A Signature-based Algorithm for Computing the Nondegenerate Locus of a Polynomial System

Christian Eder, Pierre Lairez, Rafael Mohr, Mohab Safey El Din

A Signature-based Algorithm for Computing the Nondegenerate Locus of a Polynomial System*

Christian Eder[a], Pierre Lairez[b], Rafael Mohr[a,c,*], Mohab Safey El Din[c]

[a] *RPTU Kaiserslautern-Landau, Department of Mathematics, Gottlieb-Daimler-Straße 48, 67663 Kaiserslautern, Germany*
[b] *Université Paris-Saclay, Inria, Inria Saclay, 91120 Palaiseau, France*
[c] *Sorbonne Université, LIP6, CNRS, Boîte courrier 169, 75252 Paris Cedex 05, France*

Abstract

Polynomial system solving arises in many application areas to model non-linear geometric properties. In such settings, polynomial systems may come with degeneration which the end-user wants to exclude from the solution set. The nondegenerate locus of a polynomial system is the set of points where the codimension of the solution set matches the number of equations.

Computing the nondegenerate locus is classically done through ideal-theoretic operations in commutative algebra such as saturation ideals or equidimensional decompositions to extract the component of maximal codimension.

By exploiting the algebraic features of signature-based Gröbner basis algorithms we design an algorithm which computes a Gröbner basis of the equations describing the closure of the nondegenerate locus of a polynomial system, without computing first a Gröbner basis for the whole polynomial system.

Keywords: Gröbner Basis, Ideal Decomposition, Algorithm

1. Introduction

Problem Statement. Fix a field $\mathbb{K}$ with an algebraic closure $\overline{\mathbb{K}}$ and a polynomial ring $R := \mathbb{K}[x_1, \ldots, x_n]$ over $\mathbb{K}$. Let $f_1, \ldots, f_c \in R$ and $V := \left\{p \in \overline{\mathbb{K}}^n \;\middle|\; f_1(p) = \cdots = f_c(p) = 0\right\}$. Further define the ideal $I := \langle f_1, \ldots, f_c\rangle = \{\sum_{i=1}^{c} q_i f_i \mid q_i \in R\}$. The algebraic set V is a finite union of irreducible components. By the

*This work has been supported by European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Actions, grant agreement 813211 (POEMA) by European Research Council under the European Union's Horizon Europe research and innovation programme, grant agreement 101040794 (10000 DIGITS); by the joint ANR-FWF grant ANR-19-CE48-0015 (ECARP), the ANR grant ANR-19-CE40-0018 (De Rerum Natura), the DFG Sonderforschungsbereich TRR 195, and the Forschungsinitiative Rheinland-Pfalz.

*Corresponding author

Email addresses: ederc@mathematik.uni-kl.de (Christian Eder), pierre.lairez@inria.fr (Pierre Lairez), Rafael.Mohr@lip6.fr (Rafael Mohr), mohab.safey@lip6.fr (Mohab Safey El Din)

Principal Ideal Theorem [21, Theorem 10.2] the codimension of the $\mathbb{K}$-irreducible components of V is at most c. Let V_c denote the union of the components of V of codimension exactly c. In particular $V_c = \varnothing$ when $c > n$.

The goal of this paper is to compute a Gröbner basis of an ideal whose zero set is V_c, which we call the *nondegenerate locus* of the system $f_1, \ldots, f_c$ (note that we may not compute a *radical* ideal).

Prior works and scientific locks. State-of-the-art algorithms to compute the nondegenerate locus of $f_1, \ldots, f_c$ rely on the more general problem of computing the equidimensional decomposition of the ideal that they generate. There is a vast body of literature split along what data structure is used for the output into two research lines.

The first family of algorithms computes a Gröbner basis of the ideal of each component. There are two different approaches in this line. The first uses projections, computed with *elimination orderings*, to reduce the problem to a problem for hypersurfaces [27, 35, 8]. The second relies on homological characterizations of the dimension and the computation of free resolutions [22]. See [15, 31, 43] and references therein for further references. Both approaches use Gröbner basis algorithms as a black box for performing various ideal-theoretic operations, in particular ideal quotients (also known as colon ideals).

A second family of algorithms outputs equidimensional components of I or its radical through *lazy* representations, i.e. as complete intersections over a non-empty Zariski open set. This is the case for the so-called regular chains which go back to Wu-Ritt characteristic sets [46]. Algorithms based on regular chains put into practice a kind of D5 principle [17] to split geometric objects by enforcing an equiprojectability property. See [32, 44, 45, 12, 2, 38] and references therein for further references.

When the base field $\mathbb{K}$ has characteristic 0 (or large enough characteristic), geometric resolution algorithms [28] can also be used. This class of algorithms culminates with the incremental algorithm in [36, 37] which avoids equiprojectability issues by performing a linear change of variables to ensure Noether position properties. One feature is that input polynomials are encoded with straight-line programs to take advantage of evaluation properties. See also [34] for a similar approach. It also gives the best known complexity for equidimensional decomposition: linear in the evaluation complexity of the input system and polynomial in some algebraic degree.

As of software, the computer algebra systems Singular [16], Macaulay2 [30] and Magma [6] implement the algorithm of [22] to perform equidimensional decomposition. Maple implements algorithms for computing regular chains [14, 13, 9, 10, 39] and algorithms based on Gröbner bases. The algorithm by Gianni et al. [27] is used for prime decomposition and, combined with techniques from [3], for equidimensional decomposition. All these implementations use Gröbner basis algorithms as a black box.

Main results. By contrast with previous work, we only focus on computing the nondegenerate locus of a system, not the full equidimensional decomposition of the corresponding ideal. The main difference to other Gröbner basis based tech-

niques to compute equidimensional decompositions is that we enlarge I *while* a Gröbner basis for I is computed and return a Gröbner basis of a nondegenerate locus of I when this Gröbner basis computation is finished. Modifying or splitting the ideal in question in the middle of Gröbner basis algorithms is a natural and appealing idea [e.g. 29].

This idea requires one to answer *(i)* when the ideal in question should be enlarged and *(ii)* how to minimize the cost of enlarging the ideal in question. The algorithm we propose tackles both issues.

We tackle problem *(i)* by following the incremental structure of certain *signature-based Gröbner basis* (sGB) algorithms on which our work is based [24, 25, 26]. We describe a version of such an sGB algorithm in section 3. Incremental means here that these algorithms proceed by computing first a Gröbner basis for $\langle f_1, f_2\rangle$ then use the result to compute a Gröbner basis for $\langle f_1, f_2, f_3\rangle$ and so on. In addition, sGB algorithms keep track of an auxiliary data structure, called a *signature*, which is attached to each considered polynomial. This enables one to *exclude* certain polynomials from the set of polynomials to be processed by reduction in Buchberger's algorithm.

As a consequence these algorithms have the feature that, having computed a Gröbner basis for $I_{i-1} := \langle f_1, \ldots, f_{i-1}\rangle$, a reduction to zero happens in the Gröbner basis computation for I_i if any only if f_i is a zero divisor modulo I_{i-1}. In this case $V_{i-1} := V(I_{i-1})$ has irreducible components on which f_i is identically zero (the union of which is henceforth denoted $V_{i-1,f_i=0}$) and components which are not contained in the hypersurface $V(f_i)$ (the union of which is henceforth denoted $V_{i-1,f_i\neq 0}$). Assuming that $V(I_{i-1})$ is equidimensional of codimension $i-1$, to compute an ideal representing the nondegenerate locus of $V(I_i)$ we may then proceed as follows (see Algorithm 1):

1. Compute ideals representing $V_{i-1,f_i=0}$ and $V_{i-1,f_i\neq 0}$ (via the ideal-theoretic operation of *saturation*).
2. Compute an ideal representing $W := V_{i-1,f_i\neq 0} \cap V(f_i)$.
3. Remove from W all components contained in $V_{i-1,f_i=0}$ (again via saturation).

Iterating over the set of input equations with these three steps, using the result of each iterative step as input for the next invocation of this loop and slightly adapting the third step to remove all components which are contained in components of higher dimension then yields an ideal representing the nondegenerate locus of I. We describe this algorithm from a purely algebraic perspective in section 2.

To tackle problem *(ii)* we exploit a feature of the incremental sGB algorithms first captured in the G2V algorithm [25]: The data of a signature can be enlarged so as to *simultaneously* compute a Gröbner basis for I_i and the quotient ideal $(I_{i-1} : f_i) := \{g \in R \mid gf_i \in I_{i-1}\}$ (which, if I_{i-1} is a radical ideal, corresponds precisely to $V(I_{i-1,f_i\neq 0})$) in each incremental step.

Using this idea we modify the baseline sGB algorithm we use to simultaneously perform steps 1 and 2 of the above loop (i.e. in a single Gröbner basis

computation). This is done essentially by immediately inserting an element $g \in (I_{i-1} : f_i)$ once it is identified during the run of the sGB algorithm. We manage this insertion of elements that do not lie in the original ideal I with a data structure we call an *sGB tree* (see section 3.3) which allows us to perform this modification with the needed technical properties of signatures ensured. This yields a signature-based version of Algorithm 1, Algorithm 8. Besides managing the insertion of new generators into some initial ideal, the sGB data structure also leaves open the future possibility of designing signature-based ideal decomposition algorithms.

We finally show experimentally in section 5.2 that the consequence of this simple modification is a massive cost reduction in the overhead compared to a "naive" implementation of Algorithm 1 where one uses saturation procedures as a blackbox. As is also shown, it additionally enables us to compute the nondegenerate locus of systems which are out of the reach of equidimensional decomposition algorithms available in state of the art computer algebra systems.

