Vom Fachbereich Mathematik der Technischen Universität Kaiserslautern zur Verleihung des akademischen Grades Doktor der Naturwissenschaften (Doctor rerum naturalium, Dr. rer. nat.) genehmigte Dissertation

Signature–based algorithms to compute standard bases

Christian Eder

1. Gutachter: Prof. Dr. Gerhard Pfister
2. Gutachter: Prof. Dr. Vladimir P. Gerdt
Vollzug der Promotion: 13. April 2012

D 386

Preface

Standard bases

The idea of standard bases has its origin in [90] by Gordan. Afterwards, Macaulay ([121]) and Gröbner ([98]) used monomial orders to study Hilbert functions of graded ideals. Moreover, they found $\mathcal{K}$–bases of zero–dimensional quotiend rings with this approach.

In 1965 Bruno Buchberger introduced the notion of a Gröbner basis in his PhD thesis ([34]). The terminology acknowledges the influence of Buchberger's advisor on his work, Wolfgang Gröbner. The term standard basis denotes a more general approach of Gröbner bases, which can be used not only over ground fields, but also over ground rings. Independently, Buchberger, Grauert, and Hironaka introduced the notion of a standard basis ([34, 95, 101, 102]).

There exist very few concepts in the field of computational algebraic geometry and commutative algebra with such an impact on the development of new concepts as standard bases. Standard bases have various applications, for example, solving systems of polynomial equations, elimination, ideal membership, and ideal intersection problems. They are one of the most important tools in computer algebra, computational algebraic geometry, and computational commutative algebra.

Their computations can be understood as multivariate, non–linear generalizations of the Euclidean Algorithm for computing the univariate greatest common divisors, of the Gaussian Elimination of systems of linear equations, and of integer programming problems.

Whereas the general idea of how to compute a standard basis, based on Buchberger's Criterion, is quite straightforward and a practical implementation can be easily done, such an ad–hoc algorithm is not efficient at all due to several problems concerning mostly useless computations and overhead inrtroduced by them. Over the last, nearly 50 years lots of improvements in terms of the computation of standard bases have been made. Not only criteria to detect useless data during the algorithm's working have been found, but also selection strategies for better reducers, modular methods to keep coefficients small even when computing in polynomials rings over the rationals, and quite a lot of other machinery have been developed. Due to these ideas, more efficient algorithms are possible, which can compute way harder examples previously intractable. Solving these, again, leads to new insight in other fields of algebraic geometry and commutative algebra, giving impulses for new approaches there.

Signature–based approach

A special kind of those recent, algorithmic improvements in the area of standard bases are the ones based on signatures. Those algorithms are in the focus of this thesis. In 2002, Faugère published a new standard basis algorithm, called F5. This algorithm uses a completely new concept to detect useless data during the computations, called signatures. Also it lays some restrictions on the input data, it is known as one of the most efficient standard basis algorithms nowadays due to its powerful criteria. The algorithm is known to compute very few zero reductions. In the rather common situation of the underlying polynomial system defining a complete intersection, it even does not reduce any polynomial to zero. In general, reduction to zero is the primary bottleneck in the computation of a standard basis, since it does not deliver any new information for the algorithm, but takes lots of polynomial operations. It is thus no surprise that F5 has succeeded at computing many standard bases that were previously intractable.

An open question surrounding the F5 algorithm regards correctness and termination. Due to the aggressive criteria used in F5, a proof for its correct computation cannot be given as easily as for the classical algorithms using other criteria. Moreover, in a traditional algorithm computing standard bases, the proof of termination follows from the algorithm's ability to exploit the Noetherian property of monomial ideals: each polynomial added to the basis G expands the ideal generated by the leading terms of G. This is not true with

F5; the same criteria that detect reductions to zero also lead the algorithm to add to G polynomials which do not expand the ideal of leading terms.

Lots of variants of F5 have been developed over the last couple of years, and a more general view on signature–based algorithms are possible right now. Understanding those algorithms lead to new insights in recent optimizations. The signature–based world is a field of active research right now, with lots of new and promising results to come up in the near future, hopefully leading to even better algorithms.

Results of this thesis

This thesis is devoted to efficient computations of standard bases with an emphasis on signature–based attempts. The structure of the presentation is the following:

(1) In Chapter 1 we give a short introduction to commutative algebra with a view to computational aspects. In Section 1.7 we start the particular discussion about standard bases. At the end of Chapter 1 we have covered the basic ideas behind the computational aspects and have already seen the most obvious problems with the standard approach of the Buchberger Algorithm, Std.

(2) Based on this first approach we give an in–depth overview of various optimizations of the Buchberger Algorithm. These cover not only Buchberger's criteria, but also more sophisticated improvements like modular computations, usage of the Hilbert polynomial, Gröbner walks, or involutive methods. Thus Chapter 2 should be understood as a smorgasbord of ideas that where presented over the last 40 years to receive more efficient algorithms computing standard bases.

(3) Also attempts like using staggered linear bases or syzygies to improve standard basis computations fit to the content of Chapter 2, we present them in more detail in Chapter 3. The reason we handle these different is based on the fact that those ideas represent the origin of signature–based standard basis algorithms this thesis is dedicated to.

(4) Beginning in Chapter 4 we give a complete, generalized introduction to signature–based standard basis algorithms. This is done in such a general setting for the first time. Besides introducing the basic notions and ideas, we present a rather generic implementation of an standard basis algorithm using signatures called SigStd. As this algorithm does not contain any criteria check or other optimization it can be understood as a mirroring of the first classical standard basis algorithm Std presented in Section 1.8. It turns out that all optimized signature–based algorithms presented in the rest of this thesis can be derived from this implementation by adding respective criteria checks and reduction processes.

(5) Whereas Chapter 5 discusses rather easy variants of SigStd, Chapter 6 is devoted to Faugère's F5 Algorithm. Although Faugère's algorithm is the origin of all other signature–based ones, the order of representation makes sense.

The algorithms presented in Chapter 5 can be understood in two ways:

a) On the one hand, they are optimizations of SigStd and their different approaches are presented in detail. Moreover, we give a vast comparison of their performance devoting a whole section on experimental results.

b) On the other hand, all these algorithms are nothing else but, rather good, simplifications of F5. From this point of view F5 can be interpreted as the most aggressive one of all signature–based algorithms.

This in mind it is quite good to postpone the introduction to F5 and start with the more comprehensible discussion of the other variants. Proving correctness and termination of the improved algorithms presented in Chapter 5 can easily be done deriving the ideas of the corresponding proofs given for SigStd.

(6) On the contrary, the corresponding proofs for F5 turn out to be a lot more tricky. In Section 6.1 we give the first complete proof of F5's correctness. Even more, we give not only optimizations of F5, but we also tackle the, still unsolved, problem of showing F5's termination. So besides improving the algorithm, a rather complex theoretical background is part of Chapter 6.

(7) Chapter 7 finishes this thesis, presenting various different topics, all of them focussing on the generalization and optimization of signature–based standard basis algorithms. Partiallyi, complete new proofs are given never published before, sometimes approaches of active research are discussed and give the reader a deeper insight in the topic. Most of these ideas need implementations and more theoretical results, but they represent the worthwile field of signature–based algorithms, which are promising to host lots of more improvements for standard basis computations in the near future.

This thesis contains material from the author's (partly published) articles [55–59]. In particular, the thesis contains joint work with Justin Gash and John Perry. Most of the results of these articles are generalized in this thesis, the respective publication is listed at the beginning of the corresponding chapter respectively section it gives contributions to.

Financial support

Financial support was provided by the Forschungszentrum Oberwolfach via a graduate fellowship.

Acknowledgements

Danke.

Contents

Preface i

1 An introduction to standard bases 1

1.1 Rings, ideals, and modules . . . 2
1.2 Polynomial rings . . . 10
1.3 Monomial orders on polynomial rings . . . 13
1.4 Monomial orders on free $\mathcal{P}$–modules . . . 18
1.5 Gradings . . . 22
1.6 Hilbert–Poincaré series and dimensions . . . 24
1.7 Normal forms and standard bases . . . 26
1.8 The basic standard basis algorithm . . . 35
1.9 On the complexity of standard basis computations . . . 41

2 Ways to improve standard basis computations 43

2.1 The problem of zero reductions . . . 45
2.2 Selection strategies for critical pairs . . . 46

2.3 Buchberger's criteria 49
2.4 The Gebauer–Möller implementation 52
2.5 Normal form computations and their relation to Gaussian elimination . . 55
2.6 Picking a good reducer 63
2.7 Using the Hilbert–Poincaré series 71
2.8 Going the indirect way 77
2.9 Modular standard basis computations 88
2.10 Involutive bases 98
2.11 Concluding remarks 100

3 Syzygy modules and standard bases 103
3.1 Staggered linear bases 104
3.2 Syzygies and free resolutions 111
3.3 Computing standard bases using syzygies 118

4 An introduction to signature–based standard basis algorithms 125
4.1 Basic ideas behind signatures and labeled polynomials 126
4.2 A generic signature–based standard basis algorithm 135
4.3 Some remarks on sig–safeness 143

5 Signature–based criteria to detect useless critical pairs 149
5.1 Generic criteria based on signatures 150
5.2 Reducing computational overhead in SigStd 157
5.3 An explicit choice in (RW) 162
5.4 A variant of AP using sparser polynomials 166
5.5 G2V – Complete reduction, weakened (RW) 168
5.6 Experimental results 173

6 Faugère's F5 Algorithm 183
6.1 Faugère's initial presentation of F5 184
6.2 F5C – F5 using reduced bases 203
6.3 Classifying F5 in the signature–based world 211
6.4 Experimental results 216
6.5 Termination–ensured variants of F5 217

7 Generalizing signature–based algorithms 239
7.1 Signature–based algorithms and inhomogeneous input 240
7.2 Computing the ideal quotient 247
7.3 Generalizing signatures 252
7.4 Non–incremental signature–based standard basis algorithms 254
7.5 Parallelization of signature–based algorithms 258
7.6 Computing syzygies with generalized signature–based algorithms 260

A Examples 267
Index 320

List of Figures

2.8.1 An example of fans, cones and faces 80
2.8.2 Crossing the border of two Gröbner cones 84
2.8.3 The classification of $\mathcal{K}[x, y]$ by N, E and S 86
2.9.1 Parallelized modStd 93

5.6.1 Coloration of results for variants of SigStd 174

6.1.1 Illustration of the Rewritten Criterion 192
6.4.1 Coloration of the results for different variants of F5 216
6.5.1 Coloration of the results for termination variants of F5 234

List of Tables

5.1 Time needed to compute a standard basis, given in seconds. 177
5.2 Number of zero reductions computed by the algorithms. 178
5.3 Memory used to compute a standard basis, given in Megabyte. 179
5.4 Number of all reduction steps during the computations. 180
5.5 Number of critical pairs not detected by the respective criteria used. 181
5.6 Size of the resulting standard basis. 182

6.1 Time needed to compute a standard basis, given in seconds. 218
6.2 Number of zero reductions computed by the algorithms. 218
6.3 Memory used to compute a standard basis, given in Megabyte. 219
6.4 Number of critical pairs not detected by the respective criteria used. 219
6.5 Number of all reduction steps during the computations. 220
6.6 Size of the resulting standard basis. 220
6.7 Timings (in seconds) & degrees of F5, F5B, and F5B+ 238

7.1 Computation for inhomogeneous input using F5E 246

7.2 Computation for inhomogeneous input using AP 246

List of Algorithms

1 Normal form w.r.t. G for a global order $<$ (GNF) 31
2 Reduced normal form w.r.t. G for a global order $<$ ($\mathrm{GNF}_{\mathrm{red}}$) 31
3 Normal form w.r.t. G for a non–global order $<$ (LNF) 33
4 Standard basis computation w.r.t. $<$ (Std) 38

5 Standard basis algorithm including selection strategy (Std) 46
6 Improved standard basis computation w.r.t. $<$ (GM) 54
7 Updating the set of critical pairs (Update) 55
8 Faugère's F4 Algorithm (F4) 57
9 Symbolic preprocessing of possible reducers (SymPre) 59
10 Reduction process in F4 (F4Reduction) 60
11 Improved F4 Algorithm (F4) 65
12 Improved Symbolic preprocessing of possible reducers (SymPre) 66
13 Reduction process in the improved F4 (F4Reduction) 67
14 Simplifying the reduction process in F4 (Simplify) 68

15 SlimGB Algorithm computing a standard basis w.r.t. < (SlimGB) 69
16 Normal form w.r.t. G of SlimGB (SlimNF) 70
17 Replacement check for SlimGB (Replace?) 70
18 Hilbert–driven variant of GM w.r.t. a global order < (HGM) 75
19 Dynamic variant of GM w.r.t. a global order < (DGM) 78
20 Gröbner walk to compute a reduced Gröbner basis (GBWalk) 83
21 Gröbner basis conversion algorithm (FGLM) 87
22 Modular standard basis computation (modStd) 92
23 Gröbner trace reconstruction algorithm (GBTrace) 95
24 Modular Gröbner trace algorithm (TraceModStd) 97

25 Initial staggered linear basis algorithm (StagGB1) 108
26 Revised staggered linear basis algorithm (StagGB2) 110
27 Normal form computation for staggered linear bases (StagNF) 111
28 Standard basis algorithm for the first module of syzygies (Syz1) 114
29 Standard basis algorithm for the first module of syzygies (Syz2) 116
30 Standard basis algorithm using syzygies to improve computations (SyzStd) 118
31 Normal form w.r.t. G of SyzStd (SyzNF) 119

