated := 0;

        while (not IsEmpty(todo)) do

                //TODO could be made more efficient. -- Isn't this redundant?!
                IndexRemoveDoubles(~todo,~r,pol_ring,~todo);

                cmpFunc := -42; //initialize variable
                IndexSignatureLess(~r,~cmpFunc);
                Sort(~todo,cmpFunc);

                size := #todo;


                h := todo[1];  // r-index of polynomial with minimal signature
                Remove(~todo,1);

                // Moved here from TopRed to be able to avoid
                // calling phi more often than necessary
                pos := 0;
                BinSearch(~inNFwrtG_iplus1,h,~pos); //TODO #inNFwrtG_iplus1 <= 1, so this is unnecessary?!!
                if pos lt 1 then
                        // r[h] was not reduced by phi yet or has changed since then
                        r[h] := phi_psi_stats[1](r[h]);
                    else
                        nPolsCreated +:= 1;
                end if;


                //DELTA_Pearce: one could pass todo as well, claims Pearce

                GcatDone := done cat G;
```

```
                result := [];


                TopReduction(<h,k>, ~GcatDone, ~r,~phi_psi_stats, ~rules,~result);
                is_reg_seq, h1, todo1 := Explode(result);


                if #todo1 gt 1 then
                    // h was not reduced, so we don't have to call phi(h)
                    // the next time h is treated
                    IncludeSorted(~inNFwrtG_iplus1,h);
                else
                    ExcludeSorted(~inNFwrtG_iplus1,h);
                end if;


                if is_reg_seq then
                    if IsEmpty(todo1) then // Faug'ere doesn't need this 'if' as he
                        Append(~done,h1);  // just returns the empty set for h
                    else
                        todo := todo cat todo1;
                    end if;
                end if;

                size := #todo;
        end while;

        //print "Number of polynomials created during reduction:",nPolsCreated;
        todo := done;

