Standard Bases over Euclidean Domains

Christian Eder

TU Kaiserslautern
Department of Mathematics
D-67663 Kaiserslautern

Gerhard Pfister

TU Kaiserslautern
Department of Mathematics
D-67663 Kaiserslautern

Adrian Popescu

TU Kaiserslautern
Department of Mathematics
D-67663 Kaiserslautern

Abstract

In this paper we state and explain techniques useful for the computation of strong Gröbner and standard bases over Euclidean domains: First we investigate several strategies for creating the pair set using an idea by Lichtblau. Then we explain methods for avoiding coefficient growth using syzygies. We give an in-depth discussion on normal form computation resp. a generalized reduction process with many optimizations to further avoid large coefficients. These are combined with methods to reach GCD-polynomials at an earlier stage of the computation. Based on various examples we show that our new implementation in the computer algebra system Singular is, in general, more efficient than other known implementations.

Keywords: Gröbner bases, Standard Bases, Euclidean Domains, Algorithms

1. Introduction

In 1964 Hironaka already investigated computational approaches towards singularities and introduced the notion of standard bases for local monomial orders,[1] see, for example, Hironaka, H. (1964a,b); Grauert, H. (1972). In Buchberger, B. (1965, 2006), Buchberger initiated, in 1965, the theory of Gröbner bases for global monomial orders by which many fundamental problems in mathematics, science and engineering can be

---

[1]See Definition 1.

solved algorithmically. Specifically, he introduced some key structural theory, and based on this theory, proposed the first algorithm for computing Gröbner bases. Buchberger's algorithm introduced the concept of critical pairs and repeatedly carries out a certain polynomial operation (called reduction).

Many of those reductions would be determined as "useless" (i.e. no contribution to the output of the algorithm), but only a posteriori, that is, after an (often expensive) reduction process. Thus intensive research was carried out, starting with Buchberger, to avoid the useless reductions via a priori criteria, see, for example, Buchberger, B. (1979, 1985); Gebauer, R. and Möller, H. M. (1988).

Once the underlying structure is no longer a field, one needs the notion of strong Gröbner bases resp. strong standard bases. Influential work was done by Kandri-Rody, A. and Kapur, D. (1988), introducing the first generalization of Buchberger's algorithm over Euclidean domains computing strong Gröbner bases. Moreover, Pan introduced the notion of D-bases of polynomial ideals over principal ideal domains, see Pan (1989). Since then only a few optimizations have been introduced, see, for example, Wienand (2011); Lichtblau (2012); Eder, C. et al. (2017).

In this paper we introduce several new optimizations to the computation of strong standard bases over Euclidean domains. In Section 2 we give the basic notation and introduce the idea of a reduction step, generalized from the field case. We state Buchberger's algorithm over Euclidean domains for global and also for local monomial orders. Section 3 discusses different variants of how to handle S-polynomials and GCD-polynomials, especially generalized variants of Buchberger's product and chain criterion. Over Euclidean domains like the integers, coefficient swell and the missing normalization of the lead coefficient play an important role when it comes to practical and efficient computation. Modular computation are not possible in general, but we give a new attempt for keeping coefficients small in Section 4. In Section 5 we finally give an in-depth discussion on the normal form computation which provides various attempts like lead term reductions and lead coefficient reductions. This, again, helps to keep coefficients small and minimizes the number of polynomials in a basis which have the same leading monomial by applying efficient gcd computation. We have done a new implementation of Buchberger's algorithm in the computer algebra system Singular. In Section 6 we compare our implementation with Macaulay2 and Magma, exploring the impact of the above ideas with some interesting results.

2. Basic notations

Let $\mathcal{R}$ be a Euclidean domain.[2] A polynomial in n variables $x_1, \ldots, x_n$ over $\mathcal{R}$ is a finite $\mathcal{R}$-linear combination of terms $a_{v_1,\ldots,v_n} \prod_{i=1}^n x_i^{v_i}$, $f = \sum_v a_v x^v := \sum_{v \in \mathbb{N}^n}^{\text{finite}} a_{v_1,\ldots,v_n} \prod_{i=1}^n x_i^{v_i}$, such that $v \in \mathbb{N}^n$ and $a_v \in \mathcal{R}$. The polynomial ring $\mathcal{P} := \mathcal{R}[x] := \mathcal{R}[x_1, \ldots, x_n]$ in n variables over $\mathcal{R}$ is the set of all polynomials over $\mathcal{R}$ together with the usual addition and multiplication. For $f = \sum_v a_v x^v \neq 0 \in \mathcal{P}$ we define the degree of f by $\deg(f) := \max \{v_1 + \cdots + v_n \mid a_v \neq 0\}$. For $f = 0$ we set $\deg(f) := -1$.

Let $(f_1, \ldots, f_m) \in \mathcal{P}$ be a finite sequence of polynomials. We define a module homomorphism $\pi : \mathcal{P}^m \to \mathcal{P}$ by $e_i \mapsto f_i$ for all $1 \leq i \leq m$. We use the shorthand notation

---

[2]The reader can feel free to think of $\mathcal{R} = \mathbb{Z}$.

$\overline{\alpha} := \pi(\alpha) \in \mathcal{P}$ for $\alpha \in \mathcal{P}^m$. An element $\alpha \in \mathcal{P}^m$ with $\overline{\alpha} = 0$ is called a syzygy. The module of all syzygies (of $\langle f_1, \ldots, f_m \rangle$) is denoted $\mathrm{syz}(\langle f_1, \ldots, f_m \rangle)$.

In the following we discuss computation with respect to different monomial orders:

Definition 1. Let $<$ denote a monomial order on $\mathcal{P}$.
1. $<$ is called global if $x^\alpha \geq 1$ for all $\alpha \in \mathbb{N}^n$.
2. $<$ is called local if $x^\alpha \leq 1$ for all $\alpha \in \mathbb{N}^n$.

Given such a monomial order $<$ we can highlight the maximal terms of elements in $\mathcal{P}$ w.r.t. $<$: for $f \in \mathcal{P} \setminus \{0\}$, $\mathrm{lt}(f)$ is the lead term, $\mathrm{lm}(f)$ the lead monomial, and $\mathrm{lc}(f)$ the lead coefficient of f. For any set $F \subset \mathcal{P}$ we define the lead ideal $L(F) = \langle \mathrm{lt}(f) \mid f \in F \rangle$; for an ideal $I \subset \mathcal{P}$, $L(I)$ is defined as the ideal of lead terms of all elements of I. Moreover, we define the ecart of f by $\mathrm{ecart}(f) := \deg(f) - \deg(\mathrm{lm}(f))$.