32 Generic signature–based standard basis computation w.r.t. < (SigStd) . . 136
33 Incremental signature–based standard basis computation w.r.t. < (IncSig) 138
34 Semi–complete sig–safe reduction algorithm (SigRed) 139
35 Slim semi–complete sig–safe reduction algorithm (SigRed) 145

36 IncSig including implementations of (NM) and (RW) (IncSigCrit) . . . 154
37 Generic implementation of (NM) (NonMin?) 155
38 Generic implementation of (RW) (Rewrite?) 156
39 SigStd with reduced standard bases (SigStdRed) 159
40 AP's implementation of (NM) (NonMinAP?) 164
41 AP's implementation of (RW) (RewriteAP?) 165
42 MM's implementation of (RW) (RewriteMM?) 167
43 G2V's sig–safe reduction algorithm (SigRedG2V) 169
44 G2V's implementation of (RW) (RewriteG2V?) 171
45 G2V's implementation of IncSigCrit (IncSigG2V) 172

46 The F5 Algorithm(F5) 185
47 F5's implementation of (NM) (NonMinF5?) 187
48 F5's F5 Criterion adding algorithm (addF5Crit) 187
49 F5's semi–complete sig–safe reduction algorithm (SigRedF5) 189
50 Incremental F5 step (IncF5) 191
51 F5's rule adding algorithm (addRule) 193
52 F5's implementation of the Rewritten Criterion (RewriteF5?) 193
53 F5C's interreduction process (ReduceF5) 206
54 The F5 Algorithm using reduced standard bases(F5C) 207
55 Incremental F5C step (IncF5C) 210

56 Incremental F5E step (IncF5E) 215
57 Termination ensured incremental F5 step (IncF5+) 230
58 F5's semi–complete sig–safe reduction algorithm (SigRedF5+) 231
59 Termination ensured incremental F5 step (IncF5B+) 235

60 SigStd including ideal quotients(SigStdQ) 250
61 IncSigCrit with curr–index labeled polynomials (IncSigQ) 251
62 The F5Syz Algorithm(F5Syz) 262
63 Incremental F5 step computing syzygies(IncF5Syz) 265
64 F5Syz's semi–complete sig–safe reduction algorithm (SigRedF5Syz) 266

1 An introduction to standard bases

This chapter introduces the basic notions of rings, ideals, and modules. After stating and explaining essential properties and connections of these mathematical structures, we focus our introduction on polynomial data. Section 1.2 gives a review of notations and basics of polynomials.

After presenting the fundamental structures our area of research is based on in Sections 1.1 – 1.6, we introduce the objects of main interest, namely standard bases. The notion of standard bases as well as basic algorithms for their computations are given in Section 1.7. We finish this chapter with a short discussion on the complexity of standard basis computations motivating the wish to improve computations, which is content of Chapter 2.

For a more detailed introduction on commutative algebra we refer the reader to [12,60]. Good books covering those topics with a more detailed introduction to standard bases and a stronger emphasis on computational aspects are, for example, [18, 50, 97, 111, 112].

Most of the proofs given in this chapter are either easy or can be found in any introductory book about commutative resp. computational algebra (for example in the ones mentioned above). Thus, we skip most of the proofs, give some references where needed and state them if they are short, beautiful, and give some deeper insight on the topics covered. None of the presented statements are genuine nor their proofs. We focus ourselves on the topic of standard basis computations, thus some introductory aspects of commutative resp. computer algebra are only shortly covered, whereas others are explained in detail.

Readers familiar with these topics may want to skim this chapter for notation and terminology.

1.1 Rings, ideals, and modules

Let us start with the basic structures building the ground for concepts in computer algebra.

Definition 1.1.1. A *ring* is a set R together with two binary operators

$$\begin{aligned} + : R \times R &\longrightarrow R, (a,b) \longmapsto a+b \\ \cdot : R \times R &\longrightarrow R, (a,b) \longmapsto a \cdot b =: ab \end{aligned}$$

referred to as addition and multiplication, such that

(1) $(R,+)$ is an abelian group with neutral element $0 \in R$. The inverse w.r.t. $+$ of $a \in R$ is denoted by $-a$.

(2) Multiplication is associative, i.e. $(ab)c = a(bc) = abc$ for all $a,b,c \in R$.

(3) The distributive laws $a(b+c) = ab+ac$ and $(a+b)c = ac+bc$ hold for all $a,b,c \in R$.

If, in addition, multiplication is commutative, i.e. $ab = ba$ for all $a,b \in R$, then $(R,+,\cdot)$ is called a *commutative ring*. $(R,+,\cdot)$ is called a *ring with (identity)* 1_R if $1_R \in R$, $1_R \neq 0$, and $1_R a = a$ for all $a \in R$.
A subset S of a ring R is called a *subring of R* if it is closed under the ring operations induced from R (restricted to S).

Convention. In this thesis *ring* always means *commutative ring with identity*. Omitting the notation of the two binary operators, we denote a ring just by R. Moreover, when the ring corresponding to the identity is clear, we omit the subscript and denote it by 1.

Example 1.1.2.

(1) The integers $\mathbb{Z}$, the rationals $\mathbb{Q}$, the reals $\mathbb{R}$, and the complex numbers $\mathbb{C}$ are rings with their natural addition and multiplication. field as $\mathbb{Z}^* \neq \mathbb{Z}\backslash\{0\}$.

(2) Any field is an integral domain, e.g. the rationals, the reals, and the complex numbers.

A well–known algebraic structure is the *vector space* consisting of a field and a group, connected together by an operation of the field on the group. This structure is generalized in the following way:

Definition 1.1.3.

(1) Let R be a ring. An R–*module* M is a set M together with two binary operators $+ : M \times M \to M$ and $\cdot : R \times M \to M$ (scalar multiplication) such that the following hold for all $a, b \in R$, $m, m' \in M$:

a) $(M, +)$ is an abelian group,

b) $a \cdot (b \cdot m) = (ab) \cdot m$,

c) $(a + b) \cdot m = a \cdot m + b \cdot m$,

d) $a \cdot (m + m') = a \cdot m + a \cdot m'$,

e) $1_R \cdot m = m$.

An abelian subgroup $N \subseteq M$ is called an R–*submodule* if $R \cdot N \subseteq N$. In particular, an R–submodule I of the R–module R is called an *ideal* of R.

(2) Let $(S, +)$ be a monoid. An S–*monomodule (or S–monoid module)* N is a set N together with two binary operators $+ : N \times N \to N$ and $\cdot : S \times N \to N$ such that the following hold for all $n \in N$, $s, s' \in S$:

a) $(s + s') \cdot n = s \cdot (s' \cdot n)$,

b) $1_S \cdot n = n$.

Remark 1.1.4. Note that the concept of a monomodule is used in Section 4.2 to prove termination of a generic signature–based standard basis algorithm. We do not use this definition anywhere else, besides characterizing the property of being Noetherian (see Lemma 1.1.15), in this thesis. The reader unfamiliar with this notion should not bother with it too much.

Example 1.1.5.

(1) Every abelian group is a $\mathbb{Z}$–module: As $1_{\mathbb{Z}} \cdot m = m$ the scalar multiplication is defined by $r \cdot m := \underbrace{(1 + \cdots + 1)}_{r \text{ times}} \cdot m = \underbrace{m + \cdots + m}_{r \text{ times}}$.

(2) If R is a ring, then R itself is an R–module with the ring operations. Also $\{0\}$ is an R–module resp. ideal of R.

(3) If $R = \mathcal{K}$ is a field, then R–modules are $\mathcal{K}$–vector spaces. Moreover, $\{0\}$ and $\mathcal{K}$ are the only ideals in $\mathcal{K}$.

(4) $2\mathbb{Z}$ is the ideal of all even numbers in $\mathbb{Z}$. On the other hand, $2\mathbb{Z}+1$ of all odd numbers is not an ideal, for example, $2 \in \mathbb{Z}, 3 \in 2\mathbb{Z} + 1$, but $2 \cdot 3 = 6 \notin 2\mathbb{Z} + 1$.

(5) $\{f \in \mathcal{C}(\mathbb{R}) \mid f(1) = 0\}$ is an ideal in $\mathcal{C}(\mathbb{R})$.

Definition 1.1.6. Let M be an R–module, $N = \{n_1, \dots, n_s\}$ be a non–empty subset of M, and let I be an ideal in R.

(1) The set of all R–linear combinations of elements of N is a module over R. It is denoted $\langle N \rangle := \langle n_1, \dots, n_s \rangle$, the module *generated by* N. By convention, the module generated by the empty set is $0 := \langle \emptyset \rangle$.

(2) If $M = \langle N \rangle$, then N is called a *system of generators of* M.

(3) If $\#(N) < \infty$ we call M *finitely generated*.

(4) If $\#(N) = 1$ we call M *cyclic*. In the special case of M being an ideal, we speak of a *principal ideal*. If every ideal in R is principal, then R is called a *principal ideal ring*. If R is furthermore an integral domain than R is called a *principal ideal domain*.

(5) N is called an R*–basis of* M if each $m \in M$ has a unique representation $m = \sum_{i=1}^{s} r_i n_i$. If M has an R–basis, then M is called a *free* R*–module*.

(6) If M is a finitely generated free R–module with R–basis N, then s is called the *rank of* M, $\operatorname{rank}(M) = s$.

(7) For $t \in \mathbb{N} \backslash \{0\}$ we set $R^t := \{(r_1, \dots, r_t) \mid r_1, \dots, r_t \in R\}$. Then R^t is the free R–module of all s–tuples w.r.t. component–wise addition and scalar multiplication. Moreover, let $e_i = (0, \dots, 0, 1, 0, \dots, 0) \in R^t$ with the ith entry being 1 for all $i \in \{1, \dots, t\}$, then the set $\{e_1, \dots, e_t\}$ is an R–basis of R^t, the so–called *canonical basis*.

(8) For any ideal $I \subset R$ we denote the radical of I by $\sqrt{I} := \{a \in R \mid \exists d \in \mathbb{N} \text{ such that } a^d \in I\}$.

(9) I is called a *prime ideal* if for all $a, b \in I$ such that $ab \in I \Rightarrow (a \in I \text{ or } b \in I)$.

(10) I is called a *primary ideal* if for all $a, b \in I$ such that $ab \in I$ and $a \notin I$ it holds that $b \in \sqrt{I}$.

(11) $I \subsetneq R$ is called a *maximal ideal* if there exists no $J \subsetneq R$ such that $I \subsetneq J \subsetneq R$.

(12) Let $I, J \subset R$ be two ideals. Then we denote the *ideal quotient of* I *by* J[1] by

$$I : J := \{r \in R \mid rJ \subset I\}.$$

Moreover, the *saturation of* I *by* J is given by

$$I : J^\infty := \{r \in R \mid \exists n \in \mathbb{N} \text{ such that } rJ^n \subset I\}.$$

Example 1.1.7.

(1) The ideal $2\mathbb{Z} \subset \mathbb{Z}$ is a free $\mathbb{Z}$–module with rank 1.

(2) The ring R is a free R–module of rank 1 with basis 1.

[1] Sometimes this is also denoted *colon module* resp. *colon ideal*.

(3) The ideal $6\mathbb{Z}$ is not a prime ideal in $\mathbb{Z}$ as $2 \cdot 3 \in 6\mathbb{Z}$ whereas $2, 3 \notin 6\mathbb{Z}$.

(4) $\mathbb{Z}$ is a principal ideal domain.

In this thesis we are mainly interested in modules and their computations. For this we have to consider some more basic properties and structures related to rings and ideals.

Lemma 1.1.8. *Let M_1 and M_2 be modules over the ring R. Then the following hold:*

(1) $M_1 \cap M_2$ is a module over R.

(2) We define the sum of M_1 and M_2 by

$$M_1 + M_2 := \{m_1 + m_2 \mid m_1 \in M_1, m_2 \in M_2\}.$$

$M_1 + M_2$ is a module over R, $M_k \subset M_1 + M_2$ for $k = 1, 2$.

We skip the straightforward proof and continue with an important definition we use several times when improving standard basis algorithms and try to ensure their termination.

Definition 1.1.9. Let M be an R–module, $N \subset M$ a submodule, $m \in M$. We define the *residue class of m modulo N* by

$$m + N := \{m + n \mid n \in N\}.$$

The following property for residue classes modulo N is straightforward.

Lemma 1.1.10. *Let M be an R–module, $N \subset M$ a submodule. For all $a, b \in M$ it holds that*

$$a + N = b + N \Longleftrightarrow a - b \in N.$$

Definition 1.1.11. Let M be an R–module, $N \subset M$ a submodule. The set $M/N := \{m + N \mid m \in M\}$ forms an R–module, the *quotient module*, together with the two binary opterators

$$(m + N) + (m' + N) = (m + m') + N$$
$$a \cdot (m + N) = (am) + N$$

for all $a \in R$, $m, m' \in M$.

In the special situation where we consider R as an R–module, $I \subset R$ an ideal, the set R/I forms a ring, the *quotient ring*.
There is a close connection between properties of the ideal I and those of the quotient ring R/I:

Proposition 1.1.12. *Let I be an ideal in the ring R.*

(1) I is prime $\Leftrightarrow R/I$ is an integral domain.

(2) I is maximal $\Leftrightarrow R/I$ is a field.

(3) *Every maximal ideal is prime.*

Proof.

(1) Let $a, b \in R$. $ab \in I \Leftrightarrow (ab) + I = (a + I) \cdot (b + I) = 0 \in R/I$. This proves (1).