end intrinsic;
```

B Polynomial Systems

This chapter lists the polynomial systems used for the benchmarks in section 3.6. When giving the polynomial ring of an example, we sort the variables in decreasing order (for instance, $y < x$ in $R[x, y]$). We use the notation $\mathbb{F}_q$ for the finite field with q elements. All examples were computed in the grevlex order (see Example 2.6). Those that contained non-homogeneous polynomials were homogenized, whereas the reference given might list only the original system.

Note that all these examples can be downloaded as Magma source files from the authors web page at

http://www.cdc.informatik.tu-darmstadt.de/~stegers/ .

Weispfenning94
Ring: $\mathbb{F}_{7583}[x, y, z, h]$
Source: http://www.symbolicdata.org/
Regular sequence: no
Polynomial system: $y^4 + xy^2z + x^2h^2 - 2xyh^2 + y^2h^2 + z^2h^2$, $xy^4 + yz^4 - 2x^2yh^2 - 3h^5$, $-x^3y^2 + xyz^3 + y^4h + xy^2zh - 2xyh^3$

Buchberger 87
Ring: $\mathbb{F}_{7583}[h, r, t, x, y, z]$
Source: http://www.symbolicdata.org
Regular sequence: no
Polynomial system: $hx - rt$, $hz - r^2$, $h^2y - rt^2$

Eco6
Ring: $\mathbb{F}_{7583}[x_1, x_2, x_3, x_4, x_5, x_6, h]$
Source: http://www.symbolicdata.org/
Regular sequence: no
Polynomial system: $x_1 + x_2 + x_3 + x_4 + x_5 + h$, $x_5x_6 - 5h^2$, $x_1x_5x_6 + x_4x_6h - 4h^3$, $x_1x_4x_6 + x_2x_5x_6 + x_3x_6h - 3h^3$, $x_1x_3x_6 + x_2x_4x_6 + x_3x_5x_6 + x_2x_6h - 2h^3$, $x_1x_2x_6 + x_2x_3x_6 + x_3x_4x_6 + x_4x_5x_6 + x_1x_6h - h^3$

Segers' HFE System
Ring: $\mathbb{F}_2[x_1, \ldots, x_7, h]$
Source: Reference [24]

Regular sequence: no
Polynomial system: $x_1x_2 + x_2^2 + x_3^2 + x_2x_4 + x_3x_4 + x_4^2 + x_1x_5 + x_2x_5 + x_4x_5 + x_5^2 + x_1x_6 + x_2x_6 + x_6^2 + x_6x_7 + x_6h + x_7h$,
$x_1^2 + x_1x_2 + x_1x_3 + x_2x_3 + x_1x_4 + x_4^2 + x_1x_5 + x_2x_5 + x_3x_5 + x_4x_5 + x_2x_6 + x_5x_6 + x_1x_7 + x_2x_7 + x_3x_7 + x_4x_7 + x_5x_7 + x_6x_7 + x_1h + x_3h + x_5h + h^2$,
$x_1x_2 + x_1x_3 + x_3^2 + x_1x_4 + x_2x_4 + x_4^2 + x_4x_5 + x_1x_6 + x_3x_6 + x_2x_7 + x_3x_7 + x_4x_7 + x_5x_7 + x_6x_7 + x_1h + x_3h + x_4h + x_5h + x_6h + x_7h$,