2. The basic algorithm

Consider a codimension k irreducible variety $X \subseteq \overline{\mathbb{K}}^n$ and a polynomial $f \in R$. Assume that $X \cap V(f) \neq \emptyset$. Either $X \subseteq V(f)$, and so $X \cap V(f) = X$, or $X \cap V(f)$ is equidimensional of codimension $k+1$ (that is, all the irreducible components of $X \cap V(f)$ have codimension $k+1$). If X is not irreducible, then $X \cap V(f)$ may not be equidimensional. Yet, the alternative above applies to each irreducible component of X. The components of X which are not included in $V(f)$ are exactly the components of the closure of $X \setminus V(f)$, while the components of X which are included in $V(f)$ are exacly the components of the closure of $X \setminus \overline{(X \setminus V(f))}$. This leads to the decomposition of $X \cap V(f)$ as the union of two equidimensional varieties of codimension k and $k+1$ respectively:

$$X \cap V(f) = \overline{\left(X \setminus \overline{X \setminus V(f)}\right)} \cup \left(\overline{X \setminus V(f)} \cap V(f)\right).$$

This is the basic identity that we leverage to compute, incrementally, the codimension c components of an ideal $\langle f_1, \ldots, f_c \rangle$. In an ideal theoretic language, this reformulates as follows.

For two ideals $I, J \subseteq R$, we write $I \overset{\text{rad}}{=} J$ for the equality of the radicals $\sqrt{I} = \sqrt{J}$. An ideal I is *equidimensional* if all the irreducible components of $V(I)$ have the same dimension. Recall that $I : J$ is the ideal $\{p \in R \mid pJ \subseteq I\}$. Recall also that $I : J^k$ yields an increasing sequence of ideals as $k \to \infty$, so it eventually stabilizes in an ideal denoted $I : J^\infty$, the *saturation of I by J*. If J is generated by a single element f, it is simply denoted $I : f^\infty$.

Lemma 2.1. *For any ideal $J \subseteq R$ and any $f \in R$ we have*

$$J + \langle f \rangle \overset{\text{rad}}{=} \big((J : f^\infty) + \langle f \rangle\big) \cap \big(J : (J : f^\infty)\big)$$

Moreover, if J is equidimensional of codimension $c < n$ then $((J : f^\infty) + \langle f \rangle)$ is either the unit ideal or equidimensional of codimension $c+1$ and $(J : (J : f^\infty))$ is either the unit ideal or equidimensional of codimension c.

Proof. For the left-to-right inclusion, it is clear that J is included in the right-hand side, so it remains to check that f is in the radical of both terms of the intersection. It is obvious that $f \in (J : f^\infty) + \langle f \rangle$, so it remains to prove that f is in the radical of $J : (J : f^\infty)$. So let $g \in J : f^\infty$, that is $gf^\ell \in J$ for some $\ell \geq 0$, which we may rewrite as $f \in \sqrt{J : g}$. To conclude, we observe that

$$\sqrt{J : (J : f^\infty)} = \bigcap_{g \in J : f^\infty} \sqrt{J : g},$$

so $f \in \sqrt{J : (J : f^\infty)}$.

Conversely, let $p \in ((J : f^\infty) + \langle f \rangle) \cap (J : (J : f^\infty))$. Write $p = q + af$ where $q \in (J : f^\infty)$ and $a \in R$. Since $p \in (J : (J : f^\infty))$, we have $pq \in J$. But $pq = q^2 + aqf$, so $q^2 \in J + \langle f \rangle$. It follows that $q \in \sqrt{J + \langle f \rangle}$, thus proving the stated equality.

For the statement on equidimensionality, assume that neither of the two ideals on the right hand side are the unit ideal. We may rely on the geometric interpretation above: the zero set of $J : f^\infty + \langle f \rangle$ is $\overline{X \setminus V(f)} \cap V(f)$, where $X = V(J)$. It thus only remains to show the equidimensionality of $J : (J : f^\infty)$. Note that the equidimensionality of $J : (J : f^\infty)^\infty$ is clear because its associated algebraic set consists of the union of the components of X on which f identically vanishes. Now it holds that $J : (J : f^\infty) \overset{\text{rad}}{=} J : (J : f^\infty)^\infty$, see e.g. [33, Proposition 23]. This shows the equidimensionality of $J : (J : f^\infty)$. □

We are now ready to describe Algorithm 1. To do this we suppose for now that we have an algorithm for computing the quotient ideal $J : K$ and the saturation $J : K^\infty$, given generators for J and K. Given $c \leq n$ elements $f_1, \ldots, f_c \in R$ the core loop of Algorithm 1 starts with the ideal $J = \langle f_1 \rangle$ and continously replaces it with $(J : f_k^\infty) + \langle f_k \rangle$ for each k. By Lemma 2.1 the resulting ideal will be equidimensional of codimension c. Note however that it may have components that the original ideal $I = \langle f_1, \ldots, f_c \rangle$ does not have, as shown by the following example. In the algorithm, these additional components are removed with saturations at every iterative step with the loop on line 7.

Example 2.2. Let $R = \mathbb{Q}[x, y, z]$ and $f_1 = xy$, $f_2 = xz$. Then $(\langle xy \rangle : xz^\infty) + \langle xz \rangle = \langle y, xz \rangle$ which has the component $\langle y, x \rangle$ which is not a component of $\langle f_1, f_2 \rangle = \langle x \rangle \cap \langle y, z \rangle$.

To prove the correctness of Algorithm 1 we also need the following proposition:

Lemma 2.3. *For any ideals $I, J \subseteq R$ and any $f \in R$, we have*

(i) $(I \cap J) + \langle f \rangle \overset{\text{rad}}{=} (I + \langle f \rangle) \cap (J + \langle f \rangle)$;

(ii) $I \cap J \overset{\text{rad}}{=} (I : J^\infty) \cap J$;

(iii) if $f \in I$, then $I : J^\infty = I : (J + \langle f \rangle)^\infty$.

Algorithm 1 Computation of the nondegenerate locus

Input: A set of generators $f_1, \ldots, f_c$ for an ideal I in R where $c \leq n$
Output: A set of generators G for the nondegenerate part of $f_1, \ldots, f_c$

```
J ← 0, as an ideal of R
𝒦 ← ∅
for k from 1 to c do
    H ← J : f_k^∞
    𝒦 ← 𝒦 ∪ {J : H}
    J ← H + ⟨f_k⟩
    for K ∈ 𝒦 do
        J ← J : K^∞
    end for
end for
return J
```

Proof. For the first item,

$$(I + \langle f \rangle) \cap (J + \langle f \rangle) \overset{\text{rad}}{=} (I + \langle f \rangle)(J + \langle f \rangle) \overset{\text{rad}}{=} IJ + \langle f \rangle \overset{\text{rad}}{=} (I \cap J) + \langle f \rangle.$$

For the second one, the left-to-right inclusion is clear. Conversely, let $f \in (I : J^\infty) \cap J$ and let $k > 0$ such that $fJ^k \subseteq I$. In particular $f^{k+1} \in I$. So $f \in \sqrt{I}$.

For the last item is trivial from the definition of saturation. □

Theorem 2.4. *On input $f_1, \ldots, f_c \in R$ with $c \leq n$, Algorithm 1 terminates and outputs an ideal J such that $V(J)$ is the nondegenerate locus of the input system.*

Proof. We define $J_0 := \langle 0 \rangle$, and then recursively

$$K_i := (J_{i-1} : (J_{i-1} : f_i^\infty)),$$
$$\text{and } J_i := \left((J_{i-1} : f_i^\infty) + \langle f_i \rangle : \left(\textstyle\prod_{j=1}^{i} K_j \right)^\infty \right).$$

It is clear that Algorithm 1 returns the ideal J_c. Now, let $I_i = \langle f_1, \ldots, f_i \rangle$. The main loop invariant, that we prove by induction on i, is

$$I_i \overset{\text{rad}}{=} J_i \cap \bigcap_{j=1}^{i} \left(K_j + \langle f_{j+1}, \ldots, f_i \rangle \right). \tag{1}$$

From this, we deduce that the zero set of J_c is contained in the algebraic set defined by $f_1 = \cdots = f_c = 0$. We will prove later that J_c is equidimensional of codimension c, that the components of the ideals $(K_j + \langle f_{j+1}, \ldots, f_c \rangle)$ have codimension less than c and do not contain any components of J_c.

It is trivially true that (1) holds for $i = 0$. For $i > 0$, we have

$$I_i = I_{i-1} + \langle f_i \rangle \overset{\text{rad}}{=} \left(J_{i-1} \cap \bigcap_{j=1}^{i-1} \left(K_j + \langle f_{j+1}, \ldots, f_{i-1} \rangle \right) \right) + \langle f_i \rangle$$
$$\overset{\text{rad}}{=} \left(J_{i-1} + \langle f_i \rangle \right) \cap \bigcap_{j=1}^{i-1} \left(K_j + \langle f_{j+1}, \ldots, f_i \rangle \right), \quad \text{by Lemma 2.3(i).}$$

Besides, by Lemma 2.1,

$$\begin{aligned} J_{i-1} + \langle f_i \rangle &\overset{\text{rad}}{=} \left((J_{i-1} : f_i^\infty) + \langle f_i \rangle \right) \cap \left(J_{i-1} : (J_{i-1} : f_i^\infty) \right) \\ &= \left((J_{i-1} : f_i^\infty) + \langle f_i \rangle \right) \cap K_i. \end{aligned}$$

For short, let $J_i' = (J_{i-1} : f_i^\infty) + \langle f_i \rangle$. Combining the equalities above, we have

$$\begin{aligned} I_i &\overset{\text{rad}}{=} J_i' \cap \bigcap_{j=1}^{i} \left(K_j + \langle f_{j+1}, \ldots, f_i \rangle \right) \\ &\overset{\text{rad}}{=} \left(J_i' : \left(\textstyle\prod_{j=1}^{i} K_j \right)^\infty \right) \cap \bigcap_{j=1}^{i} \left(K_j + \langle f_{j+1}, \ldots, f_i \rangle \right), \end{aligned}$$

using Lemma 2.3(ii) and (iii) (note that $f_1, \ldots, f_i \in J_i'$). This last equality is exactly (1).