(2) I and R are the only ideals containing $I \Leftrightarrow R/I$ has only the ideals 0 and R/I. This implies (2).

(3) A field is an integral domain, thus (3) follows from (1) and (2).

□

Example 1.1.13.

(1) For any prime number p, $\mathbb{Z}/p\mathbb{Z}$ is a ring with the usual additon and multiplication. Moreover, $p\mathbb{Z}$ is a maximal ideal in $\mathbb{Z}$, thus $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$ is a field. This also implies that $\left(\mathbb{Z}/p\mathbb{Z}\right)^* = \left(\mathbb{Z}/p\mathbb{Z}\right)\backslash\{0\}$.

(2) We have already seen in Example 1.1.7 that $6\mathbb{Z}$ is not a prime ideal. Thus $\mathbb{Z}/6\mathbb{Z}$ is not an integral domain:

$$(2 + 6\mathbb{Z}) \cdot (3 + 6\mathbb{Z}) = 6 + 6\mathbb{Z} = 0 + 6\mathbb{Z}.$$

Definition 1.1.14. A (mono–)module M is called *Noetherian* if every submodule $N \subset M$ is finitely generated. In particular, a ring R is called *Noetherian* if every ideal I in R is finitely generated.

Clearly, any field $\mathcal{K}$ is Noetherian as the only ideals in $\mathcal{K}$ are $\langle 0 \rangle$ and $\langle 1 \rangle$.

The next lemma is very useful when it comes to computations. It is used to ensure termination of most of the algorithms presented in this thesis.

Lemma 1.1.15.

(1) *Submodules and quotient modules of Noetherian modules are Noetherian.*

(2) *Let M be an R–module, $N \subset M$ a submodule. Then the following are equivalent:*
   a) *M is Noetherian.*
   b) *N and M/N are Noetherian.*

(3) *Let M be an R–module. Then the following are equivalent:*
   a) *M is Noetherian*
   b) *For every ascending chain of submodules of M*

$$M_1 \subset M_2 \subset \cdots \subset M_k \subset \cdots$$

   there exists $k \in \mathbb{N}$ such that $M_l = M_k$ for all $l \geq k$.
   c) *Every non–empty set of submodules of M has a maximal element with respect to inclusion.*

(4) *Let $(S, +)$ be a monoid, and let M be an S–monomodule. Then the following are equivalent:*

   a) *M is Noetherian*

   b) *For every ascending chain of submodules of M*

$$M_1 \subset M_2 \subset \cdots \subset M_k \subset \cdots$$

   there exists $k \in \mathbb{N}$ such that $M_l = M_k$ for all $l \geq k$.

   c) *Every non–empty set of submodules of M has a maximal element with respect to inclusion.*

Proof. Whereas the proof of the module part of Lemma 1.1.15 can be found in nearly any textbook about commutative or computer algebra, the following, quite similar statement for monomodules, Part (4), is not so common. For this, we refer to the proof of Proposition 1.3.4 in [112]. □

Next we define maps between rings resp. modules which respect the corresponding structure. Due to similar behaviour and properties of these maps, we do this in parallel.

Definition 1.1.16. Let R and S be rings, let 1_R and 1_S the respective units in R and S. A map $\varphi : R \to S$ is called a *ring homomorphism* if for all $a, b \in R$ it holds that

(1) $\varphi(a + b) = \varphi(a) + \varphi(b)$,

(2) $\varphi(a \cdot b) = \varphi(a) \cdot \varphi(b)$, and

(3) $\varphi(1_R) = 1_S$.

As every ring R is also an R–module, Definition 1.1.16 induces a map between modules:

Definition 1.1.17. Let M and N be R–modules. A map $\phi : M \to N$ is called a *module homomorphism* if for all $a, b \in M$ it holds that

(1) $\phi(a + b) = \phi(a) + \phi(b)$ and

(2) $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$.

Lemma 1.1.18. *Let $\varphi : R \to S$ be a ring homomorphism. Then*

(1) *$\varphi(0_R) = 0_S$ and*

(2) *$\varphi(-a) = -\varphi(a)$ for all $a \in R$.*

Proof. The claims can be seen rather nicely in the following way:

$$\varphi(0) = \varphi(0 + 0) = \varphi(0) + \varphi(0) \Longrightarrow (1),$$
$$0 = \varphi(0) = \varphi(a + (-a)) = \varphi(a) + \varphi(-a) \Longrightarrow (2).$$

□

Corollary 1.1.19. *The statement of Lemma 1.1.18 also holds for an R–module homomorphism $\phi : M \to N$ with $a \in M$.*

Definition 1.1.20. Let $\varphi : R \to S$ be a ring homomorphism, $I \subset R$, $J \subset S$ be ideals.

(1) The *preimage of J under φ* is defined by $\varphi^{-1}(J) := \{r \in R \mid \varphi(r) \in J\}$. We denote $\ker(\varphi) := \varphi^{-1}(0)$, the *kernel of φ*.

(2) The *image of I under φ* is denoted $\mathrm{im}(\varphi) := \varphi(R) = \{\varphi(r) \mid r \in R\}$. The image of φ restricted to I is denoted $\mathrm{im}(\varphi\mid_I) := \varphi(I) = \{\varphi(r) \mid r \in I\}$.

(3) φ is called *injective* if $\ker(\varphi) = 0$.

(4) φ is called *surjective* if $\mathrm{im}(\varphi) = S$.

(5) φ is called *bijective* if φ is injective and surjective. A bijective ring homomorphism is also called an *isomorphism*. If there exists an isomorphism φ between two rings R and S we say that *R is isomorphic to S*, denoted $R \cong S$

Remark 1.1.21. The above definitions can be translated one–to–one to an R–module homomorphism $\phi : M \to N$.

Proposition 1.1.22. *Let $\varphi : R \to S$ be a ring homomorphism, $I \subset R$, $J \subset S$ be ideals. The following properties hold:*

(1) $\varphi^{-1}(J)$ *is an ideal in R.*

(2) $\mathrm{im}(\varphi)$ *is a subring of S.*

Proof.

(1) $\varphi^{-1}(J) \neq \emptyset$ as $0 = \varphi(0) \in J$. If $a, b \in \varphi^{-1}(J)$ then $\varphi(a), \varphi(b) \in J$. As $\varphi(a + b) = \varphi(a) + \varphi(b) \in J$, it follows that $a + b \in \varphi^{-1}(J)$. If $a \in \varphi^{-1}(J)$ and $r \in R$ then $\varphi(ra) = \varphi(r) \cdot \varphi(a) \in J$. Thus $ra \in \varphi^{-1}(J)$, which proves the first assumption.

(2) Clear.

□

Corollary 1.1.23. *Let $\phi : M \to N$ be an R–module homomorphism. The following properties hold:*

(1) $\phi^{-1}(N)$ *is a submodule of M.*

(2) $\mathrm{im}(\phi)$ *is a submodule of N.*

Remark 1.1.24.

(1) In particular, from Corollary 1.1.23 it follows that $\ker(\phi)$ is a submodule of M.

(2) Note the differences between Proposition 1.1.22 (2) and Corollary 1.1.23 (2): $\varphi(I)$ need not be an ideal in S for a ring homomorphism $\varphi : R \to S$. For example, consider $\varphi : \mathbb{Z} \to \mathbb{Q}$, then for any $0 \neq I \subset \mathbb{Z}$ $\varphi(I)$ is not an ideal in $\mathbb{Q}$. This is due to one of the main differences of module and ring homomorphisms: The one is based on modules over the same ring, whereas the other maps from one ring to a (possibly) different ring.

(3) For an ideal $I \subset R$ there exists the surjective ring homomorphism

$$\varphi : R \to R/I,$$

with $\ker(\varphi) = I$, the *quotient map*.

Proposition 1.1.25. *Let M, N be R–modules, then*

(1) the map $\pi : M \to M/N$ is a surjective R–module homomorphism and

(2) for any module homomorphism $\phi : M \to N$ it holds that $\operatorname{im}(\phi) \cong N/\ker(\phi)$.

Proof.

(1) Clear.

(2) Let $m \in M$. We define a map $\eta : M/\ker(\phi) \to \operatorname{im}(\phi)$ by $\eta(m + \ker(\phi)) = \phi(m)$. Then η is well defined, an R–module homomorphism and surjective by construction. Take m such that $\eta(m + \ker(\phi)) = 0$. Then $\phi(m) = 0$, i.e. $m \in \ker(\phi)$. Thus $m + \ker(\phi) = 0 + \ker(\phi)$ from which injectivity of η follows.

□

To finish this section, let us present two final statements, which turn out to be very important considering our further research.

One of the main tools to improve many polynomial algorithms is using modular methods combined with the Chinese Remainder Theorem.

Theorem 1.1.26 (Chinese Remainder Theorem)**.** *Let R be a ring, let $I_1, \ldots, I_n$ be ideals in R. If $I_s + I_t = R$ for all $s \neq t$, then*

$$R/\bigcap_{k=1}^{n} I_k \cong \prod_{k=1}^{n} R/I_k.$$

In particular, consider $\varphi : R \to \prod_{k=1}^{n} R/I_k$ constructed from the n morphisms $R \to R/I_k$. Then φ is surjective with $\ker(\varphi) = \cap_{k=1}^{n} I_k$.

Proof. See, for example, Section 3.7 in [112] or Section 2.8 in [18]. □

Using Lemma 1.1.15 we can state an important connection between Noetherian rings and finitely generated modules.

Proposition 1.1.27. *Let R be a Noetherian ring, and let M be a finitely generated free R–module. Then M is Noetherian.*

Proof. We can assume $M = R^s$, $N \subset M$, and proceed by induction on s: If $s = 1$, then M is an ideal in R and thus finitely generated. If $s > 1$, then we consider the module homomorphism

$$\begin{aligned} \phi : R^s &\longrightarrow R^{s-1} \\ (a_1, \dots, a_n) &\longmapsto (a_1, \dots, a_{n-1}). \end{aligned}$$

By 1.1.23 (1) $\ker(\phi)$ is a submodule of M, which is Noetherian as $\ker(\phi) \cong R$. Moreover, $R^s / \ker(\phi) \cong R^{s-1}$, which is Noetherian by our induction hypothesis. Thus by Statement (2) of Lemma 1.1.15 the induction step is done. □

Computations of standard bases are much easier over fields of finite characteristic p, due to the fact that coefficient growth is bounded to p.

Definition 1.1.28. The *characteristic of a ring R*, denoted $\mathrm{char}(R)$, is the positive integer p that generates the kernel of the ring homomorphism

$$\varphi : \mathbb{Z} \to R,\ n \mapsto n \cdot 1_R = \underbrace{1_R + \cdots + 1_R}_{n \text{ times}}.$$

Example 1.1.29. For any prime number p the field $\mathbb{F}_p$ has characteristic $\mathrm{char}(\mathbb{F}_p) = p$.

Having set up a general basis for our discussions, we can go on and restrict ourselves to some special rings, ideals, and modules, in which we are really interested in.

1.2 Polynomial rings

After introducing the basic algebraic structures in an arbitrary manner in the last section, we now focus on special rings and ideals, namely consisting of elements called *polynomials*. We see how those can be derived from the free R–module R^s defined in Definition 1.1.6(7) by the following (general) construction:

Let R be a ring. Consider the set[2]

$$R^{(\mathbb{N})} := \{(a_i)_{i\in\mathbb{N}} \mid a_i \in R, a_i = 0 \text{ for almost all } i\}.$$

Together with the component–wise addition

$$(a_i)_{i\in\mathbb{N}} + (b_i)_{i\in\mathbb{N}} := (a_i + b_i)_{i\in\mathbb{N}}$$

and the mutliplication

$$(a_i)_{i\in\mathbb{N}} \cdot (b_i)_{i\in\mathbb{N}} := \Big(\sum_{i=0}^{k} a_i b_{k-i}\Big)_{k\in\mathbb{N}},$$

[2] Note that $0 \in \mathbb{N}$.

$R^{(\mathbb{N})}$ is a ring. Moreover, $R^{(\mathbb{N})}$ is a free R–module with canonical basis $\{e_i \mid i \in \mathbb{N}\}$, $e_i \in R^{(\mathbb{N})}$ with 1 in the $(i+1)$st position, 0 otherwise. Thus we can identify $(a_i)_{i\in\mathbb{N}}$ with $\sum_{i\in\mathbb{N}} a_i e_i$.

From this we get the following proposition:

Proposition 1.2.1. *Let $R^{(\mathbb{N})}$ be defined as above. Then it holds that*

(1) *$R^{(\mathbb{N})}$ is a commutative ring with identity e_0.*

(2) *For all $i \in \mathbb{N}$ it holds that $e_i = e_1^i$.*

(3) *There exists an injective ring homomorphism $\varphi : R \to R^{(\mathbb{N})}$ given by $a \mapsto a \cdot e_0$ for all $a \in R$.*

The proof is straightforward. Moreover, with the above discussion we have already defined our desired ring.

Definition 1.2.2. Let R be a ring, $R^{(\mathbb{N})}$ as defined above.

(1) Set $x = e_1$, the ring $R^{(\mathbb{N})}$ is called the *polynomial ring in one variable x over R*. We denote it by $R[x]$. Every element $p \in R[x]$ has a representation

$$p = \sum_{i\in\mathbb{N}}^{\text{finite}} c_i x^i, \ c_i \in R$$

where almost all $c_i = 0$. This representation is uniquely defined, up to the order of the summands.