$x_1^2 + x_2^2 + x_1x_3 + x_1x_4 + x_3x_4 + x_1x_5 + x_2x_5 + x_3x_5 + x_4x_5 + x_1x_6 + x_1x_7 + x_2x_7 + x_4x_7 + x_6x_7 + x_7^2 + x_1h + x_2h + x_3h + x_4h + x_5h + x_7h + h^2$,
$x_1^2 + x_2x_3 + x_2x_4 + x_3x_4 + x_1x_5 + x_3x_5 + x_4x_5 + x_5^2 + x_4x_6 + x_5x_6 + x_4x_7 + x_6x_7 + x_7^2 + x_2h + x_3h + x_5h + x_6h + x_7h$,
$x_1^2+x_1x_2+x_2x_5+x_5^2+x_4x_6+x_5x_6+x_6^2+x_5x_7+x_6x_7+x_1h+x_2h+x_4h+x_5h+x_6h+x_7h+h^2$,
$x_1x_3 + x_2x_3 + x_1x_4 + x_3x_4 + x_4^2 + x_1x_5 + x_2x_5 + x_4x_5 + x_5^2 + x_2x_6 + x_6^2 + x_1x_7 + x_2x_7 + x_5x_7 + x_7^2 + x_2h + x_5h + x_6h$,
$x_1^2 + x_1h$, $x_2^2 + x_2h$, $x_3^2 + x_3h$, $x_4^2 + x_4h$, $x_5^2 + x_5h$, $x_6^2 + x_6h$, $x_7^2 + x_7h$

Faugere 2002 / Moeller-Mora-Traverso 1992
Ring: $\mathbb{Q}[x, y, z, t]$
Source: References [21, 14]
Regular sequence: yes
Polynomial system: $yz^3 - x^2t^2$, $xz^2 - y^2t$, $x^2y - z^2t$

Uteshev-Bikker
Ring: $\mathbb{F}_{7583}[x, y, z, t, h]$
Source: http://fgbrs.lip6.fr/jcf/Benchs/
Regular sequence: yes
Polynomial system:
$x^2 + xy + y^2 - 2xz - 4yz + 3z^2 - 3xt + 2yt + t^2 - 3xh - 2yh + 3zh - 2th - 2h^2$,
$2x^2 - xy + y^2 - xz - yz - 6z^2 - xt + yt - 5zt - 3t^2 - 5xh + yh + 5zh + 2th + 5h^2$,
$x^3 + y^3 - x^2z + xyz - 5y^2z - 5xz^2 + 7yz^2 - 3z^3 + xyt - 5z^2t + xt^2 + 2t^3 + x^2h - 3xyh - y^2h + 2xzh + 2z^2h - 3xth - 2zth - 3t^2h - xh^2 + yh^2 + 11zh^2 - 2th^2 - 3h^3$,
$-x^3+6x^2y-12xy^2+6y^3-x^2z-4xyz+6y^2z+5xz^2+4yz^2+15z^3+6xyt-7y^2t-xzt+11xt^2+4t^3+3x^2h+2xyh+2y^2h-z^2h+2yth-zth+5t^2h-35xh^2-14yh^2+4zh^2-10th^2-15h^3$

Liu
Ring: $\mathbb{F}_2[x, y, z, t, a, h]$
Source: http://fgbrs.lip6.fr/jcf/Benchs/
Regular sequence: yes
Polynomial system: $yz - yt - xh + ah$, $zt - zx - yh + ah$, $tx - yt - zh + ah$,
$xy - zx - th + ah$

Lichtblau
Ring: $\mathbb{Q}[t, x, y]$
Source: http://fgbrs.lip6.fr/jcf/Benchs/
Regular sequence: yes
Polynomial system:
$x - 110t^2 + 495t^3 - 1320t^4 + 2772t^5 - 5082t^6 + 7590t^7 - 8085t^8 + 5555t^9 - 2189t^{10} + 374t^{11}$,
$y - 22t + 110t^2 - 330t^3 + 1848t^5 - 3696t^6 + 3300t^7 - 1650t^8 + 550t^9 - 88t^{10} - 22t^{11}$

Gerdt93
Ring: $\mathbb{F}_{7583}[h, l, s, x, y, z]$
Source: http://www.symbolicdata.org/
Regular sequence: false