Working over a field there are many equivalent definitions of how to obtain a canonical or normal form when reducing a given polynomial by a Gröbner basis G. Working over more general rings these definitions are no longer equivalent and over Euclidean domains, like the integers, this, in particular, results in the term of being "strong", see Definition 4:

Assuming that our coefficient ring $\mathcal{R}$ is an Euclidean domain we can define a total order $\prec$ using the Euclidean norm $|\cdot|$ of its elements: Let $a_1, a_2 \in \mathcal{R}$, then $a_1 \prec a_2$ if $|a_1| < |a_2|$. For example, for the integers we can use the absolute value and break ties via sign:

$$0 \prec -1 \prec 1 \prec -2 \prec 2 \prec -3 \prec 3 \prec \ldots$$

The reduction process of two polynomials f and g in $\mathcal{P}$ depends now on the uniqueness of the minimal remainder in the division algorithm in $\mathcal{R}$:

Definition 2. Let $f, g \in \mathcal{P}$ and let $G = \{g_1, \ldots, g_r\} \subset \mathcal{P}$ be a finite set of polynomials.
1. We say that g top-reduces f if $\mathrm{lm}(g) \mid \mathrm{lm}(f)$ and if there exist $a, b \in \mathcal{R}$ such that $\mathrm{lc}(f) = a\,\mathrm{lc}(g) + b$ such that $a \neq 0$ and $b \prec \mathrm{lc}(f)$. The top-reduction of f by g is then given by
$$f - a\frac{\mathrm{lm}(f)}{\mathrm{lm}(g)}g.$$

   So a top-reduction takes place if the reduced polynomial will have either a smaller lead monomial or a smaller lead coefficient.
2. Relaxing the reduction of the lead term to any term of f, we say that g reduces f. In general, we speak of the reduction of a polynomial f w.r.t. a finite set $F \subset \mathcal{P}$.
3. We say that f has a weak standard representation w.r.t. G if $f = \sum_{i=1}^r h_i g_i$ for some $h_i \in \mathcal{P}$ such that $\mathrm{lm}(f) = \mathrm{lm}(h_j g_j)$ for some $j \in \{1, \ldots, r\}$.
4. We say that f has a strong standard representation w.r.t. G if $f = \sum_{i=1}^r h_i g_i$ for some $h_i \in \mathcal{P}$ such that $\mathrm{lm}(f) = \mathrm{lm}(h_j g_j)$ for some $j \in \{1, \ldots, r\}$ and $\mathrm{lm}(f) > \mathrm{lm}(h_k g_k)$ for all $k \neq j$.

This kind of reduction is equivalent to CP3 from Kandri-Rody and Kapur (1984) and generalizes Buchberger's attempt from Buchberger, B. (1985).

The result of such a reduction might not be unique. This uniqueness is exactly the property standard bases give us. Before defining standard bases, let us give a short note on the naming convention in this paper:

Convention 3. Note that the term Gröbner basis was introduced by Buchberger in 1965 for bases w.r.t. a global monomial order (Buchberger, B. (1965, 2006)). Independently, Hironaka (Hironaka, H. (1964a,b)), and, again independently, Grauert (Grauert, H. (1972)), developed an analogous concept, called standard basis, for multivariate power series, i.e. for polynomial rings equipped with a local monomial order. For this paper we decided to use the notion standard basis since it is nowadays the more general one.

Definition 4. A finite set $G \subset \mathcal{P}$ is called a standard basis for an ideal I w.r.t. a monomial order $<$ if $G \subset I$ and $L(G) = L(I)$. Furthermore, G is called a strong standard basis if for any $f \in I \backslash \{0\}$ there exists a $g \in G$ such that $\mathrm{lt}\,(g) \mid \mathrm{lt}\,(f)$.

Remark 5. Note that G being a strong standard basis is equivalent to all elements $g \in G$ having a strong standard representation w.r.t. G. See, for example, Theorem 1 in Lichtblau (2012) for a proof.

Clearly, assuming the field case, any standard basis is a strong standard basis. But in our setting with $\mathcal{R}$ being an Euclidean ring one has to check the coefficients, too, as explained in Definition 2.

Example 6. Let $\mathcal{R} = \mathbb{Z}$ and $I = \langle x \rangle \in \mathcal{R}[x]$. Clearly, $G := \{2x, 3x\}$ is a standard basis for I: $L(I) = \langle x \rangle$ and $x = 3x - 2x \in L(G)$. But G is not a strong standard basis for I since $2x \nmid x$ and $3x \nmid x$.

In order to compute strong standard bases we need to consider two different types of special polynomials:

Definition 7. Let $f, g \in \mathcal{P}$. We assume w.l.o.g. that $\mathrm{lc}\,(f) < \mathrm{lc}\,(g)$. Let $t = \mathrm{lcm}\,(\mathrm{lm}\,(f), \mathrm{lm}\,(g))$, $t_f = \frac{t}{\mathrm{lm}(f)}$, and $t_g = \frac{t}{\mathrm{lm}(g)}$.

1. Let $a = \mathrm{lcm}\,(\mathrm{lc}\,(f), \mathrm{lc}\,(g))$, $a_f = \frac{a}{\mathrm{lc}(f)}$, and $a_g = \frac{a}{\mathrm{lc}(g)}$. The S-polynomial of f and g is denoted

$$\mathrm{spoly}\,(f, g) = a_f t_f f - a_g t_g g.$$

2. Let $b = \gcd\,(\mathrm{lc}\,(f), \mathrm{lc}\,(g))$. Choose $b_f, b_g \in \mathcal{R}$ such that $b = b_f \mathrm{lc}\,(f) + b_g \mathrm{lc}\,(g)$.[3] The GCD-polynomial of f and g is denoted

$$\mathrm{gpoly}\,(f, g) = b_f t_f f + b_g t_g g.$$

Remark 8.
1. In the field case we do not need to consider GCD-polynomials at all since we can always normalize the polynomials, i.e. ensure that $\mathrm{lc}\,(f) = 1$.
2. Note that $\mathrm{gpoly}\,(f, g)$ is not uniquely defined: Working over $\mathcal{R} = \mathbb{Z}$ we know that we can write $\langle \mathrm{lc}\,(f), \mathrm{lc}\,(g) \rangle$ as a principal ideal, say $\langle c \rangle = \langle \mathrm{lc}\,(f), \mathrm{lc}\,(g) \rangle$ for some $c \in \mathcal{R}$. Then there exist $c_f \neq c'_f$, $c_g \neq c'_g \in \mathcal{R}$ such that

$$c_f \mathrm{lc}\,(f) + c_g \mathrm{lc}\,(g) = c = c'_f \mathrm{lc}\,(f) + c'_g \mathrm{lc}\,(g).$$

Depending on the implementation of the gcd algorithm one specific choice is made for each GCD-polynomial.

---

[3]Since $\mathcal{R}$ is an Euclidean ring the extended gcd always exists.

From Example 6 it is clear that the usual Buchberger algorithm as in the field case will not compute a strong standard basis as we would only consider $\mathrm{spoly}\,(2x, 3x) = 3 \cdot 2x - 2 \cdot 3x = 0$. Luckily, we can fix this via taking care of the corresponding GCD-polynomial:

$$\mathrm{gpoly}\,(2x, 3x) = (-1) \cdot 2x - (-1) \cdot 3x = x.$$

It follows that given an ideal $I \subset \mathcal{P}$ a strong standard basis for I can now be computed using Buchberger's algorithm taking care not only of S-polynomials but also of GCD-polynomials. We refer, for example, to Lichtblau (2012) for more details.

In Algorithm 3 we give pseudo code for a generic Buchberger algorithm over the integers. Here, no criterion for detecting useless elements is applied. This is the topic of the next section. But what is necessary to discuss beforehand is how to get strong standard representations of elements handled by Algorithm 3. This is the concept of a normal form.

Definition 9. Let $<$ be a monomial order on $\mathcal{P}$. Let $\mathcal{G}$ denote the set of all finite subsets $G \subset \mathcal{P}$. We call the map

$$
\begin{aligned}
\mathrm{NF} : \quad \mathcal{P} \times \mathcal{G} \quad &\longrightarrow \quad \mathcal{P} \\
(f, G) \quad &\longmapsto \quad \mathrm{NF}\,(f, G),
\end{aligned}
$$

a weak normal form w.r.t. $<$ if for all $f \in \mathcal{P}$ and all $G \in \mathcal{G}$ the following hold:
1. $\mathrm{NF}\,(0, G) = 0$.
2. If $\mathrm{NF}\,(f, G) \neq 0$ then $\mathrm{lt}\,(\mathrm{NF}\,(f, G)) \notin L(G)$.
3. If $f \neq 0$ then there exists a unit $u \in \mathcal{P}$ such that either $uf = \mathrm{NF}\,(f, G)$ or $r = uf - \mathrm{NF}\,(f, G)$ has a strong standard representation w.r.t. G.