(2) For $n > 1$ we define the *polynomial ring in n variables $x_1, \dots, x_n$ over R* recursively by

$$R[x_1, \dots, x_n] := (R[x_1, \dots, x_{n-1}])[x_n].$$

(3) Let φ be as defined in Proposition 1.2.1 (3). An element $c \in \operatorname{im}(\varphi)$ is called a *constant (polynomial)* of the polynomial ring.

In this thesis we are mainly interested in the polynomial rings, and thus in its elements and their behaviour. Let us have a closer look at the elements of $R[x_1, \dots, x_n]$:

Definition 1.2.3. Let $R[x_1, \dots, x_n]$ be the polynomial ring in n variables, $\alpha_i \in \mathbb{N}$ for all $i \in \{1, \dots, n\}$.

(1) A *monomial* $m \in R[x_1, \dots, x_n]$ in n variables $x_1, \dots, x_n$ is a power product $\prod_{i=1}^n x_i^{\alpha_i}$. The *degree of a monomial* $m \neq 0$ is denoted $\deg(m) = \sum_{i=1}^n \alpha_i$; $\deg(0) := -1$. The degree of a variable x_i in m is denoted $\deg_{x_i}(m) = \alpha_i$.

(2) The set of all monomials in n variables is denoted by $\operatorname{Mon}(x_1, \dots, x_n) := \{\prod_{i=1}^n x_i^{\alpha_i} \mid \alpha_i \in \mathbb{N}\}$.

(3) A *term* is a monomial times a *coefficent* (constant), $c \in R$, $c \prod_{i=1}^n x_i^{\alpha_i}$.

(4) A *polynomial* p over R is a finite R–linear combination of monomials in $R[x_1, \ldots, x_n]$,

$$p = \sum_{(\alpha_1,\ldots,\alpha_n)\in\mathbb{N}^n}^{\text{finite}} c_{(\alpha_1,\ldots,\alpha_n)} \prod_{i=1}^{n} x_i^{\alpha_i}.$$

(5) The *support of* p is defined by $\operatorname{supp}(p) = \{\text{all terms in } p\}$. Moreover, the *monomial support of* p is defined by $\text{m-supp}(p) = \{\text{all monomials in } p\}$.

(6) The *(total) degree of* p is defined by $\deg(p) := \max\{\alpha_1 + \cdots + \alpha_n \mid c_{(\alpha_1,\ldots,\alpha_n)} \neq 0\}$, if $p \neq 0$.

Note that the representation of polynomials p in n variables defined in Definition 1.2.3 (4) is just a generalization of the representation in 1 variable stated in Definition 1.2.2 (1).

Remark 1.2.4. On the one hand every monomial is a term (with coefficient 1), on the other hand, for example, 0 is a term, but not a monomial.

We are only interested in some specific polynomial rings, namely those over a ground field $\mathcal{K}$. The following statement is fundamental for working with polynomial rings over fields:

Theorem 1.2.5 (Hilbert basis theorem)**.** *If R is a Noetherian ring, then the polynomial ring $R[x_1, \ldots, x_n]$ is Noetherian, too.*

Proof. See, for example, Section 1.3 in [97] or Section 4.1 in [18]. □

In particular, it follows that $\mathcal{K}[x_1, \ldots, x_n]$ is Noetherian as every field is Noetherian. For an easier notation let us agree on the following:

Convention. We introduce a multi–indices notation for monomials by

$$\boldsymbol{x}^{\alpha} := \prod_{i=1}^{n} x_i^{\alpha_i}, \; \alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n.$$

In the following we also investigate so–called *local rings*. A localization of a ring is nothing else but allowing denominators, which enlarges the ring. One can think of this as the step from the integers $\mathbb{Z}$ to the rationals $\mathbb{Q}$. In terms of polynomial rings and algebraic geometry localization of rings is used to get a more detailed view of some small neighborhood around some point in $\mathcal{K}^n$, e.g. for the study of singularities in algebraic varieties.
We give a really short introduction to local polynomial rings, limited to those cases we are interested in. Local polynomial rings are only a minor point in this thesis. If you need some more extensive introduction on local rings see for example Section 1.4 in [97].

Definition 1.2.6. A *local ring* R is a ring which has exactly one maximal ideal.

Fields are always local (with maximal ideal $\langle 0 \rangle$), whereas the polynomial ring in n variables $\mathcal{K}[x_1, \ldots, x_n]$ is not local. This is due to the fact that $I := \langle x_1 - a_1, \ldots, x_n - a_n \rangle$ is a maximal ideal for all $(a_1, \ldots, a_n) \in \mathcal{K}^n$.
Thus we need a procedure to "localize" $\mathcal{K}[x_1, \ldots, x_n]$. One way is to localize $\mathcal{K}[x_1, \ldots, x_n]$ at the point $(0, \ldots, 0)$ resp. with the maximal ideal $I := \langle x_1, \ldots, x_n \rangle$:

Definition 1.2.7. The *localization of* $\mathcal{K}[x_1,\ldots,x_n]$ *at the point* $(0,\ldots,0) \in \mathcal{K}^n$ is defined as

$$\mathcal{K}[x_1,\ldots,x_n]_{\langle x_1,\ldots,x_n\rangle} := \left\{ \frac{a}{b} \mid a,b \in \mathcal{K}[x_1,\ldots,x_n], b(0,\ldots,0) \neq 0 \right\}.$$

Next we need to add some more structure, namely a monomial order, to our polynomial rings to receive a unique representation of polynomials. We see that there exist special, so–called "local" monomial orders, such that we can compute in the localized polynomial ring $\mathcal{K}[x_1,\ldots,x_n]_{\langle x_1,\ldots,x_n\rangle}$ without explicit denominators. Moreover, we need to add a corresponding structure to modules over $\mathcal{K}[x_1,\ldots,x_n]$. Both of these structures have much in common, but they are handled differently. Since they are the main keys not only to enable the notion and computation of a standard basis, but also to understand the differences between classical algorithms to compute standard bases and the signature–based approach, we explain them in detail in their own section each.

1.3 Monomial orders on polynomial rings

Until now we have defined the basic algebraic structures we want to work with. Furthermore, we have already started focussing ourselves on polynomial rings $\mathcal{K}[x_1,\ldots,x_n]$ and their ideals. Having defined the elements of $\mathcal{K}[x_1,\ldots,x_n]$, called polynomials,

$$p = \sum_{\alpha\in\mathbb{N}^n}^{\text{finite}} c_\alpha \boldsymbol{x}^\alpha$$

they are uniquely defined, *but only up to the order of their monomials*. Thus our next task is to add some new property called a monomial order $<$ to $\mathcal{K}[x_1,\ldots,x_n]$ to achieve uniquely defined polynomials in $\mathcal{K}[x_1,\ldots,x_n]$ w.r.t. $<$.

For this we should start with some more general orders and restrict them later on to the monomial case:

Definition 1.3.1. A *strict partial order* $<$ is a binary relation on a set S such that for all $a,b,c \in S$ the following conditions hold:

(1) $\neg(a < a)$, (irreflexivity)

(2) $(a < b) \Rightarrow \neg(b < a)$, (antisymmetry)

(3) $(a < b$ and $b < c) \implies (a < c)$. (transitivity)

A *strict total order* $<$ is a strict partial together with the condition;

(4) Exactly one relation is true: $a < b$, $b < a$ or $a = b$. (trichotomy)

Every *total order* (resp. *partial order*) $\leq$ is a binary relation on a set S associated to a strict total order (resp. strict partial order) $<$ such that

$$a \leq b \iff a < b \text{ or } a = b.$$

A *well–order* $\leq$ is a total order on a set S such that every non–empty subset of S has a least element w.r.t. $\leq$.

Let us give some examples.

Example 1.3.2.

(1) The natural order $\leq_{\text{nat}}$ on $\mathbb{N}$ or $\mathbb{Z}$ is a total order. In particular, $\leq_{\text{nat}}$ induces a componentwise (natural) order on $\mathbb{N}^n$.

(2) As an example for an order being not total assume the power set of $\mathbb{Z}$ together with the order $\subseteq$. If $a \neq b \in \mathbb{Z}$ then neither $\{a\} \subseteq \{b\}$ nor $\{b\} \subseteq \{a\}$ hold.

Important in our further investigations is the partial order defined by the division of two monomials:

Definition 1.3.3. $\boldsymbol{x}^\alpha \leq_{\text{div}} \boldsymbol{x}^\beta$ iff $\beta - \alpha \in \mathbb{N}^n$. In spite of using the notation $\leq_{\text{div}}$ we say that $\boldsymbol{x}^\alpha$ *divides* $\boldsymbol{x}^\beta$. As a shorthand notation we use $\boldsymbol{x}^\alpha \mid \boldsymbol{x}^\beta$.

The following statement is a basis for the characterization of monomial orders defined above.

Lemma 1.3.4 (Dickson's Lemma). *For any subset $A \subset \mathbb{N}^n$ there exists a finite set $B \subset A$ such that for every $\alpha \in A$ there exists an element $\beta \in B$ satisfying $\beta \leq_{nat} \alpha$.*

Proof. See, for example, Section 1.2 in [97] or Section 4.3 in [18]. ☐

Let us restrict the above, general definition of a total order to an order on the set of monomials $\text{Mon}(\boldsymbol{x})$:

Definition 1.3.5. A *monomial order $<$ on $\mathcal{K}[x_1, \ldots, x_n]$* is a total order on $\text{Mon}(\boldsymbol{x})$ such that

$$\boldsymbol{x}^\alpha < \boldsymbol{x}^\beta \Longrightarrow \boldsymbol{x}^\gamma \boldsymbol{x}^\alpha < \boldsymbol{x}^\gamma \boldsymbol{x}^\beta \text{ for } \alpha, \beta, \gamma \in \mathbb{N}^n.$$

Note that language is a bit sloppy as $<$ is an order and thus defined on the set of all monomials, $\text{Mon}(x_1, \ldots, x_n)$, not on the polynomial ring $\mathcal{K}[x_1, \ldots, x_n]$ itself.

This structure of a monomial order is the main tool to enable polynomial computations: If we have defined a polynomial ring together with a monomial order we can write each polynomial in the ring in a unique way. Thus it is possible for a computer algebra system like Singular [49] to store a polynomial as an ordered list, such that a lot of computations, like equality checks of polynomials or reduction processes, are fast and easy. We have a closer look at these computations in the following sections.

Definition 1.3.6. Let $<$ be a monomial order on $\mathcal{K}[x_1, \ldots, x_n]$. Then $<$ is

(1) *global* $\Longleftrightarrow$ $x_i > 1$ for $i = 1, \dots, n$,

(2) *local* $\Longleftrightarrow$ $x_i < 1$ for $i = 1, \dots, n$, and

(3) *mixed* if it is neither global nor local.

Next we give some well–known examples of monomial orders. For this, reconsider the multi–indices notation: Let $\alpha, \beta \in \mathbb{N}^n$ then we define $\alpha + \beta$ resp. $\alpha - \beta$ to be the componentwise, natural addition resp. subtraction on $\mathbb{Z}$.

Definition 1.3.7. The following are monomial orders[3] on $\mathcal{K}[x_1, \dots, x_n]$.

(1) **Global orders**

a) *Lexicographical order* $<_{\mathrm{lp}}$:

$$\boldsymbol{x}^\alpha <_{\mathrm{lp}} \boldsymbol{x}^\beta :\Leftrightarrow \text{ the nonzero entry of lowest index in } \alpha - \beta \text{ is negative.}$$

b) *Graded lexicographical order* $<_{\mathrm{Dp}}$:

$$\begin{aligned} \boldsymbol{x}^\alpha <_{\mathrm{Dp}} \boldsymbol{x}^\beta :\Leftrightarrow\, & \deg(\boldsymbol{x}^\alpha) < \deg(\boldsymbol{x}^\beta) \text{ or,} \\ & (\deg(\boldsymbol{x}^\alpha) = \deg(\boldsymbol{x}^\beta) \text{ and} \\ & \text{ the nonzero entry of lowest index in } \alpha - \beta \text{ is negative).} \end{aligned}$$

c) *Graded reverse lexicographical order* $<_{\mathrm{dp}}$:

$$\begin{aligned} \boldsymbol{x}^\alpha <_{\mathrm{dp}} \boldsymbol{x}^\beta :\Leftrightarrow\, & \deg(\boldsymbol{x}^\alpha) < \deg(\boldsymbol{x}^\beta) \text{ or,} \\ & (\deg(\boldsymbol{x}^\alpha) = \deg(\boldsymbol{x}^\beta) \text{ and} \\ & \text{ the nonzero entry of highest index in } \alpha - \beta \text{ is positive).} \end{aligned}$$

(2) **Local orders**

a) *Negative lexicographical order* $<_{\mathrm{ls}}$:

$$\boldsymbol{x}^\alpha <_{\mathrm{ls}} \boldsymbol{x}^\beta :\Leftrightarrow \text{ the nonzero entry of lowest index in } \alpha - \beta \text{ is positive.}$$

b) *Negative graded lexicographical order* $<_{\mathrm{Ds}}$:

$$\begin{aligned} \boldsymbol{x}^\alpha <_{\mathrm{Ds}} \boldsymbol{x}^\beta :\Leftrightarrow\, & \deg(\boldsymbol{x}^\alpha) > \deg(\boldsymbol{x}^\beta) \text{ or,} \\ & (\deg(\boldsymbol{x}^\alpha) = \deg(\boldsymbol{x}^\beta) \text{ and} \\ & \text{ the nonzero entry of lowest index in } \alpha - \beta \text{ is negative).} \end{aligned}$$

c) *Negative graded reverse lexicographical order* $<_{\mathrm{ds}}$

$$\begin{aligned} \boldsymbol{x}^\alpha <_{\mathrm{ds}} \boldsymbol{x}^\beta :\Leftrightarrow\, & \deg(\boldsymbol{x}^\alpha) > \deg(\boldsymbol{x}^\beta) \text{ or,} \\ & (\deg(\boldsymbol{x}^\alpha) = \deg(\boldsymbol{x}^\beta) \text{ and} \\ & \text{ the nonzero entry of highest index in } \alpha - \beta \text{ is positive).} \end{aligned}$$

[3] We use the SINGULAR notation for the presented monomial orders.