Polynomial system: $hl - l^2 - 4ls + hy$, $h^2s - 6ls^2 + h^2z$, $xh^2 - l^2s - h^3$

Sym3-3
Ring: $\mathbb{F}_{7583}[h, x, y, z]$
Source: http://www.symbolicdata.org/
Regular sequence: yes
Polynomial system: $yz^3 + h^3x - 2h^4$, $x^3z + h^3y - 2h^4$, $xy^3 + h^3z - 2h^4$

Trinks
Ring: $\mathbb{F}_{7583}[w, p, z, t, s, b, h]$
Source: http://www.symbolicdata.org/
Regular sequence: no
Polynomial system: $35p + 40z + 25t - 27s$, $45p + 35s - 165b - 36h$, $-11sb + 3b^2 + 99wh$,
$25ps - 165b^2 + 15wh + 30zh - 18th$, $15pt + 20zs - 9wh$, $-11b^3 + wph + 2zth$

Hairer1
Ring: $\mathbb{F}_{7583}[c_2, c_3, b_3, b_2, b_1, a_{21}, a_{32}, a_{31}, h]$
Source: http://www.symbolicdata.org/
Regular sequence: yes
Polynomial system: $b_3 + b_2 + b_1 - h$, $c_3 - a_{32} - a_{31}$, $c_2 - a_{21}$, $2c_3b_3 + 2c_2b_2 - h^2$,
$6c_2b_3a_{32} - h^3$, $3c_3^2b_3 + 3c_2^2b_2 - h^3$

f633
Ring: $\mathbb{F}_{7583}[U_6, U_5, U_4, U_3, U_2, u_6, u_5, u_4, u_3, u_2, h]$
Source: http://fgbrs.lip6.fr/jcf/Benchs/
Regular sequence: no
Polynomial system: $2u_6 + 2u_5 + 2u_4 + 2u_3 + 2u_2 + h$, $2U_6 + 2U_5 + 2U_4 + 2U_3 + 2U_2 + h$,

$4U_5u_6 + 4U_4u_6 + 4U_3u_6 + 4U_2u_6 - 4U_6u_5 + 4U_4u_5 + 4U_3u_5 + 4U_2u_5 - 4U_6u_4 - 4U_5u_4 + 4U_3u_4 + 4U_2u_4 - 4U_6u_3 - 4U_5u_3 - 4U_4u_3 + 4U_2u_3 - 4U_6u_2 - 4U_5u_2 - 4U_4u_2 - 4U_3u_2 + 2u_6h + 2u_5h + 2u_4h + 2u_3h + 2u_2h + h^2,$
$-4U_5u_6 - 4U_4u_6 - 4U_3u_6 - 4U_2u_6 + 4U_6u_5 - 4U_4u_5 - 4U_3u_5 - 4U_2u_5 + 4U_6u_4 + 4U_5u_4 - 4U_3u_4 - 4U_2u_4 + 4U_6u_3 + 4U_5u_3 + 4U_4u_3 - 4U_2u_3 + 4U_6u_2 + 4U_5u_2 + 4U_4u_2 + 4U_3u_2 + 2U_6h + 2U_5h + 2U_4h + 2U_3h + 2U_2h + h^2,$
$U_2u_2 - h^2,\ U_3u_3 - h^2,\ U_4u_4 - h^2,\ U_5u_5 - h^2,\ U_6u_6 - h^2$

Katsura 5
Ring: $\mathbb{F}_{7583}[x, y, z, t, u, v, h]$
Source: http://fgbrs.lip6.fr/jcf/Benchs/
Regular sequence: yes
Polynomial system:
$2x^2 + 2y^2 + 2z^2 + 2t^2 + 2u^2 + v^2 - vh,$
$xy + yz + 2zt + 2tu + 2uv - uh,$
$2xz + 2yt + 2zu + u^2 + 2tv - th,$
$2xt + 2yu + 2tu + 2zv - zh,$
$t^2 + 2xv + 2yv + 2zv - yh,$
$2x + 2y + 2z + 2t + 2u + v - h$

Katsura 6
Ring: $\mathbb{F}_{7583}[x_1, x_2, x_3, x_4, x_5, x_6, x_7, h]$
Source: http://fgbrs.lip6.fr/jcf/Benchs/