A weak normal form NF is called a normal form if we can always choose $u = 1$.

Next we give algorithms that compute normal forms. For their correctness we refer to Section 1.6 in Greuel, G.-M. and Pfister, G. (2007). Algorithm 1 presents a normal form algorithm for computation w.r.t. a global monomial order $<$:

Algorithm 1 Normal form w.r.t. a global monomial order $<$ (NF)

Input: Polynomial $f \in \mathcal{P}$, finite subset $G \subset \mathcal{P}$
Output: NF of f w.r.t. G and $<$
1: $h \leftarrow f$
2: while ($h \neq 0$ and $G_h := \{g \in G \mid g \text{ top-reduces } h\} \neq \emptyset$) do
3:      Choose $g \in G_h$.
4:      $h \leftarrow$ Top-reduction of h by g (see Definition 2)
5: end while
6: return h

Note that Algorithm 1 may enter an infinite while loop if applied to local monomial orders. Let us illustrate this behaviour with the "standard" example.

Example 10. Let $\mathcal{P} = K[x]$ where K is a field. We equip $\mathcal{P}$ with a local monomial order $<$, i.e. $x < 1$. We set $G = \{g\}$ where $g = x - x^2$ and we want to compute $\mathrm{NF}\,(f, G)$ where $f = x$. Using Algorithm 1 we start setting $h := f$ and find that g top-reduces h:

$$h := x - \left(x - x^2\right) = x^2.$$

Now we can again top-reduce h via subtracting xg:

$$h := x^2 - x(x - x^2) = x^3.$$

This process does not stop, but constructs a power series equation: $x - \left(\sum_{i=0}^{\infty} x^i\right)(x - x^2) = 0$. Since $x < 1$ we know that $\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$. We see that Algorithm 1 computes correctly since $(1 - x)x = x - x^2$, still, it is not able to find the finite expression of the power series. The main fact is, that Algorithm 1 does not take into account the ecart of the polynomials: $\mathrm{ecart}(f) = 0$ whereas $\mathrm{ecart}(g) = 1$. Thus, once we reduce f by g the generated $h = x^2$ has $\mathrm{ecart}(h) = 0$, but the degree grows. If we would be able to reduce h not only by g, but also by the initial $f = x$, then we could terminate the reduction process: $h - xf = x^2 - x \cdot x = 0$. All in all, we would get the desired representation $(1 - x)x = x - x^2$.

In Mora, T. (1982) Mora gave the first attempt to achieve a terminating normal form algorithm also for local monomial orders taking into account the ecart of the reducer polynomials:

Algorithm 2 Mora's normal form algorithm w.r.t. a local monomial order $<$ (NF)

Input: Polynomial $f \in \mathcal{P}$, finite subset $G \subset \mathcal{P}$
Output: NF of f w.r.t. G and $<$
  1: $h \leftarrow f$
  2: $T \leftarrow G$
  3: **while** ($h \neq 0$ and $T_h := \{g \in T \mid g \text{ top-reduces } h\} \neq \emptyset$) **do**
  4:      Choose $g \in T_h$ with $\mathrm{ecart}(g)$ minimal.
  5:      **if** ($\mathrm{ecart}(g) > \mathrm{ecart}(h)$) **then**
  6:          $T \leftarrow T \cup \{h\}$
  7:      **end if**
  8:      $h \leftarrow$ Top-reduction of h by g (see Definition 2)
  9: **end while**
 10: **return** h

Note that Algorithm 2 in general computes a weak normal form for f, i.e. u might not be trivial. See $u = 1 - x$ in Example 10.

Example 11. Let $\mathcal{P} = \mathbb{Z}[x, y]$. A strong standard basis for the ideal $I = \langle 6 + y + x^2, 4 + x \rangle \subset \mathcal{P}$ w.r.t. negative degree reverse lexicographical order $<$ is given by

$$G = \left\{2 - x + y + x^2, x - 2y - x^2 - xy - x^3\right\}.$$

Since $4 + x \in I$ we assume that $\mathrm{NF}(4 + x, G) = 0$. In Table 1 we state Mora's normal form computation with notation as in Algorithm 2, i.e. if we have a choice for the reducer, we take the one of minimal possible ecart. We start with $h = 4 + x$ and $T_h = G$.

Note the importance of $4 + x$ being added to T_h. Without $4 + x$ as reducer the reduction process would not terminate.

Remark 12. Sometimes it can be more efficient to not only add the current status of h to T_h as a new reducer, but also to generate GCD-polynomials with h. Doing so, several lead coefficient reductions may be done in one step. See Example 25 in Section 5 for more details.

h	g	h added to T_h?
$4 + x$	$2 - x + y + x^2$	✓
$3x - 2y - 2x^2$	$4 + x$	-
$-x - 2y - 3x^2$	$x - 2y - x^2 - xy - x^3$	✓
$-4y - 4x^2 - xy - x^3$	$4 + x$	-
$-4x^2 - x^3$	$4 + x$	-
0	-	-

Table 1: Weak normal form computation due to Mora

Now we are ready to state Buchberger's algorithm. For the theoretical background of the algorithm (Buchberger's criterion) we refer to Theorem 17 in Section 3.

Algorithm 3 Buchberger's algorithm for computing strong standard bases (sBBA)

Input: Ideal $I = \langle f_1, \ldots, f_m \rangle \subset \mathcal{P}$, monomial order $<$, normal form algorithm NF (depending on $<$)
Output: Gröbner basis G for I w.r.t. $<$
1: $G \leftarrow \{f_1, \ldots, f_m\}$
2: $P \leftarrow \left\{ \mathrm{spoly}\left(f_i, f_j \right), \mathrm{gpoly}\left(f_i, f_j \right) \mid 1 \le i < j \le m \right\}$
3: while $(P \ne \emptyset)$ do
4:     Choose $h \in P$, $P \leftarrow P \setminus \{h\}$
5:     $h \leftarrow \mathrm{NF}\,(h, G)$
6:     if $(h \ne 0)$ then
7:         $P \leftarrow P \cup \{\mathrm{spoly}\,(g, h), \mathrm{gpoly}\,(g, h) \mid g \in G\}$
8:         $G \leftarrow G \cup \{h\}$
9:     end if
10: end while
11: return G

---

3. How to choose Pairs

Having two classes of polynomials to handle, namely S-polynomials and GCD-polynomials we also need criteria for deciding when such a polynomial is useless in the sense of predicting a zero reduction or having already a strong standard representation. The criteria presented in the following are then applied to Algorithm 3 in lines 2 and 7 when new elements for the pair set are generated.

A first criterion takes care of useless GCD-polynomials and goes back to Pan:

Lemma 13 (Pan (1989)). Let $f, g \in \mathcal{P}$ such that $\mathrm{lc}\,(f) \mid \mathrm{lc}\,(g)$ or $\mathrm{lc}\,(g) \mid \mathrm{lc}\,(f)$. Then $\mathrm{gpoly}\,(f, g)$ reduces to zero w.r.t. $\{f, g\}$.