Now, we analyze the dimensions and show that $V(J_i)$ is exactly the nondegenerate locus of $f_1, \ldots, f_i$. Indeed, using Lemma 2.1, we check by induction on i that J_i is equidimensional of codimension i (unless $J_i = \langle 1 \rangle$ in which case the nondegenerate locus of $f_1, \ldots, f_i$ is the unit ideal) and that K_i is equidimensional of codimension $i - 1$ (unless $K_i = \langle 1 \rangle$ in which case the nondegenerate locus of $f_1, \ldots, f_i$ is the one of $f_1, \ldots, f_{i-1}$ plus $\langle f_i \rangle$). It follows that all the components of $K_j + \langle f_{j+1}, \ldots, f_i \rangle$ have codimension at most $i - 1$. Moreover, no component of J_i is included in any K_j, for $j \leq i$, since J_i is saturated by the K_j. Therefore, using (1), the codimension i components of I_i are exactly the components of J_i.

Hence, we deduce that J_c is equidimensional of codimension c whose components are not contained in the ones of $K_j + \langle f_{j+1}, \ldots, f_c \rangle$, the components of which have codimention less than c. Besides, we already observed that its zero set is contained in the one defined by the input polynomials $f_1, \ldots, f_c$. Since (1) holds, we conclude that $V(J_c)$ is the nondegenerate locus of the input system. □

3. Signature-based Gröbner basis computations

We will rely on the theory of signature-based Gröbner bases in order to efficiently implement Algorithm 1.

3.1. Signatures and extended sig-poly pairs

We fix in the following a monomial order on R and a sequence of polynomials $f_1, \ldots, f_r \in R$. Let $I := \langle f_1, \ldots, f_r \rangle$ and $I_i := \langle f_1, \ldots, f_i \rangle$. We describe an algorithm which computes a Gröbner basis for I and presents the following features:

1. It computes a Gröbner basis for I incrementally, i.e. first for $\langle f_1 \rangle$ then for $\langle f_1, f_2 \rangle$ etc.
2. It simultaneously computes Gröbner bases for each ideal $(\langle f_1, \ldots, f_{i-1} \rangle : f_i)$, $i = 2, \ldots, r$.

This algorithm belongs to the class of so called *signature-based* Gröbner basis algorithms, the first of which was the F5 algorithm presented in [24]. Since then the class of signature-based algorithms has been greatly extended, see [18] for a survey. The idea of leveraging signature-based algorithms to compute simultaneously some colon ideals first appeared in [25]. The algorithm we present here is closely related, with some elements from the F5 algorithm. The algorithm presented in this section is fully encompassed by the general algorithmic framework presented in [18].

We start by defining signatures.

Definition 3.1. A *signature* is a pair $\sigma = (i, m)$ of an index in $\{1, \ldots, r\}$ and a monomial in R. The first component is called the *index*, and denoted $\mathrm{ind}(\sigma)$. The second component is called the *monomial part* of σ.

We order the signatures lexicographically, i.e. by writing

$$(i, m) < (j, n) \Leftrightarrow i < j \text{ or } i = j \text{ and } m < n.$$

The product of a monomial $a \in R$ and a signature $\sigma = (i, h)$ is defined by $a\sigma = (i, ah)$. A signature σ divides another signature τ if there is a monomial a such that $a\sigma = \tau$, so in particular $\mathrm{ind}(\sigma) = \mathrm{ind}(\tau)$.

The possible indices of a signature are the indices of the input equations. This relation between the index of a signature and one of the equations f_i is made stronger by the following object:

Definition 3.2. An *extended sig-poly pair* is a triple $\alpha = (f, \sigma, h)$, where $f, h \in R$ and σ is a signature such that $\mathrm{lm}(h)$ is equal to the monomial part of σ. The first component f is called the *polynomial part* of α, denoted $\mathrm{poly}(\alpha)$, the second component σ is called the *signature*, denoted $\mathfrak{s}(\alpha)$, and the third component is called the *quotient*, denoted $\mathrm{quo}(\alpha)$. The index of α, denoted $\mathrm{ind}(\alpha)$ is the index of its signature. We further impose that

$$\mathrm{poly}(\alpha) - \mathrm{quo}(\alpha) f_{\mathrm{ind}(\alpha)} \in I_{\mathrm{ind}(\alpha)-1}. \tag{2}$$

The product of a monomial $a \in R$ and an extended sig-poly pair α is defined by

$$\mathrm{poly}(a\alpha) = a\,\mathrm{poly}(\alpha), \quad \mathfrak{s}(a\alpha) = a\mathfrak{s}(\alpha), \quad \text{and } \mathrm{quo}(a\alpha) = a\,\mathrm{quo}(\alpha).$$

Algorithm 2 Regular reduction

```
1: procedure RegularReduction(α, G)
2:     f ← poly(α)
3:     h ← quo(α)
4:     while R := {(b, β) ∈ R × G | b lm(poly(β)) = lm(f), bs(β) < s(α)} ≠ ∅
       do
5:         (b, β) ← some element in R
6:         f ← f − b poly(β)
7:         if ind(β) = ind(α) then
8:             h ← h − b quo(β)
9:         end if
10:    end while
11:    return (f, s(α), h)
12: end procedure
```

The concept of an S-pair from Buchberger's algorithm extends to extended sig-poly pairs. Given two extended sig-poly pairs α and β with $\mathrm{poly}(\alpha) \neq 0$ and $\mathrm{poly}(\beta) \neq 0$ let $c = \mathrm{lcm}(\mathrm{lm}(\mathrm{poly}(\alpha)), \mathrm{lm}(\mathrm{poly}(\beta)))$, $a = c/\mathrm{lt}(\mathrm{poly}(\alpha))$ and $b = c/\mathrm{lt}(\mathrm{poly}(\beta))$, then define the S-pair of α and β, denoted $\mathfrak{s}p(\alpha, \beta)$ by

$$\mathrm{poly}(\mathfrak{s}p(\alpha,\beta)) = a\,\mathrm{poly}(\alpha) - b\,\mathrm{poly}(\beta), \quad \mathfrak{s}(\mathfrak{s}p(\alpha,\beta)) = \max(\mathfrak{s}(a\alpha), \mathfrak{s}(b\beta)),$$

and

$$\mathrm{quo}(\mathfrak{s}p(\alpha,\beta)) = \begin{cases} a\,\mathrm{quo}(\alpha) & \text{if } \mathrm{ind}(\alpha) > \mathrm{ind}(\beta), \\ a\,\mathrm{quo}(\alpha) - b\,\mathrm{quo}(\beta) & \text{if } \mathrm{ind}(\alpha) = \mathrm{ind}(\beta), \\ -b\,\mathrm{quo}(\beta) & \text{if } \mathrm{ind}(\alpha) < \mathrm{ind}(\beta). \end{cases}$$

In particular, the polynomial part of $\mathfrak{s}p(\alpha, \beta)$ is the usual S-pair of $\mathrm{poly}(\alpha)$ and $\mathrm{poly}(\beta)$. We say that α and β *form a regular S-pair* if $\mathfrak{s}(a\alpha) \neq \mathfrak{s}(b\beta)$. (We will only consider such S-pairs.) It is easy to check that Invariant (2) is preserved.

The *regular reduction* of an extended sig-poly pair α with respect to a set G of sig-poly pairs is defined to be the output of Algorithm 2. The procedure tries to reduce the leading term of $\mathrm{poly}(\alpha)$ using some multiple $b\beta$ of an extended sig-poly pair $\beta \in G$ such that $b\mathfrak{s}(\beta) < \mathfrak{s}(\alpha)$. The procedure stops when there is no such reducer. Compared to the usual division algorithm in polynomial rings, only reduction by lower signature elements is allowed. Moreover, there is some extra computations to preserve Invariant (2).

We may now describe a variant of Buchberger's algorithm using extended sig-poly pairs and regular reduction, see Algorithm 3. In line 6 we always choose the S-pair with minimal signature for reduction, and signatures are ordered first by indices. As a result, signatures are processed in index 1 (which may produce further S-pairs with index ≥ 1), then in index 2 (which may produce further S-pairs with index ≥ 2), etc. So a Gröbner basis for I is computed incrementally: first for $\langle f_1 \rangle$, then for $\langle f_1, f_2 \rangle$ etc. Computing with extended sig-poly pairs

Algorithm 3 Buchberger with signatures

Input: $f_1, \ldots, f_r \in R$

Output: Gröbner bases of $\langle f_1, \ldots, f_r \rangle$ and of $\langle f_1, \ldots, f_{k-1} \rangle : f_k$ $(1 \leq k \leq r)$

```
1: procedure BUCHBERGER(f_1, ..., f_r)
2:     G ← {(f_i, (i,1), 1) | 1 ≤ i ≤ r}
3:     S_1, ..., S_r ← ∅
4:     P ← {(α, β) | α, β ∈ G form a regular S-pair}
5:     while P ≠ ∅ do
6:         (α, β) ← the pair in P with 𝔰(𝔰p(α, β)) minimal
7:         P ← P \ {(α, β)}
8:         γ ← REGULARREDUCTION(𝔰p(α, β), G)
9:         G ← G ∪ {γ}
10:        if poly(γ) ≠ 0 then
11:            P ← P∪{(γ, β) | β ∈ G, poly(β) ≠ 0, forms a regular S-pair with γ}
12:        else   (record the quotient of the zero reduction)
13:            S_ind(γ) ← S_ind(γ) ∪ {quo(γ)}
14:        end if
15:    end while
16:    return {poly(β) | β ∈ G}, S_1, ..., S_r
17: end procedure
```

makes it possible to simultaneously compute a Gröbner basis for I and for all the ideals $(\langle f_1, \ldots, f_{i-1} \rangle : f_i)$ for $i = 2, \ldots, r$. Indeed, if for an extended sig-poly pair γ we find during the run of Algorithm 3 that $\mathrm{poly}(\gamma) = 0$, then $\mathrm{quo}(\gamma)$ is an element of the quotient ideal $I_{\mathrm{ind}(\gamma)-1} : f_{\mathrm{ind}(\gamma)}$, in view of Definition 3.2.