(3) **Product orders** Assume the polynomial ring $\mathcal{K}[x_1, \ldots, x_n, y_1, \ldots, y_m]$ together with two monomial orders $<_1$ on $\mathrm{Mon}(x_1, \ldots, x_n)$ and $<_2$ on $\mathrm{Mon}(y_1, \ldots, y_m)$. Then we get a *product order* $< := (<_1, <_2)$ by

$$\begin{aligned}\boldsymbol{x}^\alpha \boldsymbol{y}^{\tilde{\alpha}} < \boldsymbol{x}^\beta \boldsymbol{y}^{\tilde{\beta}} :\Longleftrightarrow & \boldsymbol{x}^\alpha <_1 \boldsymbol{x}^\beta \text{ or,} \\ & (\boldsymbol{x}^\alpha = \boldsymbol{x}^\beta \text{ and } \boldsymbol{y}^{\tilde{\alpha}} <_2 \boldsymbol{y}^{\tilde{\beta}}).\end{aligned}$$

E.g., if $<_1$ is global, then monomials containing an x_i are always larger than those which do not.
If $<_1$ and $<_2$ are both global (resp. local), then $<$ is global (resp. local). Otherwise $<$ is a mixed order.

(4) **(Matrix) weight orders** Let $W \in \mathrm{GL}(n, \mathbb{R})$ be a matrix. Then we define the *weight order* by

$$\boldsymbol{x}^\alpha <_W \boldsymbol{x}^\beta \;:\Longleftrightarrow\; \text{the nonzero entry of lowest index of } W\alpha - W\beta \text{ is negative.}$$

Example 1.3.8. Assume the three monomials m_1, m_2 and m_3 in $\mathcal{K}[x_1, x_2, x_3, x_4, x_5]$ where

$$\begin{aligned} m_1 &= x_1^3 x_2^2 x_4 x_5^4, \\ m_2 &= x_1^2 x_4^8, \\ m_3 &= x_5^{13}. \end{aligned}$$

Note that $\deg(m_1) = \deg(m_2) < \deg(m_3)$. Let us see how the above defined orders behave and how they are related to each other considering m_1, m_2 and m_3.

(1) $m_3 <_{\mathrm{lp}} m_2 <_{\mathrm{lp}} m_1$,

(2) $m_1 <_{\mathrm{ls}} m_2 <_{\mathrm{ls}} m_3$,

(3) $m_2 <_{\mathrm{Dp}} m_1 <_{\mathrm{Dp}} m_3$,

(4) $m_3 <_{\mathrm{Ds}} m_2 <_{\mathrm{Ds}} m_1$,

(5) $m_1 <_{\mathrm{dp}} m_2 <_{\mathrm{dp}} m_3$,

(6) $m_3 <_{\mathrm{ds}} m_1 <_{\mathrm{ds}} m_2$.

Remark 1.3.9. Note that any monomial order $<$ on $\mathcal{K}[x_1, \ldots, x_n]$ can be represented by a matrix weight order $<_W$ for some $W \in \mathrm{GL}(n, \mathbb{R})$.

Proposition 1.3.10. *Let $<$ be a monomial order on $\mathcal{K}[x_1, \ldots, x_n]$. Then the following are equivalent:*

(1) $<$ is a well–order.

(2) $<$ is global.

(3) If $\alpha \leq_{nat} \beta$, then $\boldsymbol{x}^\alpha < \boldsymbol{x}^\beta$ or $\boldsymbol{x}^\alpha = \boldsymbol{x}^\beta$.

Proof. $(1) \Rightarrow (2)$ as well as $(2) \Rightarrow (3)$ are trivial. So let us prove the direction $(3) \Rightarrow (1)$: For any non–empty set M of monomials in $\mathcal{K}[x_1, \ldots, x_n]$ there exists a finite set $B \subset M$ by Lemma 1.3.4 such that for any $\boldsymbol{x}^\alpha \in M$ there exists $\boldsymbol{x}^\beta \in B$ such that $\beta \leq_{\text{nat}} \alpha$. Then either $\boldsymbol{x}^\beta < \boldsymbol{x}^\alpha$ or $\boldsymbol{x}^\beta = \boldsymbol{x}^\alpha$. Thus B contains a smallest element of M w.r.t. $<$. □

Equipping $\mathcal{K}[x_1, \ldots, x_n]$ with a monomial order $<$ we get a solution for our initial problem: We receive a uniquely determined representation for polynomials in $\mathcal{K}[x_1, \ldots, x_n]$:

$$p = c_\alpha \boldsymbol{x}^\alpha + c_\beta \boldsymbol{x}^\beta + \cdots + c_\gamma \boldsymbol{x}^\gamma,$$

such that $\boldsymbol{x}^\alpha > \boldsymbol{x}^\beta > \cdots > \boldsymbol{x}^\gamma$ and $c_\alpha, c_\beta, \ldots, c_\gamma \in \mathcal{K}$. Thus we can define special parts of p:

Definition 1.3.11. Let $p \in \mathcal{K}[x_1, \ldots, x_n]$ as above. Then we denote

(1) the *leading monomial of p* $\operatorname{lm}(p) = \boldsymbol{x}^\alpha$,

(2) the *leading coefficient of p* $\operatorname{lc}(p) = c_\alpha$,

(3) the *leading term of p* $\operatorname{lt}(p) = c_\alpha \boldsymbol{x}^\alpha$,

(4) the *tail of p* $\operatorname{tail}(p) = p - \operatorname{lt}(p)$, and

(5) the *ecart*[4] *of p* $\operatorname{ecart}(p) = \deg(p) - \deg\big(\operatorname{lm}(p)\big)$.

Furthermore, a polynomial p with $\operatorname{lc}(p) = 1$ is called *monic*.

As a last step let us consider localizations of polynomial rings again: We have introduced the localization of $\mathcal{K}[x_1, \ldots, x_n]$ at the point $(0, \ldots, 0) \in \mathcal{K}^n$ in Definition 1.2.7. We show how to use local monomial orders to avoid computations with denominators in $\mathcal{K}[x_1, \ldots, x_n]_{\langle x_1, \ldots, x_n \rangle}$.

In the following let the subset U in $\mathcal{K}[x_1, \ldots, x_n]$ be defined by

$$U := \{u \in \mathcal{K}[x_1, \ldots, x_n] \mid u \neq 0, \operatorname{lm}(u) = 1\}.$$

The following property of U is straightforward.

Lemma 1.3.12. *U is multiplicatively closed, that is*

(1) $1 \in U$, and

(2) for $a, b \in S$ it holds that $ab \in U$.

Definition 1.3.13. Let $<$ be a monomial order on $\mathcal{K}[x_1, \ldots, x_n]$. We define the *localization of* $\mathcal{K}[x_1, \ldots, x_n]$ *w.r.t. U* by

$$U^{-1}\mathcal{K}[x_1, \ldots, x_n] := \left\{ \frac{a}{u} \mid a, u \in \mathcal{K}[x_1, \ldots, x_n], u \neq 0, \operatorname{lm}(u) = 1 \right\}.$$

Clearly, it holds that

[4] Actually we should write *écart* as this is French for "separation" or "difference".

(1) $<$ is a global order if and only if $U = \mathcal{K}^*$.

(2) $<$ is a local order if and only if $U = \mathcal{K}[x_1,\dots,x_n] \backslash \langle x_1,\dots,x_n\rangle$.

Proposition 1.3.14. *Let $<$ be a monomial order on $\mathcal{K}[x_1,\dots,x_n]$.*

(1) $<$ is a global order if and only if $\mathcal{K}[x_1,\dots,x_n] = U^{-1}\mathcal{K}[x_1,\dots,x_n]$.

(2) $<$ is a local order if and only if $\mathcal{K}[x_1,\dots,x_n]_{\langle x_1,\dots,x_n\rangle} = U^{-1}\mathcal{K}[x_1,\dots,x_n]$.

Proof.

(1) Clear, since $U = \mathcal{K}^* \Longleftrightarrow <$ is global.

(2) $\mathcal{K}[x_1,\dots,x_n] \backslash \langle x_1,\dots,x_n\rangle$ is the multiplicatively closed set of units in the localized polynomial ring $U^{-1}\mathcal{K}[x_1,\dots,x_n]$ if and only if every $p \in \mathcal{K}[x_1,\dots,x_n]$ with $\mathrm{lm}(p) = 1$ is in U. This is the case if and only if $<$ is local.

□

Clearly, the units in the localized polynomial ring are defined to be

$$\left(U^{-1}\mathcal{K}[x_1,\dots,x_n]\right)^* = \left\{\frac{a}{b} \mid \mathrm{lm}(a) = \mathrm{lm}(b) = 1\right\}.$$

We see in Section 1.7 how this fact sometimes must be used to ensure termination for standard basis algorithms.

As a very last note on local polynomial rings let us state the following important fact, whose proof can be found in [97]:

Proposition 1.3.15. *$U^{-1}\mathcal{K}[x_1,\dots,x_n]$ is Noetherian.*

Convention. From the above discussion it is clear that we always need to equip the polynomial ring $U^{-1}\mathcal{K}[x_1,\dots,x_n]$ with a monomial order $<$, as otherwise the elements we are interested in are not uniquely defined. In the following we do not explicitly state the monomial order for a better reading, i.e. when we write $U^{-1}\mathcal{K}[x_1,\dots,x_n]$ we always mean $U^{-1}\mathcal{K}[x_1,\dots,x_n]$ together with a monomial order $<$ on $U^{-1}\mathcal{K}[x_1,\dots,x_n]$. Furthermore, in this thesis $\mathcal{P}$ always denotes the localization of the polynomial ring in n variables $x_1,\dots,x_n$ over the ground field $\mathcal{K}$, $\mathcal{P} := U^{-1}\mathcal{K}[x_1,\dots,x_n]$. Writing $\mathcal{K}[x_1,\dots,x_n]$ we always assume our polynomial ring to be equipped with a well–order.

1.4 Monomial orders on free $\mathcal{P}$–modules

In the following we induce orders on free $\mathcal{P}$–modules from the monomial orders defined in the last section. Orders on modules are very important for signature–based standard basis algorithms, as we see in Chapter 4.

Right now we can define polynomials in $\mathcal{P}$ uniquely equipping the polynomial ring with a monomial order as defined in Section 1.3. There are two reasons why we are interested in orders on free $\mathcal{P}$–modules $\mathcal{M}$:

(1) We also want to compute standard bases of such modules.

(2) Signature–based standard basis algorithms are based on comparing elements of $\mathcal{M}$.

Convention. In the following let $\mathcal{M} = \oplus_{i=1}^{s} \mathcal{P} e_i$ always be a free $\mathcal{P}$–module of rank s with canonical basis elements e_i.

Definition 1.4.1.

(1) A *(module) monomial in* $\mathcal{M}$ is an element of the form $m = \boldsymbol{x}^\alpha e_i$ where $\boldsymbol{x}^\alpha$ is a monomial in $\mathcal{P}$.

(2) A *term* cm *in* $\mathcal{M}$ is a monomial $m \in \mathcal{M}$ times a *coefficient* $c \in \mathcal{K}$.

(3) The *index of a term* $t = c\boldsymbol{x}^\alpha e_i$ is denoted $\operatorname{index}(t) = i$.

(4) An element $f \in \mathcal{M}$ can be written as a finite $\mathcal{K}$–linear combination of such monomials m.

$$f = \sum_{i=1}^{s} \Big(\sum_{\alpha \in \mathbb{N}^n}^{\text{finite}} c_\alpha \boldsymbol{x}^\alpha \Big) e_i \tag{1.4.1}$$

where $c_\alpha \in \mathcal{K}$, $\boldsymbol{x}^\alpha \in \operatorname{Mon}(x_1, \ldots, x_n)$.

(5) The *support of* $f \in \mathcal{M}$ is defined by $\operatorname{supp}(f) = \{\text{all terms in } f\}$.

(6) The *monomial support of* $f \in \mathcal{M}$ is defined by $\text{m-supp}(f) = \{\text{all monomials in } f\}$.

(7) The notion of the *degree of a monomial* $m = \boldsymbol{x}^\alpha e_i$ is reduced to the one of the monomial $\boldsymbol{x}^\alpha \in \mathcal{P}$ as defined in 1.2.3 (1):

$$\deg(m) := \deg(\boldsymbol{x}^\alpha) = \sum_{i=1}^{n} \alpha_i .$$

Clearly, $\deg(f) := \max\{\deg(m) \mid m \text{ a module monomial of } f\}$.