Regular sequence: yes
Polynomial system:
$x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 + 2x_6 + 2x_7 - h,$
$2x_3x_4 + 2x_2x_5 + 2x_1x_6 + 2x_2x_7 - x_6h,$
$x_3^2 + 2x_2x_4 + 2x_1x_5 + 2x_2x_6 + 2x_3x_7 - x_5h,$
$2x_2x_3 + 2x_1x_4 + 2x_2x_5 + 2x_3x_6 + 2x_4x_7 - x_4h,$
$x_2^2 + 2x_1x_3 + 2x_2x_4 + 2x_3x_5 + 2x_4x_6 + 2x_5x_7 - x_3h,$
$2x_1x_2 + 2x_2x_3 + 2x_3x_4 + 2x_4x_5 + 2x_5x_6 + 2x_6x_7 - x_2h,$
$x_1^2 + 2x_2^2 + 2x_3^2 + 2x_4^2 + 2x_5^2 + 2x_6^2 + 2x_7^2 - x_1h$

Katsura 7
Ring: $\mathbb{F}_{7583}[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, h]$
Source: http://fgbrs.lip6.fr/jcf/Benchs/
Regular sequence: yes
Polynomial system:
$x_1^2 + 2x_2^2 + 2x_3^2 + 2x_4^2 + 2x_5^2 + 2x_6^2 + 2x_7^2 + 2x_8^2 - x_1h,$
$2x_1x_2 + 2x_2x_3 + 2x_3x_4 + 2x_4x_5 + 2x_5x_6 + 2x_6x_7 + 2x_7x_8 - x_2h,$
$x_2^2 + 2x_1x_3 + 2x_2x_4 + 2x_3x_5 + 2x_4x_6 + 2x_5x_7 + 2x_6x_8 - x_3h,$
$2x_2x_3 + 2x_1x_4 + 2x_2x_5 + 2x_3x_6 + 2x_4x_7 + 2x_5x_8 - x_4h,$

$x_3^2 + 2x_2x_4 + 2x_1x_5 + 2x_2x_6 + 2x_3x_7 + 2x_4x_8 - x_5h,$
$2x_3x_4 + 2x_2x_5 + 2x_1x_6 + 2x_2x_7 + 2x_3x_8 - x_6h,$
$x_4^2 + 2x_3x_5 + 2x_2x_6 + 2x_1x_7 + 2x_2x_8 - x_7h,$
$x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 + 2x_6 + 2x_7 + 2x_8 - h$

Katsura 8
Ring: $\mathbb{F}_{7583}[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, h]$
Source: http://fgbrs.lip6.fr/jcf/Benchs/
Regular sequence: yes
Polynomial system:
$x_1^2 + 2x_2^2 + 2x_3^2 + 2x_4^2 + 2x_5^2 + 2x_6^2 + 2x_7^2 + 2x_8^2 + 2x_9^2 - x_1h,$
$2x_1x_2 + 2x_2x_3 + 2x_3x_4 + 2x_4x_5 + 2x_5x_6 + 2x_6x_7 + 2x_7x_8 + 2x_8x_9 - x_2h,$
$x_2^2 + 2x_1x_3 + 2x_2x_4 + 2x_3x_5 + 2x_4x_6 + 2x_5x_7 + 2x_6x_8 + 2x_7x_9 - x_3h,$
$2x_2x_3 + 2x_1x_4 + 2x_2x_5 + 2x_3x_6 + 2x_4x_7 + 2x_5x_8 + 2x_6x_9 - x_4h,$
$x_3^2 + 2x_2x_4 + 2x_1x_5 + 2x_2x_6 + 2x_3x_7 + 2x_4x_8 + 2x_5x_9 - x_5h,$
$2x_3x_4 + 2x_2x_5 + 2x_1x_6 + 2x_2x_7 + 2x_3x_8 + 2x_4x_9 - x_6h,$
$x_4^2 + 2x_3x_5 + 2x_2x_6 + 2x_1x_7 + 2x_2x_8 + 2x_3x_9 - x_7h,$
$2x_4x_5 + 2x_3x_6 + 2x_2x_7 + 2x_1x_8 + 2x_2x_9 - x_8h,$
$x_1 + 2x_2 + 2x_3 + 2x_4 + 2x_5 + 2x_6 + 2x_7 + 2x_8 + 2x_9 - h$

Cyclic 5
Ring: $\mathbb{F}_{7583}[a, b, c, d, e, h]$