Proposition 3.3. *On input $f_1, \ldots, f_r \in R$, Algorithm 3 terminates and the set $\{\mathrm{poly}(\alpha) \mid \alpha \in G\}$ is a Gröbner basis of the ideal $\langle f_1, \ldots, f_r \rangle$. The sets S_i are Gröbner bases of the ideals $\langle f_1, \ldots, f_{i-1} \rangle : f_i$ for each $i = 2, \ldots, r$.*

We skip the proof as we will only rely on the stronger Theorem 3.5 below.

3.2. From Buchberger to sGB

The signature and the quotient of each extended sig-poly pair in the data makes it possible to compute the colon ideals $\langle f_1, \ldots, f_{i-1} \rangle : f_i$ as a by-product of an incremental computation of a Gröbner basis of $\langle f_1, \ldots, f_r \rangle$. Moreover, this is the discovery of Faugère [24], signatures make it possible to discard many S-pairs while preserving the essential properties of Algorithm 3. The overarching principle is the following: *at most one sig-poly pair has to be regular-reduced at each signature*. This is made precise by the following statement.

Lemma 3.4 ([20, Lemma 4])**.** *In the course of Algorithm 3, assume that only S-pairs in signature $\geq \sigma := (i, m)$ are left in P. Then for any extended sig-poly pairs γ and γ' with $\mathfrak{s}(\gamma) = \mathfrak{s}(\gamma') = \sigma$,*

$$\mathrm{poly}(\textsc{RegularReduction}(\gamma, G)) = \mathrm{poly}(\textsc{RegularReduction}(\gamma', G))$$

Algorithm 4 The rewritability criterion

Input: α an extended sig-poly pair, m a monomial, G a set of extended sig-poly pairs with $\alpha \in G$

Output: Returns true if $m\alpha$ is rewritable w.r.t. G; false otherwise

```
1: procedure REWRITABLE(α, m, G)
2:     for δ ∈ G do
3:         if 𝔰(δ) divides 𝔰(mα) and δ was added to G later than α then
4:             return true    (Singular criterion)
5:         else if 𝔰(δ) divides 𝔰(mα) and poly(δ) = 0 then
6:             return true    (Syzygy criterion)
7:         else if ind(δ) < ind(α) and lm(poly(δ)) divides lm(quo(α)) then
8:             return true    (Koszul criterion)
9:         end if
10:    end for
11:    return false
12: end procedure
```

This leads to Algorithm 5. It is similar to Algorithm 3, the only difference is the check on line 9, the *rewritability check*, which trims many computations. At a given signature, this check will retain at most one element of P. The condition on line 7 discards even more S-pairs by predicting that they will reduce to zero.

More precisely, in the context of Lemma 3.4, we can predict that all S-pairs with signature σ will reduce to the same element. The first effect of the rewritability check is the removal of all S-pairs with signature σ, except at most one. Secondly, Lemma 3.4 may be used to predict that an S-pair will reduce to zero. There are two criteria for that:

Syzygy criterion If an element in signature τ has reduced to zero, then every element in signature $a\tau$ (for any monomial a) will reduce to zero;

Koszul criterion If we have a sig-poly pair with polynomial part h and index $< \mathrm{ind}(\sigma)$, then every element in signature $(a \operatorname{lm} h, \mathrm{ind}(\sigma))$ will reduce to zero, (because $hf_{\mathrm{ind}(\sigma)}$ will obviously reduce to zero).

This explains the different checks in the rewritability criterion (Algorithm 3.2), see [18, section 7.1] for a detailed discussion.

Theorem 3.5. *On input $f_1, \ldots, f_r \in R$, Algorithm 5 terminates and outputs subsets G, $S_1, \ldots, S_r$ of R such that:*

(i) G is a Gröbner basis of $\langle f_1, \ldots, f_r \rangle$;

(ii) $I_{i-1} + \langle S_i \rangle = I_{i-1} : f_i$.

Moreover, on line 15, when a polynomial g is inserted in some S_i, then $\operatorname{lm}(g)$ is not divisible by the leading monomial of any element of I_{i-1} or any element previously inserted in S_i.

Algorithm 5 sGB with recording of syzygies

Input: $f_1, \ldots, f_r \in R$
Output: See Theorem 3.5

procedure sGB($f_1, \ldots, f_r$)
  $G \leftarrow \{(f_i, (i,1), 1) \mid 1 \leq i \leq r\}$
  $S_1, \ldots, S_r \leftarrow \varnothing$
  $P \leftarrow \{(\alpha, \beta) \mid \alpha, \beta \in G \text{ form a regular S-pair}\}$
  while $P \neq \varnothing$ **do**
    $(\alpha, \beta) \leftarrow$ the element in P with minimal signature
    $P \leftarrow P \setminus \{(\alpha, \beta)\}$
    $a, b \leftarrow$ the monomials such that $a\,\mathrm{poly}(\alpha) - b\,\mathrm{poly}(\beta) = \mathrm{poly}(\mathfrak{sp}(\alpha, \beta))$
    if not Rewritable(α, a, G) and not Rewritable(β, b, G) **then**
      $\gamma \leftarrow$ RegularReduction($\mathfrak{sp}(\alpha, \beta)$, G)
      $G \leftarrow G \cup \{\gamma\}$
      if $\mathrm{poly}(\gamma) \neq 0$ **then**
        $P \leftarrow P \cup \{(\gamma, \beta) \mid \beta \in G, \mathrm{poly}(\beta) \neq 0, \text{ forms a regular S-pair with } \gamma\}$
      else *(record the quotient of the zero reduction)*
        $S_{\mathrm{ind}(\gamma)} \leftarrow S_{\mathrm{ind}(\gamma)} \cup \{\mathrm{quo}(\gamma)\}$
      end if
    end if
  end while
  return $\{\mathrm{poly}(\beta) \mid \beta \in G\}$, $S_1, \ldots, S_r$
end procedure

Proof. Termination and the first two points are a special case of [18, Theorem 7.1], where we only compute partial information about the syzygy module.

The last point is a consequence from the rewritability check. We first note that every time a polynomial h is inserted into S_i, the extended sig-poly pair $(0, (i, \operatorname{lm} h), h)$ has been inserted into G just before. (The monomial part of the signature is always the leading monomial of the quotient, this is an invariant of sig-poly pairs.) Next, in the context of line 15, if $g = \operatorname{quo}(\gamma)$, then $\mathfrak{s}(\gamma) = (\operatorname{ind}(\gamma), \operatorname{lm}(g))$. Moreover, γ comes from a S-pair $\mathfrak{sp}(\alpha, \beta)$, so $\mathfrak{s}(\gamma) = a\mathfrak{s}(\alpha)$ or $b\mathfrak{s}(\beta)$, and both REWRITABLE(α, a, G) and REWRITABLE(β, b, G) were false.

The Syzygy criterion implies that $\mathfrak{s}(\gamma)$ is not divisible by any $\mathfrak{s}(\delta)$, where $\delta \in G$ and $\operatorname{poly}(\delta) = 0$. In other words, $\operatorname{lm}(g)$ is not divisible by any $\operatorname{lm} h$, where h has been previously inserted into S_i.

The Koszul criterion implies that $\operatorname{lm}(g)$ is not divisible by any $\operatorname{lm}(\operatorname{poly}(\delta))$, where $\delta \in G$ and $\operatorname{ind}(\delta) < i$. But due to the incremental nature of the algorithm, the set $\{\operatorname{poly}(\delta) \mid \delta \in G, \operatorname{ind}(\delta) < i\}$ is a Gröbner basis of I_{i-1}. So $\operatorname{lm}(g)$ is not divisible by any element in I_{i-1}. □

3.3. The sGB tree datastructure

3.3.1. Specification

We now specify a data structure, called *sGB tree*. It is meant to extend the sGB algorithm presented above in two ways: by offering the possibility to add new input equations during the computation; and by offering the possibility to split the computation into different branches while sharing the common base.

An sGB tree represents a rooted tree T where each node holds an element of the polynomial ring R. The nodes are partially ordered by the ancestor-descendant relation: $\nu \leq_T \mu$ if ν is on the unique path from μ to the root of T (or, equivalently, if μ is in the subtree rooted at ν). For a node ν, the polynomial contained in ν is denoted $\operatorname{poly}(\nu)$, and the ideal generated by the polynomials contained by the ancestors of ν (not including ν) is denoted $I_{<\nu}$. An sGB tree offers the following three operations. How we implement them is the matter of the next section.

Node insertion Insert a new node, containing a given polynomial f, anywhere in the tree, as a new leaf or on an existing edge. Denoted INSERTNODE($\mathcal{T}$, f, position).

Gröbner basis Given a node ν, outputs a Gröbner basis of the ideal generated by the polynomials contained in the nodes $\leq_T \nu$. Denoted BASIS($\mathcal{T}$, ν).

Get a syzygy Given a node ν, outputs an element of $I_{<\nu} : \operatorname{poly}(\nu)$. Denoted GETSYZYGY($\mathcal{T}$, ν).

If GETSYZYGY($\mathcal{T}$, ν) outputs zero, then $I_{<\nu} + J = I_{<\nu} : \operatorname{poly}(\nu)$, where J is the ideal generated by all previous invocations of GETSYZYGY($\mathcal{T}$, ν).

It is guaranteed that GETSYZYGY($\mathcal{T}$, ν) eventually outputs zero after sufficiently many invocation, even if nodes are inserted or GETSYZYGY is called on other nodes in between.

3.3.2. Implementation

From the point of implementation, an sGB tree is made of:

(a) a rooted tree T whose nodes are labelled with integer IDs;

(b) a set G of extended sig-poly pairs whose indices are nodes of T (see below);

(c) a set P of pairs of elements of G forming regular S-pairs;

(d) for each node ν of T, a subset S_ν of R.