Proof. Since $\gcd\,(\mathrm{lc}\,(f), \mathrm{lc}\,(g)) = \mathrm{lc}\,(f)$ we can choose $b_f = 1$ and $b_g = 0$. Thus $\mathrm{gpoly}\,(f, g) = 1 \cdot t_f \cdot f - 0 \cdot t_g \cdot g = t_f f$ for monomial multiples t_f, t_g such that $t_f \mathrm{lm}\,(f) = t_g \mathrm{lm}\,(g) =$

$\text{lcm}(\text{lm}(f), \text{lm}(g))$. It follows that $\text{gpoly}(f, g)$ just a multiple of f and thus reduces to zero w.r.t. $\{f, g\}$. $\qquad \square$

As a next step we state well-known criteria by Buchberger, the Product and the Chain criterion:

Lemma 14 (Buchberger's Product Criterion). Let $f, g \in \mathcal{P}$ be polynomials such that $\text{lm}(f)$ and $\text{lm}(g)$ are coprime and $\text{lc}(f)$ and $\text{lc}(g)$ are coprime. Then $\text{spoly}(f, g)$ reduces to zero w.r.t. $\{f, g\}$

Proof. The proof is the same as in the field case. See Theorem 11 in Lichtblau (2012) for more details. $\qquad \square$

Note that Lemma 14 does not apply to GCD-polynomials. Take, for example, $G = \{f, g\}$ with $f = 3x$ and $g = 2y$. Then $\text{lm}(f) = x$ and $\text{lm}(g) = y$ are coprime and $\text{spoly}(f, g) = gf - fg = 0$, but $\text{gpoly}(f, g) = y \cdot 3x - x \cdot 2y = xy$. Clearly, we cannot reduce xy any further w.r.t. G.

Lemma 15 (Buchberger's Chain Criterion). Let $G \subset \mathcal{P}$ finite and let $f, g, h \in G$ be polynomials such that
   1. $\text{lm}(f) \mid \text{lcm}(\text{lm}(g), \text{lm}(h))$ and
   2. $\text{lc}(f) \mid \text{lcm}(\text{lc}(g), \text{lc}(h))$.
If $\text{spoly}(f, g)$ and $\text{spoly}(f, h)$ have a strong standard representation w.r.t. G then also $\text{spoly}(g, h)$ has a strong standard representation w.r.t. G.

Proof. The lemma is proven as in the field case. We want to have an S-polynomial chain

$$\text{spoly}(g, h) = c_{f,g} m_{f,g} \text{spoly}(f, g) + c_{f,h} m_{f,h} \text{spoly}(f, h)$$

for some coefficients $c_{f,g}, c_{f,h}$ and some monomials $m_{f,g}, m_{f,h}$. Property (1) ensures proper monomial multiples $m_{f,g}$ and $m_{f,h}$. Working over the integers we have to take care of the coefficients, too. Thus property (2) is needed to ensure the existence of proper multiples $c_{f,g}$ and $c_{f,h}$. For more details see, for example, the proof of Theorem 12 in Lichtblau (2012): There the statement is proven for a strengthened version of property (2), namely $\text{lc}(f) \mid \text{lc}(g) \mid \text{lc}(h)$ resp. $\text{lc}(g) \mid \text{lc}(f) \mid \text{lc}(h)$. $\qquad \square$

Moreover, we can state a similar criterion for GCD-polynomials:

Lemma 16. Let $G \subset \mathcal{P}$ finite and let $f, g, h \in G$ be polynomials such that
   1. $\text{lm}(f) \mid \text{lcm}(\text{lm}(g), \text{lm}(h))$ and
   2. $\text{lc}(f) \mid \gcd(\text{lc}(g), \text{lc}(h))$.
Then $\text{gpoly}(g, h)$ has a strong standard representation w.r.t. G.

Proof. See Theorem 10.11 in Becker, T. et al. (1993) and Theorem 13 in Lichtblau (2012). $\qquad \square$

Besides these statements one can "decide" for two polynomials f and g if we need to compute the corresponding GCD-polynomial or the corresponding S-polynomial. This goes back to Kandri-Rody, A. and Kapur, D. (1988) and Pan (1989) and can be also found as a variant of Buchberger's criterion as Theorem 10 in Lichtblau (2012) resp. in Mora (2016).

Theorem 17 (Variant of Buchberger's Criterion). Let $G \subset \mathcal{P}$ be a finite set of polynomials, let $<$ be a monomial order on $\mathcal{P}$. The following are equivalent:

1. G is a strong standard basis w.r.t. $<$.
2. For all $f, g \in G$ $\operatorname{spoly}(f, g)$ and $\operatorname{gpoly}(f, g)$ reduce to zero.
3. For all $f, g \in G$ the following hold:
    (a) If $\operatorname{lc}(f) \mid \operatorname{lc}(g)$ or $\operatorname{lc}(g) \mid \operatorname{lc}(f)$ then $\operatorname{spoly}(f, g)$ reduces to zero.
    (b) If $\operatorname{lc}(f) \nmid \operatorname{lc}(g)$ and $\operatorname{lc}(g) \nmid \operatorname{lc}(f)$ then $\operatorname{gpoly}(f, g)$ reduces to zero.

Clearly, when implementing the above criteria especially the choice between Condition 2 and Condition 3 in Theorem 17 has a huge influence on the computation:

1. Depending the choice of the next element in the pair set in Algorithm 3 it is obvious that Condition 3 places an emphasis on GCD-polynomials. For a pair of polynomials $f, g \in G$ the algorithm tries to keep of the lead coefficient of the generated polynomial as small as possible. This process goes on until at some point eventually this smaller lead coefficient divides $\operatorname{lc}(f) \operatorname{lc}(g)$. Then the corresponding S-polynomial is generated which then removes the whole lead term.
2. If we use Condition 2 then there might be a lead term cancellation, i.e. S-polynomial, being handled before the complete reduction process of the lead coefficient, i.e. handling of GCD-polynomials, is finished.

Of course, one can have an influence on the above situation depending on the choice of the next element from the pair set P in Line 4 of Algorithm 3. Lichtblau notes in Lichtblau (2012) that, until now, there is no real comparison between the two attempts due to missing implementations.

One statement we can make is the following:

Proposition 18. Theorem 17 already includes Lemma 14.

Proof. By Theorem 17 we only consider $\operatorname{spoly}(f, g)$ if either $\operatorname{lc}(f) \mid \operatorname{lc}(g)$ or $\operatorname{lc}(g) \mid \operatorname{lc}(f)$. Lemma 14 applies only in the other situations, but there no S-polynomial is generated at all. $\qquad\square$

We have implemented Buchberger's algorithm with both variants of Buchberger's criterion in Singular. In Section 6 we present more detailed results. Note that in both cases we implemented the criteria to remove useless pairs in the vein of the Gebauer-Möller installation, we refer to Gebauer, R. and Möller, H. M. (1988) as well as its reformulation in Mora (2005) resp. Ceria et al. (2019) for more details.

4. Avoiding Coefficient Swell

One main problem when computing over the integers is coefficient growth. We cannot normalize polynomials as usually done over fields. The only method to keep at least lead coefficients as small as possible from inside the algorithm is to efficiently compute GCD-polynomials as discussed in Section 3.

The first idea to handle coefficient swell might be to use modular methods as done over fields, see, for example, Arnold, E. A. (2003). Sadly this concept is not working over the integers.