Although we can write any element $f \in \mathcal{M}$ as a sum of terms as in Equation 1.4.1, this representation is, again, unique *only up to the order of the monomials*. Thus we need a monomial order on $\mathcal{M}$. Naturally, this is a generalization of a corresponding monomial order on $\mathcal{P}$ taking into account the canonical basis elements e_i:

Definition 1.4.2. Let $<$ be a monomial order on $\mathcal{P}$. A *module (monomial) order* $\prec$ on $\mathcal{M}$ is a total order on the set of all monomials of $\mathcal{M}$ such that

(1) $\boldsymbol{x}^\alpha e_i \prec \boldsymbol{x}^\beta e_j \Longrightarrow \boldsymbol{x}^\gamma \boldsymbol{x}^\alpha e_i \prec \boldsymbol{x}^\gamma \boldsymbol{x}^\beta e_j$ and

(2) $\boldsymbol{x}^\alpha < \boldsymbol{x}^\beta \Longrightarrow \boldsymbol{x}^\alpha e_i \prec \boldsymbol{x}^\beta e_i$.

for $\alpha, \beta, \gamma \in \mathbb{N}^n$ and $i, j \in \{1, \ldots, s\}$.

Let us note some important facts about the correspondences of the module monomial order on $\mathcal{M}$ and the monomial order on $\mathcal{P}$, from which it is induced.

Remark 1.4.3. From the above definition it is clear that any monomial order on the polynomial ring $\mathcal{P}$ can also be understood as a module monomial order on the module $\mathcal{P} \cong \mathcal{P}e_1$. Thus the module monomial order is a generalization of the usual monomial order defined in Section 1.3.

Proposition 1.4.4. *Let $\prec$ be a module order on $\mathcal{M}$, $<$ the corresponding monomial order on $\mathcal{P}$. Then the following hold:*

(1) *$\prec$ is a well–order $\Longleftrightarrow$ $<$ is a well–order.*

(2) *$\prec$ is global resp. local resp. mixed $\Longleftrightarrow$ $<$ is global resp. local resp. mixed.*

Proof. These facts follow easily from Property (2) of Defintion 1.4.2. □

Let us give examples for module orders we need in the signature–based attempt of computing standard bases.

Example 1.4.5. Again, let $<$ be the monomial order on $\mathcal{P}$ which induces the module order $\prec$ on $\mathcal{M}$. The main new structure one can tweak with are the canoncial basis elements of $\mathcal{M}$.

(1) $\prec_{\mathrm{i}}$ denotes the order which emphasizes the index of the canonical basis element:

$$\begin{aligned} \boldsymbol{x}^\alpha e_i \prec_{\mathrm{i}} \boldsymbol{x}^\beta e_j :\Longleftrightarrow\ & i < j \text{ or,} \\ & i = j \text{ and } \boldsymbol{x}^\alpha < \boldsymbol{x}^\beta. \end{aligned}$$

(2) $\prec_{\mathrm{m}}$ denotes the order which emphasizes the monomial:

$$\begin{aligned} \boldsymbol{x}^\alpha e_i \prec_{\mathrm{m}} \boldsymbol{x}^\beta e_j :\Longleftrightarrow\ & \boldsymbol{x}^\alpha < \boldsymbol{x}^\beta \text{ or,} \\ & \boldsymbol{x}^\alpha = \boldsymbol{x}^\beta \text{ and } i < j. \end{aligned}$$

When talking about signature–based algorithms in the following we see that there are other useful module orders. More about this is postponed to Chapter 4.

Similar to the polynomial case we can now identify and define special parts of elements $f \in \mathcal{M}$.

Definition 1.4.6. Given a module order $\prec$ on $\mathcal{M}$ every element $f \in \mathcal{M}$ can be uniquely represented by

$$f = c_\alpha \boldsymbol{x}^\alpha e_i + f',$$

$c_\alpha \in \mathcal{K}$, $\boldsymbol{x}^\alpha \in \mathrm{Mon}(x_1, \ldots, x_n)$ such that for all nonzero terms $c_\beta \boldsymbol{x}^\beta e_j$ of f' it holds that

$$\boldsymbol{x}^\alpha e_i \succ \boldsymbol{x}^\beta e_j.$$

As in the situation of polynomials, Definition 1.3.11, we can identify special parts of f:

(1) the *leading monomial of f* $\operatorname{lm}(f) = \boldsymbol{x}^\alpha e_i$,

(2) the *leading coefficient of f* $\operatorname{lc}(f) = c_\alpha$,

(3) the *leading term of f* $\operatorname{lt}(f) = c_\alpha \boldsymbol{x}^\alpha e_i$,

(4) the *tail of f* $\operatorname{tail}(f) = f - \operatorname{lt}(f)$, and

(5) the *ecart of f* $\operatorname{ecart}(f) = \deg(f) - \deg\big(\operatorname{lm}(f)\big)$.

Furthermore, a module element f with $\operatorname{lc}(f) = 1$ is called *monic*.

Convention. Likewise the polynomial case we always equip $\mathcal{M}$ with a module monomial order $<$ to receive uniquely defined elements, thus the notation of $\mathcal{M}$ implies a module monomial order.

It is urgent to get some more information of the relationship between two module monomials when talking about normal forms and standard bases in the following sections.

Definition 1.4.7. We say that $\boldsymbol{x}^\alpha e_i$ *divides* $\boldsymbol{x}^\beta e_j$ if and only if

$$i = j \text{ and } \boldsymbol{x}^\alpha \mid \boldsymbol{x}^\beta.$$

As a shorthand notation we use $\boldsymbol{x}^\alpha e_i \mid \boldsymbol{x}^\beta e_j$.

As a last note let us give some example why we need to be cautious with the representation of an element $f \in \mathcal{M}$ as given in Equation 1.4.1 where we grouped the different monomials by the index of the canonical basis elements e_i. We have used this straightforward notation only to simplify notations at that point of our study of free $\mathcal{P}$–modules.

Example 1.4.8. Let $\mathcal{P} = \mathcal{K}[x, y, z]$, $\mathcal{M} = \mathcal{P}^2$. Assume the three monomials

$$\begin{aligned} m_1 &= -2x^2 y e_1, \\ m_2 &= 4x^3 yz^2 e_1, \\ m_3 &= z^4 e_2. \end{aligned}$$

Let us construct the element $f \in \mathcal{M}$ being the sum of m_1, m_2 and m_3. At this point we have to fix a monomial order on $\mathcal{M}$ to give a uniquely defined representation of f:

(1) If we pick $<_{\mathrm{i}}$ induced by $<_{\mathrm{dp}}$ we get the following:

$$f = (4x^3 yz^2 - 2x^2 y)e_1 + z^4 e_2.$$

This coincides with the representation given in Equation 1.4.1.

(2) If we pick $<_{\mathrm{m}}$ induced by $<_{\mathrm{dp}}$ we get a different sequence of the monomials:

$$f = 4x^3 yz^2 e_1 + z^4 e_2 - 2x^2 y e_1,$$

which does not correlate with Equation 1.4.1.

With this, we conclude our introduction to monomial orders on polynomial rings and free modules. For more details on monomial orders see for example [97].

1.5 Gradings

In this section we want to characterize gradings. We define them in general, but focus on the main usage of them in this thesis: Homogenizing module elements resp. polynomials. These homogenized elements have some properties one can use to improve standard basis computations as explained in more detail in Chapter 2. Moreover, the restriction to homogeneous input was one of the drawbacks of the initial presention of Faugère's F5 Algorithm in [62] (see Section 6.1 for more details).

Definition 1.5.1.

(1) A ring R is called a *graded ring* if there exist abelian subgroups R_ν such that
   a) $R = \oplus_{\nu \geq 0} R_\nu$, and
   b) for all $\nu, \mu \geq 0$ it holds that $R_\nu R_\mu \subseteq R_{\nu+\mu}$.

(2) An R–module M is called a *graded R–module* if there exist abelian subgroups M_ν such that
   a) $M = \oplus_{\nu \in \mathbb{Z}} M_\nu$, and
   b) for all $\mu, \nu \geq 0$ it holds that $R_\nu M_\mu \subseteq M_{\nu+\mu}$.

(3) An element of f of R_ν resp. M_ν is called *homogeneous (of degree ν)*. A not homogeneous element is sometimes also called *inhomogeneous*. Moreover, we define that 0 is a homogeneous element of every degree.

(4) A module $M = \langle f_1, \ldots, f_r \rangle$ is called *homogeneous* if f_i is homogeneous for all $i \in \{1, \ldots, r\}$.

Remark 1.5.2.

(1) Every element f in M can be decomposed into

$$f = \sum_{\nu \geq 0} f_\nu,$$

such that $f_\nu \in M_\nu$ for all ν. This decomposition is unique due to the fact that M is a direct sum of the M_νs.

(2) Note that an homogeneous module resp. ideal $M = \langle f_1, \ldots, f_r \rangle$ is only generated by homogeneous elements, we do *not* require that each element $f \in M$ is homogeneous. What follows from the definition in 1.5.1 (4) is that f is generated by homogeneous elements. For example let $M = \langle x^2 + y^2, x^3 - x^2 y \rangle$ be a homogenenous ideal in $\mathcal{K}[x, y]$ with $<_{\text{dp}}$ then

$$f = x^3 - x^2 y + x^2 + y^2 \in M,$$

but f is inhomogeneous.

We are interested in some special gradings on the polynomial rings $\mathcal{P}$: Some polynomials have a special structure where all monomials it consists of share a property, in particular the degree.

Definition 1.5.3.

(1) A polynomial $p \in \mathcal{P}$ is called *homogeneous (of degree d)* if every monomial of p has degree d. We denote the set of all such polynomials $\mathcal{P}_d = \{p \in \mathcal{K}[x_1, \ldots, x_n] \mid \deg(t) = d \text{ for all } t \in \operatorname{supp}(p)\}$ for $d \geq 0$. This is sometimes called the *standard grading on $\mathcal{P}$*.

(2) Given any polynomial $p \in \mathcal{K}[x_1, \ldots, x_n]$ and some extra variable x_0

$$p^{\mathrm{h}} = x_0^{\deg(p)} p\left(\frac{x_1}{x_0}, \ldots, \frac{x_n}{x_0}\right) \in \mathcal{K}[x_0, \ldots, x_n]$$

denotes the *homogenization of p with respect to x_0*. p^{h} is then homogeneous of degree $\deg(p)$.

(3) Conversely, for every homogeneous polynomial $P \in \mathcal{K}[x_0, \ldots, x_n]$ there exists a *dehomogenization with respect to x_0* defined by

$$P^{\mathrm{deh}} = P(1, x_1, \ldots, x_n) \in \mathcal{K}[x_1, \ldots, x_n].$$

(4) Furthermore, the following connection hold:

$$P = x_0^l \; P(1, x_1, \ldots, x_n)^{\mathrm{h}}$$

where $l = \max\{s \in \mathbb{N} \mid \text{ every monomial of } P \text{ includes } x_0^t \text{ such that } t > s\}$[5].

Of course, one needs to adjust a new order $<_h$ switching from $U^{-1}\mathcal{K}[x_1, \ldots, x_n]$ to $U^{-1}\mathcal{K}[x_0, \ldots, x_n]$ when homogenizing:

Definition 1.5.4. Let $<$ be a monomial order on $U^{-1}\mathcal{K}[x_1, \ldots, x_n]$, which can be represented by a weight matrix $A \in \mathbb{R}^{m \times n}$. Furthermore, let $\boldsymbol{x}^\alpha$ and $\boldsymbol{x}^\beta$ be two monomials in $\mathcal{K}[x_1, \ldots, x_n]$. We define the induced *homogenized monomial order $<_h$* by

$$\begin{aligned} x_0^s \boldsymbol{x}^\alpha <_{\mathrm{h}} x_0^t \boldsymbol{x}^\beta :\Leftrightarrow\; & s + \deg(\boldsymbol{x}^\alpha) <_{\mathrm{nat}} t + \deg(\boldsymbol{x}^\beta) \text{ or,} \\ & \big(s + \deg(\boldsymbol{x}^\alpha) = t + \deg(\boldsymbol{x}^\beta) \text{ and} \\ & \boldsymbol{x}^\alpha < \boldsymbol{x}^\beta\big). \end{aligned}$$

Any such induced homogenized monomial order $<_{\mathrm{h}}$ can be represented by a weight matrix

$$A_{\mathrm{h}} := \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 0 & & & \\ \vdots & & A & \\ 0 & & & \end{pmatrix}$$

[5] In Section 1.3 we see that this is equivalent to the condition that x_0^s divides x_0^t.

Using the definitions of Section 1.4 homogenization and dehomogenization generalize naturally to the world of modules.

One can conclude from the above defnitions easily the following statements:

Corollary 1.5.5. *With the notations from above the following statements hold:*

(1) *The induced homogenized order $<_{\mathrm{h}}$ is a well–order on $\mathcal{M}$.*

(2) *If $f \in \mathcal{M}$ is homogeneous, then* $\operatorname{ecart}(f) = 0$.

(3) *For any $f \in \mathcal{M}$ we have the following correspondence between the leading monomial of f and the leading monomial of f^{h}:*

$$\operatorname{lm}_{<_{\mathrm{h}}}(f^{\mathrm{h}}) = x_0^{\operatorname{ecart}(f)} \operatorname{lm}_{<}(f).$$

Remark 1.5.6. Note that similar to Corollary 1.5.5 (3) for any module element f in $\mathcal{M}$ the $\operatorname{ecart}(f)$ can be intepreted in terms of the homogenization of f: Homogenizing f with respect to x_0 we just need to know the degree of x_0 in $\operatorname{lm}_{<_{\mathrm{h}}}(f^h)$,

$$\operatorname{ecart}(f) = \deg_{x_0}\left(\operatorname{lm}_{<_{\mathrm{h}}}(f^h)\right).$$

In the following sections of this chapter we use the homogeneity of elements only as a sideline, but define the basic ideas of standard basis computations. We get back to the ideas of homogenization when introducing improvements to the fundamental standard basis algorithm (Section 2.2).