The sets G, P and S_ν have the same role as their counterparts in the sGB algorithm (Algorithm 5). The main difference is a twist in the definition of signatures and indices. In §3.1, an index (that is the first component of a signature) is a nonnegative integer. From now on, indices are nodes in T. Indices are partially ordered by the ancestor-descendant relation $\leq_T$. Note that for a given node ν, the subset $\{\mu \mid \mu \leq_T \nu\}$ is totally ordered: it is the set of nodes on the path from the root of T to ν. Lastly, we adjust the definition of a regular S-pair. We say that sig-poly pairs α and β form a regular S-pair if $\mathrm{ind}(\alpha)$ and $\mathrm{ind}(\beta)$ are comparable (that is either $\mathrm{ind}(\alpha) \leq_T \mathrm{ind}(\beta)$ or $\mathrm{ind}(\beta) \leq_T \mathrm{ind}(\alpha)$) and $\mathfrak{s}(a\alpha) \neq \mathfrak{s}(b\beta)$, with a and b as in §3.1. To analyze the behavior of the sGB tree data structure, we always consider totally ordered subsets of indices, thus reducing to the context of Algorithm 5.

To implement Basis($\mathcal{T}$, ν), we process the S-pairs with index $\leq_T \nu$. The indices of these S-pairs are totally ordered, so we are actually in the situation of §3.2 and we may apply the main loop of Algorithm 5. The body of this loop is isolated in the procedure ProcessSPair (Algorithm 6), with the appropriate alterations.

The implementation of GetSyzygy($\mathcal{T}$, ν) is similar, with the difference that we abort the computation as soon as the set S_ν is not empty and return an element of it, see Algorithm 6. If S_ν is still empty after having processed all S-pairs which may lead to new elements in S_ν, the value 0 is returned.

We assume that the state of a sGB tree always results from a sequence of calls to InsertNode, Basis or GetSyzygy applied to an initially empty tree.

Proposition 3.6. *Let $\mathcal{T}$ be a sGB tree and let ν be a node of $\mathcal{T}$.* Basis($\mathcal{T}$, ν) *(Algorithm 6) terminates and outputs a Gröbner basis of $I_{\leq\nu} := \{\mathrm{poly}(\mu) \mid \mu \leq \nu\}$.*

Proof. This algorithm considers only S-pairs whose signatures are above a given node ν. After this restriction, the signatures are totally ordered, so Basis behaves exacly like Algorithm 5 (sGB). We note that, contrary to sGB, Basis may start in a state where several S-pairs have already been processed, in an unspecified order, by earlier calls to Basis or GetSyzygy on different nodes. This does not invalidate neither the termination proof given in [20], nor the proof of correctness. □

Proposition 3.7. *Let $\mathcal{T}$ be a sGB tree and let ν be a node of $\mathcal{T}$.* GetSyzygy($\mathcal{T}$, ν) *(Algorithm 6) terminates and outputs some $f \in R$ such that:*

Algorithm 6 Implementation of the sGB tree data structure

Input: An sGB tree $\mathcal{T}$ and a label ν of T
Output: Process the pair in P with index above ν with smallest signature

procedure ProcessSPair($\mathcal{T}$, ν)
  (restrict to S-pairs whose indices are above ν)
  $P' \leftarrow \{(\alpha, \beta) \mid \alpha, \beta \in G, \max\{\mathrm{ind}(\alpha), \mathrm{ind}(\beta)\} <_T \nu\}$
  if $P' \neq \varnothing$ **then**
    $(\alpha, \beta) \leftarrow$ the pair in P' with $\mathfrak{s}(\mathfrak{s}p(\alpha, \beta))$ minimal
    $P \leftarrow P \setminus \{(\alpha, \beta)\}$
    $a, b \leftarrow$ the monomials such that $a\,\mathrm{poly}(\alpha) - b\,\mathrm{poly}(\beta) = \mathrm{poly}(\mathfrak{s}p(\alpha, \beta))$
    if not Rewritable(α, a, G) and not Rewritable(β, b, G) **then**
      $\gamma \leftarrow$ RegularReduction($\mathfrak{s}p(\alpha, \beta)$, G)
      $G \leftarrow G \cup \{\gamma\}$
      if $\mathrm{poly}(g) \neq 0$ **then**
        $P \leftarrow P \cup \{(\gamma, \beta) \mid \beta \in G, \mathrm{poly}(\beta) \neq 0,$ forms a regular S-pair with $\gamma\}$
      else *(record the quotient of the zero reduction)*
        $S_{\mathrm{ind}(\gamma)} \leftarrow S_{\mathrm{ind}(\gamma)} \cup \{\mathrm{quo}(\gamma)\}$
      end if
    end if
  end if
end procedure

Input: A sGB tree $\mathcal{T}$ and a label ν of T
Output: A Gröbner basis of $I_{<\nu}$

procedure Basis($\mathcal{T}$, ν)
  while there is a pair in P with index $\leq_T \nu$ **do**
    ProcessSPair($\mathcal{T}$, ν)
  end while
  return $\{\mathrm{poly}(\alpha) \mid \alpha \in G$ and $\mathrm{ind}(\alpha) \leq_T \nu\}$
end procedure

Input: A sGB tree $\mathcal{T}$ and a label ν of T
Output: An element of the quotient ideal $I_{<\nu} : \mathrm{poly}(\nu)$ not contained in $I_{<\nu}$

procedure GetSyzygy($\mathcal{T}$, ν)
  while there is a pair in P with index $\leq_T \nu$ **and** $S_\nu = \varnothing$ **do**
    ProcessSPair($\mathcal{T}$, ν)
  end while
  if $S_\nu \neq \varnothing$ **then**
    pick and remove some h in S_ν
    return h
  else
    return 0
  end if
end procedure

Algorithm 7 The sGB tree data structure, insertion of a node

Input: A sGB tree $\mathcal{T}$, a polynomial f and a description of the position of the new node in T

Output: The label of the newly inserted node

```
1: procedure InsertNode(𝒯, f, position)
2:     ν ← (largest label in T) + 1
3:     insert a node in T with label ν, as described by "position"
4:     S_ν ← ∅
5:     ε ← (f, (ν, 1), 1)
6:     P ← {(ε, β) | β ∈ G and (ε, β) forms a regular S-pair}
7:     G ← G ∪ {ε}
8:     return ν
9: end procedure
```

(i) $f \in I_{<\nu} : \mathrm{poly}(\nu)$*;*

(ii) if $f \neq 0$*, then* $\mathrm{lm}(f)$ *is not divisible by the leading monomial of any other polynomial previously output by* GetSyzygy$(\mathcal{T}, \nu)$*, or any polynomial in* $I_{<\nu}$*;*

(iii) if $f = 0$*, then* $I_{<\nu} : \mathrm{poly}(\nu)$ *is generated by* $I_{<\nu}$ *and the polynomials previously output by* GetSyzygy$(\mathcal{T}, \nu)$*.*

Proof. Termination follows from the termination of Basis since the main loop is similar, but with the possibility of earlier termination. Correctness follows from Theorem 3.5 after restricting to indices above ν. □

As a consequence of Proposition 3.7(ii), it is guaranteed that GetSyzygy$(\mathcal{T}, \nu)$ eventually outputs zero after sufficiently many invocation, even if nodes are inserted or GetSyzygy is called on other nodes in between. Indeed, the leading monomial of a nonzero output of GetSyzygy$(\mathcal{T}, \nu)$ is constrained to be outside the monomial ideal generated by the leading monomials of previous output. By Dickson's lemma, this may only happen finitely many times.

4. Computation of the nondegenerate locus

The sGB tree data structure can be used to implement an efficient variant of Algorithm 1 for computing the nondegenerate locus. We use an sGB tree to efficiently compute saturations $I : f^\infty$, and also double quotient $I : (I : f^\infty)$, with the idea to exploit as soon as possible newly discovered relations to simplify further computations. This leads to Algorithm 8, which we describe informally as follows.

Similarly to Algorithm 1, we introduce the equations $f_1, \ldots, f_r$ one after the other. We maintain a sGB tree which, at the beginning of the kth iteration,

that is after having processed $f_1, \ldots, f_{k-1}$, has the following shape:

$$\mathbf{g}_1 \leftarrow f_1 \leftarrow \mathbf{p}_1 \leftarrow \cdots \leftarrow \mathbf{g}_{k-1} \leftarrow f_{k-1} \leftarrow \mathbf{p}_{k-1} \leftarrow \underbrace{0}_{\nu} \begin{array}{l} \swarrow h_1 \\ \leftarrow h_2, \\ \vdots \end{array}$$

where bold letters represent a sequence of zero, one or several nodes. The tree grows from the node labeled ν, by adding new leaf nodes, or inserting nodes just above ν. Using the notations of Algorithm 1, the nodes $\mathbf{g}_i$ are related to the saturation by the f_i, the leaf nodes h_i are generic elements of the ideals in the set $\mathcal{K}$, and the nodes $\mathbf{p}_i$ are related to the cleaning steps $G : K^\infty$. We added $\mathbf{g}_1$ for the consistency of the above picture although it will always be empty and so f_1 is the root of the tree.

Remark 4.1 (Deterministic variant). The leaf nodes h_i are generic in the sense that they are linear combinations of generators of the ideals in $\mathcal{K}$ where all coefficients are either random scalars or new variables. Algorithm 8 chooses either a random scalar or a new variable in line 12. For a randomized algorithm, favoring speed over certain correctness, choose t to be a random scalar, this choice is justified by lemma 4.2. For a deterministic algorithm, choose t to be a slack variable, unused in the input equation. It is guaranteed that such a t is generic enough. In this case the monomial order on R has to be extended to $R[t]$ by a monomial order that eliminates t. The implementation discussed in the next section exclusively chooses t to be a random scalar.