Example 19. Let $I = \langle 6x, 8x \rangle \subset \mathbb{Z}[x]$. A strong standard basis w.r.t. a global monomial order $<$ is $G = \{2x, 6x, 8x\}$ where $2x = \mathrm{gpoly}(8x, 6x)$. Next we try to compute modular standard bases for $\langle 6x, 8x \rangle \subset \mathbb{F}_p[x]$ for some prime number $p > 3$.[4] The corresponding standard basis is $G_p = \{6x, 8x\}$: Buchberger's algorithm over $\mathbb{F}_p$ only considers $\mathrm{spoly}(8x, 6x) = 0$ and terminates afterwards, thus only the initial generators are added to G_p. If we ensure that Buchberger's algorithm computes a reduced[5] standard basis we then get $G_p = \{x\}$. The problem is that in no case we would get $2x \in G_p$ which is the important element for the strongness of the standard basis for I over $\mathbb{Z}$. Even before applying Hensel lifting or the Chinese remainder theorem the information needed is lost.

Thus there is no way to compute a strong standard basis over $\mathbb{Z}$ via modular computations over $\mathbb{F}_p$ and lifting techniques in general.

One trick we can do is trying to find monomials or constants in the ideal we want to compute a strong standard basis for. If we can add such elements to the list of input polynomials of Algorithm 3 this can give a huge speed up to the overall computation. The following lemma gives us a hint on how to do so.

Lemma 20. Let $<$ be a monomial order on $\mathbb{Z}[x]$ and let $I = \langle f_1, \ldots, f_m \rangle \subset \mathbb{Z}[x]$. Let $\tilde{I} = \langle f_1, \ldots, f_m \rangle \subset \mathbb{Q}[x]$. If the standard basis $G = \langle 1 \rangle$ for $\tilde{I}$ w.r.t. $<$ then there exists a constant $c \in \mathbb{Z}$ such that $c \in I$.

Proof. Let $G = \langle 1 \rangle \subset \mathbb{Q}[X]$. Consider $J = \langle 1, f_1, \ldots, f_m \rangle \subset \mathbb{Q}[x]$. Consider the free module $\mathbb{Q}[x]^{m+1}$ with standard generators $e_0, \ldots, e_m$ together with the map $\pi : \mathbb{Q}[x]^{m+1} \to \mathbb{Q}[x]$ defined via $e_0 \mapsto 1$ and $e_i \mapsto f_i$ for all $1 \le i \le m$. Since $1 \in G$ there must exist a syzygy $\sigma \in \mathbb{Q}[x]^{m+1}$ of the following structure

$$\sigma = e_0 + \sum_{i=1}^{m} p_i e_i$$

where $p_i \in \mathbb{Q}[x]$ are polynomials for all $1 \le i \le m$. In other words, we can represent $1 = \pi(e_0)$ as a $\mathbb{Q}[x]$-linear combination of the $f_i = \pi(e_i)$. Moreover, we define

$$c := \mathrm{lcm}\,(\text{all denominators of all coefficients of all terms of all } p_i)\,.$$

Thus we get another syzygy $c\sigma$ which corresponds to the equation

$$c = c \cdot 1 = c \cdot \pi(e_0) = \sum_{i=1}^{m}(cp_i) \cdot \pi(e_i) = \sum_{i=1}^{m}(cp_i) \cdot f_i$$

where $cp_i \in \mathbb{Z}[x]$ for all $1 \le i \le m$. By construction it follows that $c \in I$. $\square$

Example 21. We give two examples, a small one we do by hand and a bigger one which gives a real benefit for the overall computational time:

---

[4]We choose $p > 3$ since it should at least not divide the lead coefficients of the input polynomials.

[5]A reduced standard basis over a field w.r.t. a global monomial order means that the lead coefficients of all elements in the basis are 1 and no lead term divides any term of any other polynomial in the basis.

Algorithm 4 RationalPreCheck (RPC)

Input: Ideal $I = \langle f_1, \dots, f_m \rangle \subset \mathcal{P}$, monomial order $<$
Output: Ideal J such that $J = I$
  1: $G \leftarrow$ standard basis for $\langle f_1, \dots, f_m \rangle$ in $\mathbb{Q}[x]$ w.r.t. $<$
  2: **if** $1 \in G$ **then**
  3:     $S \leftarrow \mathrm{syz}\left(\langle 1, f_1, \dots, f_m \rangle\right) \subset \sum_{i=0}^{m} \mathbb{Q}[x]e_i$ w.r.t. $<$
  4:     Search for $\sigma \in S$ with 0th component of the form ce_0, $c \in \mathbb{Q}$.
  5:     Find multiple $\lambda \in \mathbb{Z}$ such that $\lambda\sigma \in \sum_{i=0}^{m} \mathbb{Z}[x]e_i$
  6:     $J \leftarrow \langle \lambda c, f_1, \dots, f_m \rangle$
  7: **end if**
  8: **return** J
---

1. Let $I \subset \mathbb{Z}[x,y]$ be given by $I = \langle x+4, xy+9, x-y+8 \rangle$. We want to compute a strong standard basis for I w.r.t. the degree reverse-lexicographical order $<$. The standard basis for I over $\mathbb{Q}$ includes 1, so we have a constant in the standard basis for I over $\mathbb{Z}$. We compute $\mathrm{syz}\left(\langle 1, x+4, xy+9, x-y+8 \rangle\right) \subset \sum_{i=0}^{3} \mathbb{Q}[x,y]$ and get the following three syzygies:

$$
\begin{aligned}
\sigma_1 &= 7e_0 - (x+4)e_1 + e_2 + xe_3, \\
\sigma_2 &= (y-4)e_0 - e_1 + xe_3, \\
\sigma_3 &= (x+4)e_0 - e_1.
\end{aligned}
$$

   σ_1 is the relation from which we can extract the corresponding constant for I:

$$
\begin{aligned}
7 = 7\pi(e_0) &= (x+4)\pi(e_1) - \pi(e_2) - x\pi(e_3) \\
&= (x+4)(x+4) - (xy+9) - x(x-y+8) \\
&= x^2 + 8x + 16 - xy - 9 - x^2 + xy - 8x.
\end{aligned}
$$

   Thus, we add 7 to the initial generators of I and run Algorithm 3. We receive a strong standard basis $G = \{7, x+4, y-4\} \subset \mathbb{Z}[x,y]$ for I.

2. A bigger example is given as rationalPreCheckExample() in our publicly available benchmark library (Eder, C. (2018b)): The ideal I we are considering is generated by 70 polynomials in $\mathbb{Z}[x,y,z]$. We want to compute a strong standard basis for I w.r.t. the degree reverse-lexicographical order $<$. In Table 2 we give some characteristics of the computation of a standard basis for I on an Intel Core i7-5557U CPU with 16 GB RAM. Note that the computation of Algorithm 4 over $\mathbb{Q}$ takes < 0.01 seconds and needs < 0.3 MB of memory, so it is negligible compared to the computational cost over $\mathbb{Z}$. The same holds for the syzygy computation. The strong standard basis for I computed by our implementation consists of only 9 elements

$$
\begin{aligned}
G = \Big\{ &18, 6z-12, 2y-4, 2z^2+4z+8, . \\
&yz+z+3, 3x^2z-15x^2, x^2y+3x^2z+x2, \\
&x^3+10z, x^2z^2-4x^2z-11x^2 \Big\}.
\end{aligned}
$$

From Algorithm 4 we do not directly get the constant $18 \in G$, but we get a multiple of it: $6,133,248$. Adding this constant to the generators of I and applying Algorithm 3 represents the third column of Table 2, whereas a direct application of Algorithm 3 on I is given in column two.