1.6 Hilbert–Poincaré series and dimensions

Next we define one very important invariant in commutative algebra, the Hilbert–Poincaré series. Used in graded rings resp. modules it stores the dimensions of the homogeneous parts of the graded structures. There are various ways introducing this topic, we use an attempt strongly related to [97].

Let R be a Noetherian graded ring with $R_0 = \mathcal{K}$. Then we know by Proposition 1.1.27 that every finitely generated graded R–module M is Noetherian, too. Thus each homogeneous part M_ν is a finite dimensional $\mathcal{K}$–vector space, it makes sense to speak of $\dim_{\mathcal{K}}(M_\nu)$. This is an important invariant in the following.

Definition 1.6.1. Let R be a Noetherian graded ring, and let $M = \oplus_{\nu\in\mathbb{Z}} M_\nu$ be a finitely generated R–module.

(1) We define the *Hilbert function of M* by

$$\begin{array}{rccc} \mathrm{H}_M : & \mathbb{Z} & \longrightarrow & \mathbb{Z} \\ & \nu & \longmapsto & \dim_{\mathcal{K}}(M_\nu). \end{array}$$

(2) The *Hilbert–Poincaré series of M* is defined by

$$\mathrm{HS}_M(t) := \sum_{\nu\in\mathbb{Z}} H_M(\nu)t^\nu \in \mathbb{Z}[[t,t^{-1}]].$$

Example 1.6.2. For each degree $d \geq 0$ we have

$$\mathrm{H}_{\mathcal{P}}(d) = \binom{d+n-1}{n-1} = \frac{(d+n-1)(d+n-2)\cdots(d+1)}{(n-1)!}.$$

Remark 1.6.3. For $\nu \in \mathbb{N}$ it holds that $1 \leq \mathrm{H}_M(\nu)$ as R_ν is never empty.

A well–known statement on the Hilbert–Poincaré series for positively graded modules over the polynomial ring $\mathcal{P}$ is the following.

Theorem 1.6.4. *Let $M = \oplus_{\nu\geq 0} M_\nu$ be a finitely generated, positively graded $\mathcal{P}$–module. Then there exists a polynomial $p(t) \in \mathbb{Z}[t]$ such that the Hilbert–Poincaré series can be written as*[6]

$$\mathrm{HS}_M(t) = \frac{p(t)}{(1-t)^n}. \tag{1.6.1}$$

Proof. See, for example, Section 5.2 in [97]. □

Theorem 1.6.5 (Hilbert)**.** *Let M be a finitely generated, graded module over $\mathcal{P}$. Then there exists a polynomial* $\mathrm{HP}_M(t)$ *with* $\deg\left(\mathrm{HP}_M(t)\right) \leq n-1$ *such that* $\mathrm{HS}_M(t) = \mathrm{HP}_M(t)$ *for* $t \gg 0$.

Moreover, one can deduce from the above theorem the Hilbert polynomial: Cancel out the common factor of $p(t)$ and $(1-t)^n$ in Equation 1.6.1 and use the results $q(t) = \sum_{i=0}^{d} q_i t^i$ and $(1-t)^m$ for $m \leq n$ to construct the above mentioned polynomial:

Definition 1.6.6. With the above construction the polynomial[7]

$$\mathrm{HP}_M(t) = \sum_{i=0}^{d} q_i \binom{t-i+m-1}{m-1}$$

is called the *Hilbert polynomial of M*.

In Section 2.7 we discuss how to use the information stored in the Hilbert polynomial to improve the computation of a standard basis of a homogeneous ideal I.

With this we finish our introduction to Hilbert–Poincaré series, giving a last remark on the situation in the case of local rings.

If $<$ is a local order on $\mathcal{P}$ the *Hilbert–Samuel function* is the counterpart of the Hilbert function in the homogeneous case. The connection to the Hilbert function is given by the following theorem.

[6] Note that n is the number of variables in $\mathcal{P}$.
[7] $\binom{n}{k} = 0$ for $k < 0$

Theorem 1.6.7. *Let $\mathcal{P}$ be equipped with a local order $<$, $Q \subset \mathcal{P}$ a primary ideal, and M a finitely generated $\mathcal{P}$–module. Then the Hilbert–Samuel function χ fulfills the following equation:*

$$\chi_M^Q(d+1) = \sum_{i=0}^{d} \mathrm{H}_{gr_Q(M)}(i),$$

where $gr_Q(M) = \oplus_{\nu=0}^{\infty} Q^\nu / Q^{\nu+1}$.

We do not go any further with this local situation, as it is not of main interest in this thesis.

1.7 Normal forms and standard bases

Having equipped $\mathcal{M} = \mathcal{P}^s$ with a monomial order $<$ in Section 1.4, we receive uniquely defined elements in $\mathcal{M}$. This enables us to define the standard basis of a submodule $M \subset \mathcal{M}$. A standard basis is nothing else but a nice set of generators of M, where *nice* should be understood as being equipped with some properties useful for computations with M.

A standard basis is a generalization of a Gröbner basis which was discovered by Bruno Buchberger in 1965 in his PhD thesis ([34]). He named it after his advisor Wolfgang Gröbner. Independently, Buchberger, Grauert and Hironaka introduced the notion of a standard basis ([34, 95, 101, 102]).

In the following sections we present algorithms for computing such bases, which are, in the special situation of $<$ being global and M being an ideal in $\mathcal{P}$, just generaliziations of the Gaussian elimination algorithm and the Euclidean algorithm.

Next, we focus on the characterization of the normal form of an element $f \in \mathcal{M}$ w.r.t. some $G \subset \mathcal{M}$. It turns out that computing the normal forms of special elements called s–vectors is the main step when searching for a standard basis of a given submodule.

Although the situation having a global order is of main interest in this thesis, the special behaviour in case of local (and thus also mixed) orders is important to be understood for a deeper inside in the advantages and disadvantages of signature–based standard basis algorithms.

Remark 1.7.1. We introduce standard bases in the world of modules. Clearly, we are interested in standard bases of ideals in $\mathcal{P}$, too. This is just a specialization of the module case, thus included. We give extensive explanations and examples of peculiarities, wherever these are important for our further investigations.

The next lemma is very important in what follows.

Lemma 1.7.2. *Any free, finitely generated $\mathcal{P}$–module is Noetherian.*

Proof. This follows from the fact that $\mathcal{P}$ is Noetherian combined with Proposition 1.1.27. □

As a last preliminary step we need to define some more structure needed in the following discussion.

Definition 1.7.3. A *sequence* S is an ordered list of objects. It contains elements and has a (possibly infinite) length denoted $\#(S)$, like a set. On the contrary, the elements are ordered and the same element can appear several times at different positions in the sequence.

Let us start with the definition of the element of main interest in this thesis.

Definition 1.7.4.

(1) For any subset $S \subset \mathcal{M}$ we define

$$L_<(S) := \left\langle \operatorname{lm}(s) \mid s \in S\backslash\{0\} \right\rangle,$$

the *leading submodule of* S. In particular, if $L_<(S) \subset \mathcal{P}$ is an ideal we speak of the *leading ideal*[8] *of* S. If the order is clear by context, we just write $L(S)$.

(2) Let $M \subset \mathcal{M}$ be a submodule. A finite sequence $G = \{g_1, \dots, g_r\} \subset M$ is a *standard basis for* M if

$$L(M) = \left\langle \operatorname{lm}(g_1), \dots, \operatorname{lm}(g_r) \right\rangle = L(G).$$

Remark 1.7.5.

(1) Note that $L(M) = L(G) = \langle \operatorname{lm}(g_1), \dots, \operatorname{lm}(g_r)\rangle$ is equivalent to the fact that for every $f \in M$ there exists $l \in \{1, \dots, r\}$ such that $\operatorname{lm}(g_l) \mid \operatorname{lm}(f)$.

(2) For a shorter notation we say that G *is a standard basis* when we mean that G *is a standard basis for* $\langle G\rangle$.

(3) If $<$ is a well–order on $\mathcal{M}$, then G is also called a *Gröbner basis*.

Proposition 1.7.6. *Let $M \subset \mathcal{M}$ be a nonzero submodule.*

(1) There exists a standard basis G for M.

(2) Let $G \subset M$ be a standard basis for M. Then $\langle G\rangle = M$.

Proof.

(1) By Lemma 1.7.2 $\mathcal{M}$ is Noetherian, thus $L(M)$ is finitely generated. We can choose finitely many monomials $m_1, \dots, m_r$ generating $L(M)$. By definition the m_i are leading monomials of appropriate elements $g_1, \dots, g_r \in M$. It follows that $G := \{g_1, \dots, g_r\}$ is a standard basis of M.

(2) Clear.

□

Note that a standard basis depends on the chosen monomial order $<$.

[8] In the following, an ideal generated by monomials only is called a monomial ideal.

Example 1.7.7. Let us have a closer look at the ideal $I = \langle p_1, p_2 \rangle \subset U^{-1}\mathcal{K}[x, y, z]$ where $p_1 = x^2 - y$ and $p_2 = xy - z$.

(1) On the one hand, if we choose $<_{\text{ds}}$ $\text{lm}(p_1) = y$ and $\text{lm}(p_2) = z$, thus we easily see that $\langle y, z \rangle = L(I)$. In other words, $\{p_1, p_2\}$ is already a standard basis for I.

(2) On the other hand we can choose the order $<_{\text{dp}}$. Then $\text{lm}(p_1) = x^2$ and $\text{lm}(p_2) = xy$, but $\langle x^2, xy \rangle \neq L(I)$ as

$$p_3 := -y(x^2 - y) + x(xy - z) = y^2 - xz \in I$$

and $\text{lm}(p_3) = y^2 \notin \langle x^2, xy \rangle$. One can easily show that $G = \{p_1, p_2, p_3\}$ is a standard basis for I.

This is a crucial problem in the theory of standard basis computations. Given a module M, it can be possible to compute a standard basis w.r.t. an order $<_1$ for it in seconds on a small computer, whereas the computation w.r.t. to another order $<_2$ can be unsolvable, even on super computers. There exist methods to use a standard basis w.r.t. $<_1$ to compute a standard basis w.r.t. $<_2$, but this is not an easy process and can sometimes be harder (slower, consuming more memory, etc.) than computing from scratch w.r.t. $<_2$. We investigate this problem and possible solutions in more detail in Chapter 2.

Right now we have shown the existence of a standard basis G for any $0 \neq M \subset \mathcal{M}$ and any monomial order $<$. G needs not to be uniquely defined as there could be another standard basis G' consisting of all elements of G and some linear combinations of those. We can require some more properties on G to receive the unique, so–called reduced standard basis of M.

Definition 1.7.8. Let G be a finite sequence in the free $\mathcal{P}$–module $\mathcal{M}$.

(1) G is called *interreduced* if
   a) $0 \notin G$ and
   b) for every $g \in G$ it holds that $\text{lm}(g) \notin L(G \backslash \{g\})$.

(2) Let $f \in \mathcal{M}$. Then we say that *f is top–reduced with respect to G* if $\text{lm}(t) \notin L(G)$. Furthermore, we say that *f is reduced with respect to G* if no monomial of the power series expansion of f is contained in $L(G)$.

(3) We say that *G is reduced* if
   a) $0 \notin G$,
   b) each $g \in G$ is top–reduced w.r.t. $G \backslash \{g\}$,
   c) for each $g \in G$ it holds that $\text{tail}(g)$ is reduced w.r.t. G and
   d) for every $g \in G$ it holds that $\text{lc}(g) = 1$.

Lemma 1.7.9. *Let $M \in \mathcal{M}$ be a submodule. If G is a reduced standard basis for M, then G is unique.*

Proof. Assume that there exists another reduced standard basis H for M. By 1.7.8 (3)b $\#(G) = \#(H)$. Let $G = \{g_1, \ldots, g_r\}$ and $H = \{h_1, \ldots, h_r\}$, both sorted by increasing leading monomials. For all $i \in \{1, \ldots, r\}$ $g_i - h_i \in M$. If $g_i - h_i \neq 0$ then $\operatorname{lm}(g_i - h_i) \in L(M)$. By 1.7.8 (3)c $\operatorname{lm}(g_i - h_i) \notin L(G)$ as well as $\operatorname{lm}(g_i - h_i) \notin L(H)$. This contradicts our assumption that G and H are reduced standard bases for M. □

The definition of the reduced standard basis needs a bit of explanation: A reduced standard basis might not always exist. Moreover, its computation might be not possible in finitely many steps using polynomials only in general:

(1) Starting with a standard basis G we want to transform G to an interreduced basis G':
  a) Delete all zeros from G.
  b) Delete all elements g' such that there exists $g \in G \backslash \{g'\}$ with $\operatorname{lm}(g) \mid \operatorname{lm}(g')$.

(2) From G' we get the reduced standard basis G'':
  a) For all polynomials $g \in G'$ we set $g := \frac{1}{\operatorname{lc}(g)} g$.
  b) If there exists a polynomial $g' \in G'$ such that $\operatorname{lm}(g') \mid \operatorname{lm}(\operatorname{tail}(g))$ we need to reduce[9] $\operatorname{tail}(g)$. This is the hard part computing a reduced standard basis and sometimes even impossible (see Example 1.7.10).