The kth iteration proceeds as follows. Firstly, a new node μ containing f_k is created just above ν:

$$\cdots \leftarrow \underbrace{f_k}_{\mu} \leftarrow \underbrace{0}_{\nu} \leftarrow \cdots .$$

As long as GetSyzygy($\mathcal{T}$, μ) returns nonzero elements $(g_1, g_2, \ldots)$, we insert them above μ:

$$\cdots \leftarrow g_1 \leftarrow g_2 \leftarrow \cdots \leftarrow \underbrace{f_k}_{\mu} \leftarrow \underbrace{0}_{\nu} \leftarrow \cdots .$$

This saturation has the effect of completing $I_{<\mu}$ into $I_{<\mu} : f_k^\infty$. Each time we insert a polynomial g_i in a node, say γ, we also record the syzygies GetSyzygy($\mathcal{T}$, γ), take a generic linear combination and insert it as a new leaf node. These syzygies are related to the double quotient $I_{<\mu} : (I_{<\mu} : f_k^\infty)$. Before going to the next iteration, insert above ν all the syzygies obtained from the children of ν. Which again has the effect of saturating $I_{<\nu}$ by the polynomials contained in these nodes.

After all the input equations have been processed, the ideal $I_{<\nu}$ is a nondegenerate part of the input ideal, which we prove by comparing with Algorithm 1.

Lemma 4.2. *Let $I, J \subseteq R$ be two ideals with $J = \langle g_1, \ldots, g_u \rangle$.*

Algorithm 8 Computation of the nondegenerate locus with an sGB tree

Input: $f_1, \dots, f_c \in R$
Output: A Gröbner basis G of a nondegenerate locus of $(f_1, \dots, f_c)$

$\mathcal{T} \leftarrow$ an empty sGB tree
$\nu \leftarrow$ InsertNode($\mathcal{T}$, 0)
for k from 1 to c **do**
    $\mu \leftarrow$ InsertNode($\mathcal{T}$, f_k, just above ν)
    loop
        $g \leftarrow$ GetSyzygy($\mathcal{T}$, μ)
        if $g = 0$ **then**
            break
        end if
        $\gamma \leftarrow$ InsertNode($\mathcal{T}$, g, just above μ)
        $h \leftarrow 0$
        $t \leftarrow$ a random scalar (or the slack variable, see Remark 4.1)
        loop
            $h' \leftarrow$ GetSyzygy($\mathcal{T}$, γ)
            if $h' = 0$ **then**
                break
            end if
            $h \leftarrow th + h'$
        end loop
        InsertNode($\mathcal{T}$, h, as a child of ν)
    end loop
    for all child β of ν **do**
        loop
            $b \leftarrow$ GetSyzygy($\mathcal{T}$, β)
            if $b = 0$ **then**
                break
            end if
            InsertNode($\mathcal{T}$, b, just above ν)
        end loop
    end for
end for
return Basis(ν)

1. *If* $S = R[t_1, \ldots, t_u]$ *then* $(I : J^\infty) = (IS : (\sum_{j=1}^u t_j g_j)^\infty) \cap R$.

2. *There exists a Zarisiki-open subset* $D \subset \mathbb{K}^u$ *such that for any* $(a_1, \ldots, a_u) \in D$ *we have* $(I : J^\infty) = (I : (\sum_{j=1}^u a_j g_j)^\infty)$.

3. *If* $K \overset{\text{rad}}{=} J$ *then* $(I : K^\infty) \overset{\text{rad}}{=} (I : J^\infty)$.

Proof. (1) and (2) follows e.g. from [21, Exercise 15.41]. For (3), if $p \in R$ such that $p^k J^l \subset I$ for $k, l \in \mathbb{N}$ then for a suitably large $m \in \mathbb{N}$ we have $K^m \subseteq J^l$ so $p^k K^m \subset I$ and hence $p \in \sqrt{(I : K^\infty)}$. □

Theorem 4.3. *Algorithm 8 terminates. Algorithm 8 is correct, provided that one chooses* t *as a new variable in line 12 or that* t *is chosen as a random scalar in a suitable Zariski open subset of* $\mathbb{K}$.

Proof. Termination follows from the assumption that for any node ν of an sGB tree $\mathcal{T}$, GetSyzygy($\mathcal{T}$, ν) eventually returns 0 after sufficiently many calls.

To prove correctness, we show that Algorithm 8 computes the same ideal as Algorithm 1. Let J_{k-1} be the value of I_ν at the beginning of the kth iteration. After line 4, we also have $I_{<\mu} = J_{k-1}$, while $I_{<\nu} = I_{<\mu} + \langle f_k \rangle$.

We first examine the loop on line 5. It inserts above the node μ all the polynomials obtained from GetSyzygy($\mathcal{T}$, μ). Every node inserted on line 10 is in $I_{<\mu} : f_k$. No other node is inserted above μ. So by induction, it follows that all along the loop, we have $J_{k-1} \subseteq I_{<\mu} \subseteq J_{k-1} : f_k^\infty$. Moreover, after the loop terminates, we have $I_{<\mu} : f_k = I_{<\mu}$, due to the specification of GetSyzygy (Proposition 3.7). If now $g \in J_{k-1} : f_k^\infty$ then $g \in I_{<\mu} : f_k^\infty = I_{<\mu}$ and so all in all it follows that before line 22, we have

$$I_{<\mu} = J_{k-1} : f_k^\infty \quad \text{and} \quad I_{<\nu} = (J_{k-1} : f_k^\infty) + \langle f_k \rangle. \tag{3}$$

Next, we examine the loop on line 22 and its inner loop on line 23. By the same argument as above, the inner loop has the effect of saturating $I_{<\nu}$ by $\text{pol}(\beta)$. So after the loop on line 22, we have

$$I_{<\nu} = J_k = ((J_{k-1} : f_k^\infty) + \langle f_k \rangle) : \left(\prod_{\beta \text{ child of } \nu} \text{pol}(\beta) \right)^\infty. \tag{4}$$

It remains to understand the nature of the children of ν. They all come from the insertion of h on line 20. And h is simply a generic linear combination of the return values of GetSyzygy($\mathcal{T}$, γ). So h is a generic linear combination of some $h_1, \ldots, h_r$ such that $I_{<\gamma} + \langle h_1, \ldots, h_r \rangle = I_{<\gamma} : \text{poly}(\gamma)$ (by Proposition 3.7). For each node γ inserted on line 10, let L_γ denote the ideal $I_{<\gamma} : \text{poly}(\gamma)$. If $g_1, \ldots, g_s$ are the successive return values of GetSyzygy($\mathcal{T}$, μ) on line 6, and $\gamma_1, \ldots, \gamma_r$ the corresponding nodes, we have $L_{<\gamma_i} = I_{<\gamma_i} : g_i$ and $I_{<\gamma_i} = J_{k-1} + \langle g_1, \ldots, g_i \rangle$. By Lemma 4.4, it follows that

$$L_{\gamma_1} \cap \cdots \cap L_{\gamma_r} \overset{\text{rad}}{=} J_{k-1} : \langle g_1, \ldots, g_r \rangle^\infty. \tag{5}$$

Moreover, by (3), we obtain that before line 22

$$I_{<\mu} = J_{k-1} + \langle g_1, \ldots, g_r \rangle = J_{k-1} : f_k^\infty, \tag{6}$$

so, combining with (5),

$$\begin{aligned} L_{\gamma_1} \cap \cdots \cap L_{\gamma_r} &\overset{\text{rad}}{=} J_{k-1} : \langle g_1, \ldots, g_r \rangle^\infty && (7)\\ &= J_{k-1} : \big(J_{k-1} + \langle g_1, \ldots, g_r \rangle\big)^\infty && (8)\\ &\overset{\text{rad}}{=} J_{k-1} : (J_{k-1} : f_k^\infty). && (9) \end{aligned}$$

As remarked above, the loop on line 23 has the effect of saturating $I_{<\nu}$ by $\mathrm{pol}(\beta)$. By the analysis above, $\mathrm{pol}(\beta)$ is actually a generic linear combination of some $h_1, \ldots, h_r$ such that $I_{<\gamma} + \langle h_1, \ldots, h_r \rangle = L_\gamma$, for some node γ above ν. By Lemma 4.2, saturating by $\mathrm{pol}(\beta)$ is the same as saturating by $\langle h_1, \ldots, h_r \rangle$ (assuming that $\mathrm{pol}(\beta)$ is sufficiently generically in the case where Algorithm 8 chooses random scalars in line 12). Besides, $I_{<\nu}$ contains $I_{<\gamma}$, so saturating $I_{<\nu}$ by $\langle h_1, \ldots, h_r \rangle$ is the same as saturating by L_γ. Back to (4), we conclude from (9) that saturating $I_{<\nu}$ by all the $\mathrm{pol}(\beta)$ is the same as saturating by all the ideals $J_{i-1} : (J_{i-1} : f_i^\infty)$, for $i \leq k$.

Therefore J_k satisfies the same recurrence relation as its analogue defined the proof of Theorem 2.4:

$$J_k = \left((J_{k-1} : f_k^\infty) + \langle f_k \rangle \right) : \left(\bigcap_{i \leq k} \big(J_{i-1} : (J_{i-1} : f_i^\infty)\big) \right)^\infty. \tag{10}$$

This proves that Algorithm 8 and Algorithm 1 compute the same ideal. □

Lemma 4.4. *Let $I, J \subseteq R$ be two ideals and let $J = \langle g_1, \ldots, g_t \rangle$. Then*

$$(I : J) \overset{\text{rad}}{=} (I : g_1) \cap ((I + \langle g_1 \rangle) : g_2) \cap \cdots \cap ((I + \langle g_1, \ldots, g_{t-1} \rangle) : g_t).$$

Proof. The inclusion "$\subseteq$" is obvious. Now, let $p \in R$ be such that

$$p^m \in (I : g_1) \cap ((I + \langle g_1 \rangle) : g_2) \cap \cdots \cap ((I + \langle g_1, \ldots, g_{t-1} \rangle) : g_t)$$

for some $m \in \mathbb{N}$. Then we have in particular $p^m g_1 \in I$. Now let $i > 1$. By induction, if for some $k \in \mathbb{N}$ we have $p^k g_j \in I$ for all $j \leq i$ then

$$p^{km} g_{i+1} = p^k f + p^k a_1 g_1 + \cdots + p^k a_i g_i \in I$$

for a suitable $f \in I$, $a_1, \ldots, a_i \in R$ and so $p^{km} \in (I : g_{i+1})$. We deduce that a power of p actually lies in $(I : J)$ which ends the proof. □

5. Implementation and Experiments

5.1. Further Implementational Considerations

We start by describing some further optimizations in our implementations of Algorithms 5 and 8.