Source: http://fgbrs.lip6.fr/jcf/Benchs/
Regular sequence: yes
Polynomial system:
$a + b + c + d + e,$
$ab + bc + cd + ae + de,$
$abc + bcd + abe + ade + cde,$
$abcd + abce + abde + acde + bcde,$
$abcde - h^5$

Cyclic 6
Ring: $\mathbb{F}_{7583}[a, b, c, d, e, f, h]$
Source: http://fgbrs.lip6.fr/jcf/Benchs/
Regular sequence: no
Polynomial system:
$abcdef - h^6,$
$abcde + abcdf + abcef + abdef + acdef + bcdef,$
$abcd + bcde + abcf + abef + adef + cdef,$
$abc + bcd + cde + abf + aef + def,$
$ab + bc + cd + de + af + ef,$
$a + b + c + d + e + f$

Cyclic 7
Ring: $\mathbb{F}_{7583}[a, b, c, d, e, f, g, h]$
Source: http://fgbrs.lip6.fr/jcf/Benchs/
Regular sequence: yes
Polynomial system:
$abcdefg - h^7$,
$abcdef + abcdeg + abcdfg + abcefg + abdefg + acdefg + bcdefg$,
$abcde + bcdef + abcdg + abcfg + abefg + adefg + cdefg$,
$abcd + bcde + cdef + abcg + abfg + aefg + defg$,
$abc + bcd + cde + def + abg + afg + efg$,
$ab + bc + cd + de + ef + ag + fg$,
$a + b + c + d + e + f + g$

Gonnet 83
Ring: $\mathbb{F}_{7583}[a_0, a_2, a_3, a_4, a_5, b_0, b_1, b_2, b_3, b_4, b_5, c_0, c_1, c_2, c_3, c_4, c_5, h]$
Source: http://www.symbolicdata.org/
Regular sequence: no
Polynomial system:
a_5b_5,
$a_5b_4 + a_4b_5$,
a_4b_4,
$a_5b_3 + a_3b_5$,
$a_5b_3 + a_3b_5 + 2a_5b_5$,
$a_3b_3 + a_5b_3 + a_3b_5 + a_5b_5$,
$2a_3b_3 + a_5b_3 + a_3b_5$,
$a_4b_2 + a_2b_4$,
a_2b_2,
$a_5b_1 + a_4b_3 + a_3b_4 + b_5h$,
$a_4b_1 + b_4h$,
$a_2b_1 + b_2h$,
$a_0b_1 + a_4b_1 + a_3b_2 + a_2b_3 + b_0h + 2b_1h + b_4h + c_1h$,
$a_5b_0 + a_5b_1 + a_4b_3 + a_3b_4 + 2a_5b_4 + a_0b_5 + 2a_4b_5 + b_5h + c_5h$,
$a_4b_0 + a_4b_1 + a_5b_2 + a_0b_4 + 2a_4b_4 + a_2b_5 + b_4h + c_4h$,
$a_3b_0 + 2a_3b_1 + a_5b_1 + a_0b_3 + a_4b_3 + a_3b_4 + 2b_3h + b_5h + c_3h$,
$a_3b_0 + a_5b_0 + a_3b_1 + a_5b_1 + a_0b_3 + a_4b_3 + a_3b_4 + a_5b_4 + a_0b_5 + a_4b_5 + b_3h + b_5h + c_3h + c_5h - h^2$,
$a_2b_0 + a_2b_1 + a_0b_2 + a_4b_2 + a_2b_4 + b_2h + c_2h$,
$a_0b_0 + a_4b_0 + a_0b_1 + a_4b_1 + a_3b_2 + a_5b_2 + a_2b_3 + a_0b_4 + a_4b_4 + a_2b_5 + b_0h + b_1h + b_4h + c_0h + c_1h + c_4h$

f744
Ring: $\mathbb{F}_{7583}[U_7, U_6, U_5, U_4, U_3, U_2, u_7, u_6, u_5, u_4, u_3, u_2, h]$