Characteristics \ Algorithms	sBBA	RPC + sBBA
maximal degree	13	13
# zero reductions	1,130	795
# product / chain criteria	1,279 / 2,990	826 / 1,925
memory usage (in MB)	1.51	0.78

Table 2: Characteristics of standard basis computations of Example 21

Remark 22.
1. Clearly, one can generalize Algorithm 4: If we do not have $1 \in G$ over $\mathbb{Q}$ we might still get a short polynomial, even a monomial whose corresponding $\mathbb{Z}[x]$ representation can then be recovered from the corresponding syzygy module. Note that in this case the reconstruction is a bit harder and the precheck might take longer, since we have a more complex standard basis computation over $\mathbb{Q}$ that does not terminate early.
2. If one has a computer with at least two cores available the usage of parallel.lib resp. task.lib (Steenpaß, A. (2016a,b)) in Singular might be worthwhile: One could start the direct computation of sBBA over $\mathbb{Z}$ on one core plus RPC over $\mathbb{Q}$ on the other core. Using the waitfirst command one could always ensure that the fatest possible running time is achieved.

5. Normal Form Computations

Let us recall Definition 2 and give the two different types of a reduction step a name:

Definition 23. Let $f \in \mathcal{P}$ and let $G \subset \mathcal{P}$ be a finite subset.
1. If there exists $g \in G$ such that $\mathrm{lm}(g) \mid \mathrm{lm}(f)$ and $\mathrm{lc}(g) \mid \mathrm{lc}(f)$ then $f - \frac{\mathrm{lc}(f)}{\mathrm{lc}(g)} \frac{\mathrm{lm}(f)}{\mathrm{lm}(g)} g$ is a top-lt-reduction of f (w.r.t. g).
2. If there exists $g \in G$ such that $\mathrm{lm}(g) \mid \mathrm{lm}(f)$, $\mathrm{lc}(g) \prec \mathrm{lc}(f)$ and $\mathrm{lc}(g) \nmid \mathrm{lc}(f)$ then $f - a\frac{\mathrm{lm}(f)}{\mathrm{lm}(g)} g$ with $\mathrm{lc}(f) = a\,\mathrm{lc}(g) + b$ where $a, b \in \mathbb{Z}$, $b \neq 0$ amd $b \prec \mathrm{lc}(f)$ is a top-lc-reduction of f (w.r.t. g).

First we can note that it is enough to consider lt-reductions since lc-reductions are taken care of when generating new pairs:

Lemma 24. Algorithm 3 terminates and computes a correct strong standard basis for a given set of geneators and a given momonmial order if we change the corresponding normal form algorithms to consider only lt-reductions.

Proof. We need to show that lc-reductions are considered when adding new GCD-poly-nomials to the pair set P. Assume h is the outcome of an lt-reduction. If there exist possible lc-reductions for h w.r.t. G then there is a $g \in G$ such that $\mathrm{lm}(g) \mid \mathrm{lm}(h)$, $\mathrm{lc}(g) \nmid \mathrm{lc}(h)$ and $\mathrm{lc}(h) \nmid \mathrm{lc}(g)$. Thus $\mathrm{gpoly}(h, g)$ is generated:

$$\mathrm{gpoly}(h, g) = ah + btg$$

where $a, b \in \mathbb{Z}$ such that $\gcd(\mathrm{lc}(h), \mathrm{lc}(g)) = a\mathrm{lc}(f) + b\mathrm{lc}(g)$ and $t = \frac{\mathrm{lm}(h)}{\mathrm{lm}(g)}$. Now we distinguish two cases:

1. If $a = 1$ then $\mathrm{gpoly}(h, g) = h - btg$ which is exactly the corresponding lc-reduction of h w.r.t. g.
2. If $a \neq 1$ then the lc-reduction of h w.r.t. g, $h' = h - ctg$ for some $c \in \mathbb{Z}, t \in \mathcal{P}$, coresponds to the first step of the Euclidean algorithm calculating $\gcd(\mathrm{lc}(h), \mathrm{lc}(g))$. If there is no further reduction of h' then Algorithm 3 generates a corresponding GCD-polynomial between h' and g. It follows that

$$h - ctg + \mathrm{gpoly}(h', g) = \mathrm{gpoly}(h, g).$$

If h' is further reducible by some $g' \in G$ then we note the following: First of all $g' \in G \setminus \{g\}$ since $\mathrm{lc}(g) > \mathrm{lc}(h')$ thus there cannot exist a lt-reduction or lc-reduction of h' w.r.t. g. Now the reduction of h' by g' is given as

$$h' - c't'g' = h - ctg - c't'g'$$

for some $c' \in \mathbb{Z}$ and $t' \in \mathcal{P}$. Since $\mathrm{lm}(h) = \mathrm{lm}(tg) = \mathrm{lm}(t'g')$ we conclude that $ctg + c't'g'$ corresponds to a multiple of either $\mathrm{spoly}(g, g')$ or $\mathrm{gpoly}(g, g')$ depending on divisibility of $\mathrm{lc}(g)$ and $\mathrm{lc}(g')$. Nonetheless, once we have a standard representation for $\mathrm{spoly}(g, g')$ or $\mathrm{gpoly}(g, g')$ we can redefine h by $h - ctg - c't'g'$. Either the lead term or the lead coefficient of h increases in this process. Thus after finitely many steps this process terminates and we reach either case (1) or there is no further lc-reduction reduction possible.

$\square$

This was the usual way Singular implemented standard basis reduction over the integers. Clearly, this is due to historical reasons where the implementation over $\mathbb{Z}$ was only thought of as a slight generalization of the computation over fields.

Example 25. Recall Example 11 from Section 2: If we allow only lt-reductions when reducing $h = 4+x$ w.r.t. the strong standard basis $G = \left\{2 - x + y + x^2, x - 2y - x^2 - xy - x^3\right\}$ then we get the following reduction table with $T_h = G$ at the beginning.

If we compare it to Table 1 we see one more reduction step introduced by choosing $x - 2y - x^2 - xy - x^3$ as reducer in the second reduction step, since lc-reductions are not allowed.

Note that Lemma 24 shows that lc-reductions are, from the theoretical point of view, not needed. A lc-reduction corresponds to a first step in the Euclidean algorithm when calculating $\gcd(\mathrm{lc}(f), \mathrm{lc}(g))$ which will be done in the algorithm when considering $\mathrm{gpoly}(f, g)$. Still, it has a strong impact on the performance of the algorithm in practice: cutting the lead coefficient down as much as possible means that the element might

h	g	h added to T_h?
$4 + x$	$2 - x + y + x^2$	✓
$3x - 2y - 2x^2$	$x - 2y - x^2 - xy - x^3$	✓
$4y + x^2 + 3xy + 3x^3$	$4 + x$	-
$x^2 + 2xy + 3x^3$	$x - 2y - x^2 - xy - x^3$	✓
$4xy + 4x^3 + x^2y + x^4$	$4 + x$	-
$4x^3 + x^4$	$4 + x$	-
0	-	-

Table 3: Only lt-reductions used in Mora normal form

be used more often for reduction purposes. Moreover, generating new S-polynomials and GCD-polynomials with it leads to lower lead coefficients there. In some sense lc-reductions are the bridge between an lt-reduction of f by g and the $\mathrm{gpoly}\,(f, g)$. In one specific situation we can go directly from one to another, without the need of a bridge. Lemma 24 also gives a hint to this situation stated in the following during the reduction process over the integers. [6]

Lemma 26. Let $f \in \mathcal{P}$, $G \subset \mathcal{P}$ finite and $g \in G$ such that $\mathrm{lm}\,(g) = \mathrm{lm}\,(f)$. Then it holds that
$$\langle f, g \rangle = \langle \mathrm{spoly}\,(f, g), \mathrm{gpoly}\,(f, g) \rangle .$$