Both these implementations use an *F4-like reduction strategy*. This means that several S-pairs are selected out of the pairset at once and are subsequently, together with their regular reducers, organized in a matrix whose rows are labeled by the selected extended sig-poly pairs and whose columns are labeled by all the monomials occuring in the polynomial parts of these extended sig-poly pairs. This matrix is then put into row echelon form and the rows of this reduced matrix whose first entry has changed during the computation of this row echelon form are then processed as new basis elements or newly identified zero divisors, depending on if this reduced row is zero or not. Compared to the original F4 algorithm, one has to make sure during the computation of this row echelon form that a row is only reduced by rows whose corresponding sig-poly pair has lower signature. We refer to [23] for details on the original F4 algorithm or to [18, section 13] for a more thorough explanation as to how to combine the F4 algorithm with signature-based techniques.

For Algorithm 8, this has the consequence that the GetSyzygy routine has the ability to return several zero divisors $g_1, \ldots, g_s$ at once and Algorithm 8 may benefit from it. We implemented the following probabilistic optimization: We replaced g_1 by a random linear combination $g_1' := \sum_{j=1}^{s} a_i g_i$. Let $\nu_1, \ldots, \nu_s$ be the nodes assigned to $g_1', g_2, \ldots, g_s$ in Algorithm 8. Then, if the choice of the a_i was "sufficiently random", we know by Lemma 4.2 that for $h \in R$ we have

$$hg_1' \in I_{<\nu_1} \quad \Leftrightarrow \quad hg_i \in I_{<\nu_1} \forall i.$$

If then $\text{GetSyzygy}(\mathcal{T}, \nu_1)$ returned such an element $h \neq 0$ we regarded the signatures $(\nu_2, \text{lm}(h)), \ldots, (\nu_s, \text{lm}(h))$ as known signatures of syzygies during the calls to Rewriteable, i.e. $\text{GetSyzygy}(\mathcal{T}, \nu_i)$ would, for $i = 2, \ldots, s$, only return a non-zero result if there exists an element $h' \in (I_{<\nu_i} : g_i)$ with $\text{lm}(h')$ not divisible by $\text{lm}(h)$. Furthermore, only the zero divisors h of g_1' as above were considered in the loop from line 14-20 of Algorithm 8.

We implemented both Algorithm 5 and 8 in the programming language Julia [5] with an interface to the Singular.jl Julia-library [16]. An interface to the new computer algebra system OSCAR [1] is planned for the future. The implementation is available at

`https://github.com/RafaelDavidMohr/SignatureGB.jl`

In this implementation we use our own data structures for polynomials and polynomial arithmetic. The linear algebra routines for computing row echelon forms in our implementations closely follow the corresponding routines presented in [41]. Additionally, our implementation makes use of the modifications to Algorithm 5 presented in [19]. Currently the implementation works only for fields of finite characteristic.

While our implementation is currently not competitive with optimized implementations of Gröbner basis algorithms such as in Maple [40] or msolve [4], we do make use of some standard optimization techniques in Gröbner basis algorithm implementations such as monomial hash tables and divisor bitmasks (see e.g. [42] for a description of these techniques).

5.2. Experimental Results

We used the following examples to benchmark our implementations. All computations were made over the field $\mathbb{Z}/65521\mathbb{Z}$

1. Cyclic(8), coming from the classical Cyclic(n) benchmark.
2. Pseudo(n), encoding pseudo-singularities via polynomials

$$f_1, \ldots, f_{n-1}, g_1, \ldots, g_{n-1}$$

with $f_i \in \mathbb{K}[x_1, \ldots, x_{n-2}, z_1, z_2]$, $g_i \in \mathbb{K}[y_1, \ldots, y_{n-2}, z_1, z_2]$, the f_i being chosen as a random dense quadrics, and g_i chosen such that $g_i(x_1, \ldots, x_{n-2}, z_1, z_2) = f$, i.e. as a copy of f_i in the variables $y_1, \ldots, y_{n-2}, z_1, z_2$.

3. Sos(s, n), encoding the critical points of the restriction of the projection on the first coordinate to a hypersurface which is a sum of s random dense quadrics in $\mathbb{K}[x_1, \ldots, x_n]$.

$$f, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}, \quad f = \sum_{i=1}^{s} g_i^2.$$

4. Sing(n), encoding the critical points of the restriction of the projection on the first coordinate to a (generically singular) hypersurface which is defined by the resultant of two random dense quadrics A, B in $\mathbb{K}[x_1, \ldots, x_{n+1}]$:

$$f, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}, \quad f = \text{resultant}(A, B, x_{n+1}).$$

5. The Steiner polynomial system, coming from [7].

All these systems are generated by a number of polynomials equal to the number of variables of the underlying polynomial ring. They all have components of different dimensions, one of those being zero-dimensional, i.e. they have a nontrivial nondegenerate locus.

In Table 1 we compare Algorithm 8 and a straightforward implementation of ours of Algorithm 1 in Maple. In this implementation, we saturated an ideal J by an ideal K by picking a random linear combination p of generators of K and saturating J by p using Maple's internal saturation routine. Table 1 shows the improvement of Algorithm 8 over Algorithm 1: While Maple's highly optimized Gröbner basis engine beats our implementation of Algorithm 5 by a wide margin the ratio between the timings of our F5 implementation and our implementation of Algorithm 8 is much better than the ratio between the time it took to compute a Gröbner basis in Maple and our Maple implementation of Algorithm 1. This indicates that the overhead over a Gröbner basis computation incurred by Algorithm 1 is significantly reduced by bringing in signature-based techniques as in Algorithm 8. This can be seen by looking at the two respect "ratio"-columns of table 1. To additionally show the overhead of Algorithm 8

over Algorithm 5 we noted the number of arithmetic operations in $\mathbb{K}$ when running each of the two algorithms on the polynomial system in question. Our implementation of Algorithm 8 never takes more than 10 times the number of arithmetic operations Algorithm 5 takes, on certain examples we compare very favorably in terms of arithmetic operations to Algorithm 5.

In Table 2 we compare Algorithm 8 to other ideal decomposition methods available in the computer algebra systems Singular, Maple and Macaulay2 [30]. In Singular there is an elimination method [15] and an implementation of the algorithm for equidimensional decomposition presented in [22]. In Maple we compared against the Regular Chains package [11, 10]. In Macaulay2 one is able to compute the intersection of all components of non-minimal dimension again with the method presented in [22]. We then saturated the original ideal by the result to obtain the nondegenerate locus. On a high level, our algorithm works similarly, incrementally obtaining information about the component of higher dimension and then removing it via saturation. One should keep in mind that all of these methods, compared to Algorithm 8, work more generally: Except for what we tried in Macaulay2 they are all able to obtain a full equidimensional decomposition of the input ideal.

We gave all of these methods at least an hour for each polynomial system and at most roughly 50 times the time our implementation of Algorithm 8 took. We indicated when these times were exceeded by using ">" in Table 2. We computed all examples on a single Intel Xeon Gold 6244 CPU @ 3.60GHz with a limit of 200G memory. If this limit was exceeded, or if another segfault occured, we indicate it with 'segfault' in Table 2.

Table 1: Comparing Algorithm 1 and Algorithm 8

	Alg. 5 arith. op.	Alg. 8 arith. op.	Alg. 5	Alg. 8	Ratio	GB in Maple	Alg. 1 in Maple	Ratio
Cyclic 8	$1.2 \cdot 10^{10}$	$1.3 \cdot 10^{11}$	4m	40m	10	1.2s	154m	7700
Pseudo(2, 12)	$5.3 \cdot 10^{7}$	$3.1 \cdot 10^{8}$	1.16s	5.2s	4.5	0.268s	3.44s	13
Sing(2, 10)	$5.6 \cdot 10^{7}$	$6.5 \cdot 10^{7}$	1.9s	2.9s	1.5	0.11s	1.642s	14.5
Sing(2, 9)	$2.5 \cdot 10^{7}$	$2.9 \cdot 10^{7}$	1.1s	1.4s	1.27	0.06s	0.788s	13.1
Sos(2,5,4)	$1.3 \cdot 10^{8}$	$1.1 \cdot 10^{8}$	8.5s	7.3s	0.85	0.022s	0.479s	21.3
Sos(2,6,3)	$2.1 \cdot 10^{7}$	$2.1 \cdot 10^{7}$	1.11s	1.4s	1.26	0.021s	0.261s	12.4
Sos(2,6,4)	$4.8 \cdot 10^{9}$	$3.8 \cdot 10^{9}$	148s	169s	1.14	0.172s	22.7s	132
Sos(2,6,5)	$4.2 \cdot 10^{9}$	$2.0 \cdot 10^{9}$	75s	43s	0.57	0.458s	10.38s	22.7
Sos(2,7,3)	$1.3 \cdot 10^{8}$	$6.7 \cdot 10^{8}$	5.2s	41s	7.9	0.047s	7.162s	152.4
Sos(2,7,4)	$6.5 \cdot 10^{9}$	$4.5 \cdot 10^{10}$	3m	32m	10.7	0.433s	1h	8314
Sos(2,7,5)	$7.2 \cdot 10^{10}$	$3.5 \cdot 10^{11}$	25m	20h	48	2.294s	>359h	$> 4.4 \cdot 10^{6}$
Sos(2,7,6)	$1.7 \cdot 10^{12}$	$3.0 \cdot 10^{12}$	31h	73h	2.4	14.348s	5.5h	23
Steiner	$3.1 \cdot 10^{10}$	$2.3 \cdot 10^{11}$	4.2m	42m	10	27s	13m	28.9