Source: http://fgbrs.lip6.fr/jcf/Benchs/

Regular sequence: no
Polynomial system:
$2u_7 + 2u_6 + 2u_5 + 2u_4 + 2u_3 + 2u_2 + h,$
$2U_7 + 2U_6 + 2U_5 + 2U_4 + 2U_3 + 2U_2 + h,$
$8U_6u_7 + 8U_5u_7 + 8U_4u_7 + 8U_3u_7 + 8U_2u_7 + 8U_6u_6 + 8U_5u_6 + 8U_4u_6 + 8U_3u_6 + 8U_2u_6 +$
$8U_5u_5 + 8U_4u_5 + 8U_3u_5 + 8U_2u_5 + 8U_4u_4 + 8U_3u_4 + 8U_2u_4 + 8U_3u_3 + 8U_2u_3 + 8U_2u_2 - 17h^2,$
$8U_7u_6 + 8U_6u_6 + 8U_7u_5 + 8U_6u_5 + 8U_5u_5 + 8U_7u_4 + 8U_6u_4 + 8U_5u_4 + 8U_4u_4 + 8U_7u_3 +$
$8U_6u_3 + 8U_5u_3 + 8U_4u_3 + 8U_3u_3 + 8U_7u_2 + 8U_6u_2 + 8U_5u_2 + 8U_4u_2 + 8U_3u_2 + 8U_2u_2 - 17h^2,$
$16U_5U_3u_4 + 16U_5U_2u_4 + 16U_5U_2u_3 + 16U_4U_2u_3 + 8U_5u_4h + 8U_5u_3h + 8U_4u_3h +$
$8U_5u_2h + 8U_4u_2h + 8U_3u_2h + 18U_5h^2 + 18U_4h^2 + 18U_3h^2 + 18U_2h^2 + 11h^3,$
$16U_4u_5u_3 + 16U_4u_5u_2 + 16U_3u_5u_2 + 16U_3u_4u_2 + 8U_4u_5h + 8U_3u_5h + 8U_2u_5h +$
$8U_3u_4h + 8U_2u_4h + 8U_2u_3h + 18u_5h^2 + 18u_4h^2 + 18u_3h^2 + 18u_2h^2 + 11h^3,$
$U_2u_2 - h^2,\ U_3u_3 - h^2,\ U_4u_4 - h^2,\ U_5u_5 - h^2,\ U_6u_6 - h^2,\ U_7u_7 - h^2$

Schrans-Troost
Ring: $\mathbb{F}_{7583}[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, h]$
Source: http://www.math.msu.edu/~jan/Demo/TIMINGS.html
Regular sequence: yes
Polynomial system:
$8x_1^2 + 8x_1x_2 + 8x_1x_3 - 8x_2x_3 + 2x_1x_4 + 2x_1x_5 + 2x_1x_6 - 2x_5x_6 + 2x_1x_7 - 2x_4x_7 - x_1h,$
$8x_1x_2 + 8x_2^2 - 8x_1x_3 + 8x_2x_3 + 2x_2x_4 + 2x_2x_5 + 2x_2x_6 - 2x_4x_6 + 2x_2x_7 - 2x_5x_7 - x_2h,$
$-8x_1x_2 + 8x_1x_3 + 8x_2x_3 + 8x_3^2 + 2x_3x_4 + 2x_3x_5 - 2x_4x_5 + 2x_3x_6 + 2x_3x_7 - 2x_6x_7 - x_3h,$
$2x_1x_4 + 2x_2x_4 + 2x_3x_4 + 8x_4^2 - 2x_3x_5 + 8x_4x_5 - 2x_2x_6 + 2x_4x_6 - 2x_1x_7 + 2x_4x_7 +$
$6x_4x_8 - 6x_5x_8 - x_4h,$
$-2x_1x_4 - 2x_2x_5 - 2x_3x_6 + 2x_1x_7 + 2x_2x_7 + 2x_3x_7 + 2x_4x_7 + 2x_5x_7 + 8x_6x_7 + 8x_7^2 -$
$6x_6x_8 + 6x_7x_8 - x_7h,$
$-2x_2x_4 - 2x_1x_5 + 2x_1x_6 + 2x_2x_6 + 2x_3x_6 + 2x_4x_6 + 2x_5x_6 + 8x_6^2 - 2x_3x_7 + 8x_6x_7 +$
$6x_6x_8 - 6x_7x_8 - x_6h,$
$-2x_3x_4 + 2x_1x_5 + 2x_2x_5 + 2x_3x_5 + 8x_4x_5 + 8x_5^2 - 2x_1x_6 + 2x_5x_6 - 2x_2x_7 + 2x_5x_7 -$
$6x_4x_8 + 6x_5x_8 - x_5h,$
$-6x_4x_5 - 6x_6x_7 + 6x_4x_8 + 6x_5x_8 + 6x_6x_8 + 6x_7x_8 + 8x_8^2 - x_8h$