Proof. Let $u, v, d \in \mathbb{Z}$ such that
$$u \operatorname{lc}(f) + v \operatorname{lc}(g) = d = \gcd\,(\operatorname{lc}(f), \operatorname{lc}(g)) . \tag{1}$$

We can then write
$$
\begin{aligned}
\mathrm{gpoly}\,(f, g) &= & u & \quad f & + & \quad v & \quad g. \\
\mathrm{spoly}\,(f, g) &= & \tfrac{\operatorname{lc}(g)}{d} & \quad f & - & \quad \tfrac{\operatorname{lc}(f)}{d} & \quad g.
\end{aligned}
$$

In order to show the statement we have to prove that (f, g) and $(\mathrm{gpoly}\,(f, g), \mathrm{spoly}\,(f, g))$ generate the same $\mathbb{Z}$-lattice. So, in the above representation of $\mathrm{gpoly}\,(f, g)$ and $\mathrm{spoly}\,(f, g)$ in terms of f and g we have to show that the corresponding coefficient matrix is invertible, i.e. has determinant $\pm 1 \in \mathbb{Z}$: To see this we set
$$
M := \begin{pmatrix} u & v \\ \tfrac{\operatorname{lc}(g)}{d} & -\tfrac{\operatorname{lc}(f)}{d} \end{pmatrix}.
$$

---

[6]This idea is also implemented in the computer algebra system Macaulay2. We have discovered it independently and since we have not found any proof for the statement we give one here.

14

Finally, we compute

$$\det(M) = -u\,\frac{\mathrm{lc}\,(f)}{d} - v\,\frac{\mathrm{lc}\,(g)}{d} = -\frac{1}{d}\,(u\,\mathrm{lc}\,(f) + v\,\mathrm{lc}\,(g)) \stackrel{(1)}{=} -1.$$

$\square$

How to use Lemma 26 in sBBA? The idea is that whenever we reduce a new element f we check if there exists a reducer $g \in G$ with $\mathrm{lm}\,(f) = \mathrm{lm}\,(g)$, but $\mathrm{lc}\,(g) \nmid \mathrm{lc}\,(f)$. In this situation we do a 2-by-2 replacement:

1. Compute $\mathrm{gpoly}\,(f, g)$ and replace $g \in G$ by $\mathrm{gpoly}\,(f, g) \in G$. Clearly, already generated pairs with g as generator have to be adjusted respectively.
2. Compute $\mathrm{spoly}\,(f, g)$ and replace f by $\mathrm{spoly}\,(f, g)$. Note that we have not changed the degree of f, but probably only multiplied f with some coefficient. With the newly defined f we again enter the reduction process and see, if we can further reduce it.

This has two main advantages to the usual reduction process that would compute only an lc-reduction of f w.r.t. g:

1. We directly compute the GCD-polynomial of f and g whereas the before mentioned lc-reduction would represent only one step in the Euclidean algorithm for reaching $\mathrm{gpoly}\,(f, g)$. So we can directly replace g with $\mathrm{gpoly}\,(f, g)$ which leads to smaller coefficients and multiples during the pair generation. Furthermore, $\mathrm{gpoly}\,(f, g)$ reduces other elements at least in all situations g would reduce, but it can possibly fulfill more lt-reductions due to its smaller lead coefficient.
   Furthermore, since $\mathrm{lm}\,(\mathrm{gpoly}\,(f, g)) = \mathrm{lm}\,(g)$ we can replace all S-polynomials already generated with g, again giving smaller coefficients in upcoming reduction processes. Even more, using $\mathrm{gpoly}\,(f, g)$ we are possibly able to render more S-polynomials useless applying the chain criterion, due to the smaller lead coefficient.
2. For f the advantage is that we are not stuck with a lc-reduction only, but we can go on and perform the lt-reduction $\mathrm{spoly}\,(f, g)$ and thus directly lower the lead term without the need of adding f to the basis, which would blow up pair generation.

Remark 27.
1. Note that $\mathrm{lm}\,(f) = \mathrm{lm}\,(g)$ is an essential condition for the correctness of Lemma 26. If, for example, only $\mathrm{lm}\,(g) \mid \mathrm{lm}\,(f)$ holds such that $\lambda = \frac{\mathrm{lm}(f)}{\mathrm{lm}(g)} > 1 \in \mathcal{P}$, then we can only recover f and λg via $\mathrm{spoly}\,(f, g)$ and $\mathrm{gpoly}\,(f, g)$, but we are no longer able to recover g.
2. Overall applying lc-reductions has a huge effect on running time: In most examples we get a speedup factor of 3. If we even apply coefficient reductions to the tail terms of the newly added element to the basis, we get another factor of $3 - 5$.

When we enter the reduction process of an element f in sBBA we search for reducers in the following order:

1. Is there $g \in G$ for a lt-reduction of f? If so we can cut down the lead term of f without the need of multiplying f with any coefficient $\neq 1$.
2. Is there are $g \in G$ fulfilling Lemma 26? If so we can cut down the lead term of some coefficient multiple of f and we can further replace g by $\mathrm{gpoly}\,(f, g)$ leading to a better reducer.

15

3. Is there $g \in G$ for a lc-reduction of f? We cannot cut down the lead term of f, but at least we can reduce the lead coefficient before adding f to G and generating new S-polynomials and GCD-polynomials.

6. Computational results

In this section we present the new implementation for standard basis computation over Euclidean domains in Singular 4-1-2.[7] We compare it to the current implementations in the computer algebra systems Macaulay2 (version 1.12) and Magma (version 2.23). For the comparison we use benchmarks with different properties and behaviours. All examples are computed with respect to the degree reverse lexicographical order. All algorithms run single threaded, we use an Intel Core i7-6700 CPU with 3.4 GHz and 64GB RAM. The machine runs Arch Linux with unmodified 4.18.12 kernel. For the examples we refer to Eder, C. (2018b).

Examples	Singular (Thm. 17)	Singular (all pairs)	Macaulay2	Magma
Cyclic-6	0.330	0.320	4.708	2.799
Cyclic-7	18,731.820	5,636.210	out of memory	366.060
Katsura-7	2.200	2.250	204.928	251.630
Katsura-8	133.390	135.360	64,555.420	(> 24h)
Katsura-9	13,366.590	12,951.160	(> 24h)	(> 24h)
Eco-9	3.920	4.050	870.409	22.520
Eco-10	38.760	40.670	(> 24h)	289.540
F-633	0.140	0.120	14.982	12.880
F-744	118.610	117.890	(> 24h)	(> 24h)
Noon-7	34.930	32.700	(> 24h)	(> 24h)
Noon-8	3,1390.060	3,079.370	(> 24h)	(> 24h)
Reimer-5	3.620	3.590	out of memory	1,932.400
Reimer-6	1,216.960	1,232.530	out of memory	(> 24h)
Lichtblau	1.910	1.830	69.536	2,242.900
Bayes-148	9.970	9.900	117.635	46.240
Mayr-42	212.320	212.770	218.635	40.270
Yang-1	149.120	147.250	181.210	50.330
Jason-210	47.010	46.780	(> 24h)	(> 24h)

Table 4: Benchmark timings given in seconds ("(> 24h)" means that we have stopped the computation after at least 24 hours.)

We can see that Singular's new implementation is always faster than Macaulay2. Comparing it to Magma we see that Magma is way faster for Cyclic-7. We assume

---

[7]In the Singular sources since git commit 7d2091affbf4b4a1a382e5eb0a47f66c0f3c42f7a.

that this is due to Magma using Faugère's F4 algorithm (Faugère, J.-C. (1999)). Our implementation considers fewer S-polynomials and GCD-polynomials, but the reduction steps in higher degree are way slower than the linear algebra done in Magma. We believe that there are more classes of examples where the linear algebra attempt is more efficient than the polynomial arithmetic used in Singular, still, we need to investigate this problem in more detail. We can see that the F4 algorithm seems to be beneficial also for Mayr-42 and Yang-1. Nevertheless, for most of the other examples considered, Singular is by a factor faster than Magma.