Table 2: Comparing with other Decomposition Methods

	Algorithm 8	Singular: Elimination Method	Singular: Algorithm in [22]	Maple: Regular Chains	Macaulay2
Cyclic 8	40m	segfault	>35h	>35h	>35h
Pseudo(2, 10)	0.3s	40s	>1h	>1h	>1h
Pseudo(2, 12)	5.2s	>1h	>1h	>1h	>1h
Pseudo(2, 6)	0.008s	<1s	<1s	0.29s	0.07s
Pseudo(2, 8)	0.03s	<1s	23m	5.82s	13.78s
Sing(2, 10)	2.9s	>1h	>1h	>1h	>1h
Sing(2, 4)	0.02s	1s	>1h	91.32s	0.42s
Sing(2, 5)	0.07s	4s	>1h	>1h	1.94s
Sing(2, 6)	0.15s	56s	>1h	>1h	16.64s
Sing(2, 7)	0.35s	8m	>1h	>1h	289s
Sing(2, 8)	0.68s	23m	>1h	>1h	>1h
Sing(2, 9)	1.4s	>1h	>1h	>1h	>1h
Sos(2,4,2)	0.03s	<1s	<1s	19.4s	0.16s
Sos(2,4,3)	0.03s	1s	3m	14m	0.63s
Sos(2,5,2)	0.02s	<1s	>1h	>1h	0.37s
Sos(2,5,3)	0.34s	>1h	>1h	>1h	9.35s
Sos(2,5,4)	7.3s	>1h	>1h	>1h	183s
Sos(2,6,2)	0.17s	<1s	>1h	>1h	0.7s
Sos(2,6,3)	1.4s	>1h	>1h	>1h	107s
Sos(2,6,4)	169s	>140m	>140m	>140m	>140m
Sos(2,6,5)	43s	>1h	>1h	>1h	>1h
Sos(2,7,2)	2.91s	<1s	>1h	2.94s	0.18s
Sos(2,7,3)	41s	>1h	>1h	>1h	>1h
Sos(2,7,4)	32m	>26h	segfault	>26h	>26h
Sos(2,7,5)	20h	segfault	segfault	>200h	>200h
Sos(2,7,6)	73h	segfault	segfault	>334h	>500h
Steiner	42m	>50h	segfault	>50h	>50h

Acknowledgments. We wish to thank the two anonymous referees for their helpful remarks and suggestions which allowed us to improve the exposition of several results in this article.

References

[1] OSCAR – open source computer algebra research system, version 0.7.1, 2022.

[2] P. Aubry, D. Lazard, and M. Moreno Maza. On the Theories of Triangular Sets. *Journal of Symbolic Computation*, 28(1):105–124, 1999.

[3] T. Becker and V. Weispfenning. *Gröbner Bases*, volume 141 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1993.

[4] J. Berthomieu, C. Eder, and M. Safey El Din. Msolve: A library for solving polynomial systems. In *ISSAC'21*, Saint Petersburg, Russia, 2021.

[5] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[6] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997.

[7] P. Breiding, B. Sturmfels, and S. Timme. 3264 conics in a second. *Notices of the American Mathematical Society*, 67(1):30–37, 2020.

[8] M. Caboara, P. Conti, and C. Traverse. Yet another ideal decomposition algorithm. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, Lecture Notes in Computer Science, pages 39–54. Springer, 1997.

[9] C. Chen, O. Golubitsky, F. Lemaire, M. M. Maza, and W. Pan. Comprehensive triangular decomposition. In *International Workshop on Computer Algebra in Scientific Computing*, pages 73–101. Springer, 2007.

[10] C. Chen, F. Lemaire, M. M. Maza, W. Pan, and Y. Xie. Efficient computations of irredundant triangular decompositions with the regularchains library. In *International Conference on Computational Science*, pages 268–271. Springer, 2007.

[11] C. Chen and M. Moreno Maza. Algorithms for computing triangular decomposition of polynomial systems. *Journal of Symbolic Computation*, 47(6):610–642, 2012.

[12] S.-C. Chou and X.-S. Gao. Ritt-Wu's decomposition algorithm and geometry theorem proving. In *10th International Conference on Automated Deduction*, Lecture Notes in Computer Science, pages 207–220. Springer, 1990.

[13] X. Dahan, X. Jin, M. M. Maza, and E. Schost. Change of order for regular chains in positive dimension. *Theoretical Computer Science*, 392(1-3):37–65, 2008.

[14] X. Dahan, M. Moreno Maza, E. Schost, W. Wu, and Y. Xie. Lifting techniques for triangular decompositions. In *ISSAC'05*, pages 108–115. ACM, New York, 2005.

[15] W. Decker, G.-M. Greuel, and G. Pfister. Primary decomposition: Algorithms and comparisons. In *Algorithmic Algebra and Number Theory*, pages 187–220. Springer, 1999.

[16] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann. Singular 4-3-0 — A computer algebra system for polynomial computations, 2022.

[17] J. Della Dora, C. Dicrescenzo, and D. Duval. About a new method for computing in algebraic number fields. In *Research Contributions from the European Conference on Computer Algebra-Volume 2*, EUROCAL '85, pages 289–290. Springer-Verlag, 1985.

[18] C. Eder and J.-C. Faugère. A survey on signature-based algorithms for computing Gröbner bases. *Journal of Symbolic Computation*, 80:719–784, 2017.

[19] C. Eder and J. Perry. F5C: A variant of Faugère's F5 algorithm with reduced gröbner bases. *Journal of Symbolic Computation*, 45(12):1442–1458, 2010.

[20] C. Eder and B. H. Roune. Signature rewriting in Gröbner basis computation. In *Proceedings of ISSAC 2013*, pages 331–338. ACM, 2013.

[21] D. Eisenbud. *Commutative Algebra: With a View toward Algebraic Geometry*. Springer New York, New York, NY, 1995.

[22] D. Eisenbud, C. Huneke, and W. Vasconcelos. Direct methods for primary decomposition. *Inventiones Mathematicae*, 110(1):207–235, Dec. 1992.

[23] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, 139(1):61–88, 1999.

[24] J.-C. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In *ISSAC'02*, pages 75–83, 2002.

[25] S. Gao, Y. Guan, and F. Volny IV. A new incremental algorithm for computing Gröbner bases. In *ISSAC'10*, pages 13–19, 2010.

[26] S. Gao, F. Volny IV, and M. Wang. A new framework for computing Gröbner bases. *Mathematics of computation*, 85(297):449–465, 2016.

[27] P. Gianni, B. Trager, and G. Zacharias. Gröbner bases and primary decomposition of polynomial ideals. *Journal of Symbolic Computation*, 6(2):149–167, 1988.

[28] M. Giusti, G. Lecerf, and B. Salvy. A Gröbner free alternative for polynomial system solving. *Journal of Complexity*, 17(1):154–211, 2001.

[29] H.-G. Gräbe. Minimal primary decomposition and factorized Gröbner bases. *Applicable Algebra in Engineering, Communication and Computing*, 8(4):265–278, 1997.

[30] D. R. Grayson and M. E. Stillman. Macaulay2, a software system for research in algebraic geometry. Available at http://www.math.uiuc.edu/Macaulay2/.

[31] G.-M. Greuel and G. Pfister. *A Singular Introduction to Commutative Algebra*. Springer Berlin Heidelberg, second edition, 2007.

[32] E. Hubert. Notes on triangular sets and triangulation-decomposition algorithms I. In *Symbolic and Numerical Scientific Computation*, Lecture Notes in Computer Science, pages 1–39. Springer, 2003.

[33] Y. Ishihara and K. Yokoyama. Effective localization using double ideal quotient and its implementation. In V. P. Gerdt, W. Koepf, W. M. Seiler, and E. V. Vorozhtsov, editors, *Computer Algebra in Scientific Computing*, Lecture Notes in Computer Science, pages 272–287. Springer, 2018.

[34] G. Jeronimo and J. Sabia. Effective equidimensional decomposition of affine varieties. *Journal of Pure and Applied Algebra*, 169(2):229–248, Apr. 2002.

[35] T. Krick and A. Logar. An algorithm for the computation of the radical of an ideal in the ring of polynomials. In *AAECC 1991*, pages 195–205. Springer-Verlag, 1991.

[36] G. Lecerf. Computing an equidimensional decomposition of an algebraic variety by means of geometric resolutions. In *ISSAC'00*, pages 209–216, 2000.

[37] G. Lecerf. Computing the equidimensional decomposition of an algebraic closed set by means of lifting fibers. *Journal of Complexity*, 19(4):564–596, Aug. 2003.

[38] F. Lemaire, M. Moreno Maza, W. Pan, and Y. Xie. When does $\langle \mathrm{T} \rangle$ equal sat(T)? *Journal of Symbolic Computation*, 46(12):1291–1305, 2011.

[39] X. Li, M. Moreno Maza, and W. Pan. Computations modulo regular chains. In *ISSAC'09*, pages 239–246. ACM, New York, 2009.

[40] Maplesoft, a division of Waterloo Maple Inc.. Maple, 2021.

[41] M. Monagan and R. Pearce. A compact parallel implementation of F4. In *Proceedings of the 2015 International Workshop on Parallel Symbolic Computation*, pages 95–100, 2015.

[42] B. H. Roune and M. Stillman. Practical gröbner basis computation. In *ISSAC'12*, pages 203–210, 2012.

[43] W. Vasconcelos. *Computational Methods in Commutative Algebra and Algebraic Geometry*. Algorithms and Computation in Mathematics. Springer-Verlag, 1998.

[44] D. Wang. An elimination method for polynomial systems. *Journal of Symbolic Computation*, 16(2):83–114, Aug. 1993.

[45] D. Wang. *Elimination Methods*. Texts and Monographs in Symbolic Computation. Springer Vienna, 2001.

[46] W.-T. Wu. Basic principles of mechanical theorem proving in elementary geometries. 2(3):221–252, Sept. 1986.