In the above table we list timings for Macaulay2's Buchberger implementation using polynomial arithmetic (as for Singular). We also tested Macaulay2's F4 implementation, but in most of the above examples it was slower or just as fast as the polynomial arithmetic implementation. Due to its much higher memory usage, we could not finish most of the examples on the given machine. The only example where we have seen a better result was Mayr-42 where Macaulay2's F4 algorithm finished in 185 seconds, still using way more memory than Magma and nearly 10 times as much as Singular. In the given example Singular's new algorithm uses, aside from Cyclic-7, the least memory of all compared implementations.

For Singular we can see that generating all possible S-polynomials and GCD-polynomials or only the ones needed (recall Theorem 17) does not have a bigger influence on the computation for most of the examples. Still, for the bigger examples like Noon-8 or Katsura-9 we see that applying Theorem 17 leads to slightly slower computation. In Cyclic-7 we can even see a huge impact, the computation slows down by a factor of more than 3! It sems that having more pairs available at an earlier stage of the algorithm is advantageous compared to having fewer pairs overall. In most of the examples the algorithm taking care of all possible pairs does not even compute more reductions, the product and chain criterion removes those pairs which are really useless at some later stage. All in all it seems that considering all possible S-polynomials and GCD-polynomials leads to a more stable algorithm.

We found that the application of Lemma 26 becomes sometimes a bottleneck. For example, always exchanging f and g by $\mathbf{spoly}(f, g)$ and $\mathbf{gpoly}(f, g)$ has a huge drawback in the computation of a basis for Cyclic-7, leading to a slow down of a factor of more than 3. We found that overall it is a good heuristic to apply Lemma 26 at most 5 times per a single reduction process. The Singular timings in the above table correspond exactly to this implementation.

7. Conclusion

We have presented new ideas for computing standard bases over Euclidean domains. The implementation of the corresponding algorithms is available in Singular. We have seen that Singular is in general faster than Macaulay2 and Magma in various examples.

Our next steps include an implementation of the new ideas in the open source C library GB which implements Faugère's F4 algorithm (Eder, C. (2018a)). Doing so we hope to benefit from the new ideas and the fast linear algebra. Still, not all ideas presented here are trivial to move to an F4-style algorithm.

Moreover, we still see a lot of zero reductions in higher degree which slow down the computation. In order to tackle this problem, we work on a more general chain criterion

trying to exploit more of the structure of the input system. Even more, a further attempt on signature-based computation over Euclidean rings, see (Eder, C. et al. (2017)), should be possible.

Finding better heuristics for the application of Lemma 26 depending on the structure of the input systems is also an interesting topic to study further. If applied in a "good" way it can have a strong impact on the overall computation.

Another topic we are working on is to improve the implementation in Singular for Euclidean domains with zero divisors. There, special care needs to be taken of the annihilator polynomials.

References

Arnold, E. A., April 2003. Modular algorithms for computing Gröbner bases. Journal of Symbolic Computation 35, 403–419.
URL http://portal.acm.org/citation.cfm?id=937627.937629

Becker, T., Weispfenning, V., Kredel, H., 1993. Gröbner Bases. Graduate Texts in Mathematics, Springer Verlag.

Buchberger, B., 1965. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. Ph.D. thesis, University of Innsbruck.

Buchberger, B., 1979. A Criterion for Detecting Unnecessary Reductions in the Construction of Gröbner Bases. In: EUROSAM '79, An International Symposium on Symbolic and Algebraic Manipulation. Vol. 72 of Lecture Notes in Computer Science. Springer, pp. 3–21.

Buchberger, B., 1985. Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory, 184–232.

Buchberger, B., 2006. An Algorithm for Finding the Basis Elements of the Residue Class Ring of Zero Dimensional Polynomial Ideal (English translation of Bruno Buchberger's PhD thesis. Journal of Symbolic Computation 41 (3-4), 475 – 511.

Ceria, M., Mora, T., Roggero, M., 2019. A general framework for noetherian well ordered polynomial reductions. Journal of Symbolic Computation 95, 100 – 133.
URL http://www.sciencedirect.com/science/article/pii/S0747717119300124

Eder, C., 2018a. gb. https://github.com/ederc/gb.

Eder, C., 2018b. singular-benchmarks. https://github.com/ederc/singular-benchmarks.

Eder, C., Pfister, G., Popescu, A., 2017. On Signature-based Gröbner Bases over Euclidean Rings. In: ISSAC 2017: Proceedings of the 2017 International Symposium on Symbolic and Algebraic Computation. pp. 141–148.

Faugère, J.-C., June 1999. A new efficient algorithm for computing Gröbner bases (F4). Journal of Pure and Applied Algebra 139 (1–3), 61–88, http://www-salsa.lip6.fr/~jcf/Papers/F99a.pdf.

Gebauer, R., Möller, H. M., October/December 1988. On an installation of Buchberger's algorithm. Journal of Symbolic Computation 6 (2-3), 275–286.

Grauert, H., 1972. Über die Deformation isolierter Singularitäten analytischer Mengen. Inventiones Mathematicae 15 (3), 171–198.

Greuel, G.-M., Pfister, G., 2007. A Singular Introduction to Commutative Algebra, 2nd Edition. Springer Verlag.

Hironaka, H., 1964a. Resolution of Singularities of an Algebraic Variety over a Field of Characteristic Zero: I. Annals of Mathematics 79 (1), 109–203.

Hironaka, H., 1964b. Resolution of Singularities of an Algebraic Variety over a Field of Characteristic Zero: II. Annals of Mathematics 79 (2), 205–326.

Kandri-Rody, A., Kapur, D., 1984. Algorithms for computing gröbner bases of polynomial ideals over various euclidean rings. In: Fitch, J. (Ed.), EUROSAM 84. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 195–206.

Kandri-Rody, A., Kapur, D., 1988. Computing a Gröbner basis of a polynomial ideal over a Euclidean domain. Journal of Symbolic Computation 6 (1), 37 – 57.
URL http://www.sciencedirect.com/science/article/pii/S0747717188800208

Lichtblau, D., 2012. Effective computation of strong Gröbner bases over Euclidean domains. Illinois J. Math. 56 (1), 177–194.
URL http://projecteuclid.org/euclid.ijm/1380287466

Mora, T., 2005. Solving Polynomial Equation Systems II: Macaulay's Paradigm and Gröbner Technology. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

Mora, T., 2016. Solving Polynomial Equation Systems IV. Vol. 4 of Encyclopedia of Mathematics and its Applications. Cambridge University Press.

Mora, T., 1982. An Algorithm to Compute the Equations of Tangent Cones. EUROCAM 82, Lecture Notes in Comp. Sci.

Pan, L., 1989. On the d-bases of polynomial ideals over principal ideal domains. Journal of Symbolic Computation 7 (1), 55 – 69.
URL http://www.sciencedirect.com/science/article/pii/S0747717189800069

Steenpaß, A., 2016a. parallel.lib. A Singular library for parallel processes. http://www.singular.uni-kl.de.

Steenpaß, A., 2016b. tasks.lib. A Singular library for task handling. http://www.singular.uni-kl.de.

Wienand, O., 2011. Algorithms for symbolic computation and their applications - standard bases over rings and rank tests in statistics. Ph.D. thesis.