Example 1.7.10. Assume the polynomial ring $U^{-1}\mathcal{K}[x]$ with the local monomial order $<_{\text{ds}}$ and the ideal $I = \langle g \rangle$ where $g = x - x^2$ is a polynomial. Clearly, a corresponding standard basis is $G = \{g\}$. Trying to compute the reduced standard basis we see that $\operatorname{tail}(g) = x^2$ is divisible by $\operatorname{lm}(g)$. Reducing g by xg we get

$$g := g + xg = x - x^2 + x^2 - x^3 = x - x^3.$$

Now $\operatorname{tail}(g) = x^3$ is again divisible by $\operatorname{lm}(g)$ and we easily see that this process of reduction does not end in finitely many steps. Although we have seen that there exists a reduced standard basis of I we cannot compute it this way!

Remark 1.7.11. Note that in the case of $\mathcal{M}$ being equipped with a well–order $<$ the computation of a reduced Gröbner basis for any given submodule M is always possible in finitely many steps. This is due to the fact that for any $g \in G$ it holds that $\operatorname{lm}(g) \nmid \operatorname{tail}(g)$ by Definition 1.3.3 and Proposition 1.3.10, thus a situation as in Example 1.7.10 is not possible.

Having stated the term "reduction" quite too many times without a correct definition it is time to introduce the notion of a normal form:

Definition 1.7.12. Let $\mathcal{G}$ denote the set of all finite sequences G in $\mathcal{M}$. The map

$$\begin{aligned} \eta : \mathcal{M} \times \mathcal{G} &\to \mathcal{M} \\ (f, G) &\mapsto \eta(f, G) \end{aligned}$$

is called a *normal form* of $\mathcal{M}$ if for all $f \in \mathcal{M}$ and all $G \in \mathcal{G}$ the following hold:

[9] We explain in the following what we exactly mean by the term "reduction", right now the reader's intuition and natural understanding is quite propriate.

(1) $\eta(0, G) = 0$.

(2) If $\eta(f, G) \neq 0 \Rightarrow \mathrm{lm}\left(\eta(f, G)\right) \notin L(G)$.

(3) Let $G = \{g_1, \ldots, g_r\}$, $u \in \mathcal{P}^*$ a unit. Then there exists a representation

$$uf - \eta(f, G) = \sum_{i=1}^{r} p_i g_i, \; p_i \in \mathcal{K}[x_1, \ldots, x_n],$$

such that

$$\max\{\mathrm{lm}(p_i g_i) \mid 1 \leq i \leq r\} \leq \mathrm{lm}\left(uf - \eta(f, G)\right).$$

This is called the *standard representation of* $uf - \eta(f, G)$ *w.r.t.* G.

We say that $\eta(f, G)$ is the *normal form of* f *w.r.t.* G. Moreover, if we demand $\eta(f, G)$ to be reduced w.r.t. G for all $G \in \mathcal{G}$, then we call $\eta_{\mathrm{red}}(f, G)$ a *reduced normal form*.

Lemma 1.7.13. *Let M be a submodule of $\mathcal{M}$, $G \subset M$ a standard basis for M, and η a normal form of M. Then the following hold:*

(1) For any $f \in \mathcal{M}$ it holds: $f \in M \Longleftrightarrow \eta(f, G) = 0$.

(2) $M = \langle G \rangle$.

Proof.

(1) On the one hand, if $\eta(f, G) = 0$, then $uf \in M$. Thus $f \in M$. On the other hand, if $\eta(f, G) \neq 0$, then $\eta(f, G) \notin M$. Since $\langle G \rangle \subset M$ this implies $f \notin M$.

(2) $\langle G \rangle \subset M$ is clear. Now assume $g \in \langle G \rangle$ such that $g \notin M$. By (1) this means that $\eta(g, G) \neq 0$, a contradiction.

□

Next we state 3 different algorithms of how to compute normal forms: Algorithms 1 and 2 compute the normal form resp. the reduced normal form if a global order $<$ is given. We see that these algorithms can lead to an endless loop computing normal forms if a non–global order is given. We illustrate all these algorithms, fundamental for the computation of a standard basis, with an example:

Let $\mathcal{P} = U^{-1}\mathcal{K}[x, y, z]$, $\mathcal{M} = \mathcal{P}^2$, $f = x^3 e_2$. Let G be the sequence consisting only of the module element $r = x^2 e_2 - xze_1 - xe_2$. First, we assume the following orders: $<_{\mathrm{dp}}$ and $<_{\mathrm{m}}$. Let us compute the normal form of f w.r.t. G:

$$\begin{aligned} h &:= x^3 e_2 \\ D_h &:= \{r\} \\ h &:= h - xr = x^3 e_2 - x^3 e_2 + x^2 z e_1 + x^2 e_2 \\ &= x^2 z e_1 + x^2 e_2 \\ D_h &:= \{\} \end{aligned}$$

Algorithm 1 Normal form w.r.t. G for a global order $<$ (GNF)

Input: $f \in \mathcal{M}$, a finite sequence G in $\mathcal{M}$
Output: $h \in \mathcal{M}$, a normal form of f w.r.t. G

1: $h \leftarrow f$
2: **while** $(h \neq 0$ and $D_h := \{g \in G \mid \mathrm{lm}(g) \mid \mathrm{lm}(h)\} \neq \emptyset)$ **do**
3: Choose any $g \in D_h$.
4: Let $t \in \mathcal{P}$ such that $t\,\mathrm{lt}(g) = \mathrm{lt}(h)$.
5: $h \leftarrow h - tg$
6: **return** h

Algorithm 2 Reduced normal form w.r.t. G for a global order $<$ ($\mathrm{GNF_{red}}$)

Input: $f \in \mathcal{M}$, a finite sequence G in $\mathcal{M}$
Output: $h \in \mathcal{M}$, a reduced normal form of f w.r.t. G

1: $h \leftarrow 0, g \leftarrow f$
2: **while** $(g \neq 0)$ **do**
3: $g \leftarrow \mathrm{GNF}(g, G)$
4: **if** $(g \neq 0)$ **then**
5: $h \leftarrow h + \mathrm{lt}(g)$
6: $g \leftarrow \mathrm{tail}(g)$
7: **return** h

So at this point the normal form computation stops and we get

$$\eta(f, G) = x^2 z e_1 + x^2 e_2.$$

From here the reduced normal form (Algorithm 2) would go on, having already computed the normal form in Line 3. Note that in Algorithm 2 g has the role h plays in Algorithm 1. h is just the bucket the nonzero leading terms of $\eta(g, G)$ are stored in (Line 5), at this point $h = 0$.

$$g := \eta(g, G) = x^2 z e_1 + x^2 e_2.$$

As $g \neq 0$ we go on in Line 5 and set

$$\begin{aligned} h &:= h + \mathrm{lt}(g) = x^2 z e_1 \\ g &:= \mathrm{tail}(g) = x^2 e_2. \end{aligned}$$

Next we are back in Line 3 and compute the normal form of g:

$$\begin{aligned} g &:= x^2 e_2 \\ D_g &:= \{r\} \\ g &:= g - r = x^2 e_2 - x^2 e_2 + xze_1 + xe_2 \\ &= xze_1 + xe_2. \end{aligned}$$

At this point we see that neither xze_1 nor xe_2 is divisible by x^2e_2 thus in the following steps we just add those terms to h:

$$\begin{aligned}
h &:= h + \mathrm{lt}(g) = x^2ze_1 + xze_1 \\
g &:= \mathrm{tail}(g) = xe_2 \\
D_g &:= \{\} \\
h &:= h + \mathrm{lt}(g) = x^2ze_1 + xze_1 + xe_2 \\
g &:= \mathrm{tail}(g) = 0
\end{aligned}$$

Thus the reduced normal form of the initial f is

$$\eta_{\mathrm{red}}(f, G) = x^2ze_1 + xze_1 + xe_2.$$

As we have explicitly stated, these algorithms are assumed to terminate only if we have a global order. Assume the same elements, but now with $<_{\mathrm{ds}}$ and $<_{\mathrm{i}}$. In this setting the terms of the elements are reordered:

$$\begin{aligned}
f &= x^3e_2 \\
r &= xe_2 + x^2e_2 - xze_1.
\end{aligned}$$

Once more, let us try to compute the normal form $\eta(f, G)$ using Algorithm 1:

$$\begin{aligned}
h &:= x^3e_2 \\
D_h &:= \{r\} \\
h &:= h + x^2r = x^3e_2 - x^3e_2 + x^4e_2 - x^3ze_1 \\
&= x^4e_2 - x^3ze_1 \\
D_h &:= \{r\} \\
h &:= h + x^3r = x^4e_2 + x^3ze_1 - x^4e_2 + x^5e_2 - x^4ze_1 \\
&= x^5e_2 + x^3ze_1 + x^4ze_1 \\
D_h &:= \{r\} \\
&\ldots
\end{aligned}$$

We see that this computation does not terminate: The exponent k of $\mathrm{lm}(h) = x^ke_2$ increases by 1 every time we reduce h by r. Right now, the initially a bit strange definition of a normal form (1.7.12 (3)) with the multiplier $u \in \mathcal{P}^*$ rescues us. Remember our discussion about the localized polynomial ring $\mathcal{P} = U^{-1}\mathcal{K}[x_1, \ldots, x_n]$ at the end of Section 1.3. Having a local order $<_{\mathrm{ls}}$ $\mathcal{P}^*$ is no longer restricted to $\mathcal{K}^*$, but includes all elements $u \in \mathcal{P}$ such that $\mathrm{lm}(u) = 1$.
The main idea behind computing the normal form of an element for a non–global order is to compare the ecarts of the reducer and the element to be reduced and possibly add new elements to the list of reducers D_h. This was first presented by Mora in [130]. We state the variant of the surroundings of the Singular team ([94, 96, 97]) using the slightly different definition of the ecart we have already given in Definitions 1.3.11 and 1.4.6.

Algorithm 3 Normal form w.r.t. G for a non–global order $\prec$ (LNF)

Input: $f \in \mathcal{M}$, a finite sequence G in $\mathcal{M}$
Output: $h \in \mathcal{M}$, a normal form of f w.r.t. G

```
1: h ← f
2: D ← G
3: while (h ≠ 0 and D_h ← {g ∈ D | lm(g) | lm(h)} ≠ ∅) do
4:     Choose g ∈ D_h such that ecart(g) = min{ecart(g') | g' ∈ D}.
5:     if (ecart(g) > ecart(h)) then
6:         D ← D ∪ {h}
7:     Let t ∈ P such that t lt(g) = lt(h).
8:     h ← h − tg
9: return h
```

We see that the choice of the reducer depends on the corresponding ecart (Line 4). Moreover, in Line 6 possibly new reducers are added to D. These are the two main changes compared to Algorithm 1 which ensure termination in the non–global case. Let us review our example again, now using LNF:

$$\begin{aligned} h &:= x^3 e_2 \\ D &:= \{r\} \\ D_h &:= \{r\} \end{aligned}$$

Inasmuch as $\operatorname{ecart}(h) = 0$ and $\operatorname{ecart}(r) = 2$ we add h (now denoted by h_{old}) to D.

$$D := \{r, h_{\text{old}}\}$$

$$\begin{aligned} h &:= h + x^2 r = x^3 e_2 - x^3 e_2 + x^4 e_2 - x^3 z e_1 \\ &= x^4 e_2 - x^3 z e_1 \\ D_h &:= \{r, h_{\text{old}}\} \end{aligned}$$

Again, $\operatorname{ecart}(h_{\text{old}}) < \operatorname{ecart}(r)$, thus we use h_{old} to reduce h:

$$\begin{aligned} h &:= h - x h_{\text{old}} = x^4 e_2 - x^3 z e_1 - x^4 e_2 \\ &= -x^3 z e_1 \end{aligned}$$

Now $D_h = \{\}$ and the algorithm terminates with a normal form of f:

$$\eta(f, G) = -x^3 z e_1.$$

Let us review the reduction steps in LNF once more to see how this selection of a reducer w.r.t. a minimal ecart and addition of new elements to the list of reducers leads to a terminating normal form computation including the above mentioned unit $u \in \mathcal{P}^*$:

$$h = f + x^2 r - x f = -x^3 z e_1.$$

This normal form computation can be reformulated combining both summands including f:

$$(1-x)f - x^2 r = x^3 z e_1$$

This is the normal form of f as given in Definition 1.7.12 with unit $u = 1 - x \in \mathcal{P}^*$.

Remark 1.7.14.

(1) Algorithm 3 is the most general one, i.e. it ensures correctness and termination for any given order. Note that in the case of a global order $<$ appending h to the set of possible reducers D is useless as $\mathrm{lm}(h) > \mathrm{lm}\left(h - \frac{\mathrm{lt}(h)}{\mathrm{lt}(g)} g\right)$, which implies $\mathrm{lm}(h) \nmid \mathrm{lm}\left(h - \frac{\mathrm{lt}(h)}{\mathrm{lt}(g)} g\right)$ due to Proposition 1.3.10. Thus h is never used as a reducer in the following. In this situation LNF is just GNF with a special choice of reducers (minimal ecart) and some overhead due to appending elements not usable as reducers to D. Requiring an optimized implementation of the normal form algorithms one always implement LNF and GNF, using LNF only for non–global orders.

(2) Note that the choice of reducers in Algorithm 1 can influence the result.

We do not prove correctness or termination of the presented algorithms. We consider ourselves satisfied with the understanding of how they work and the intuitive insight that both are correct and terminate. For example, proofs can be found in sections 1.6 and 1.7 resp. 2.3 of [97].

As a last note let us recover Example 1.7.10: Using LNF we can easily compute the normal form of $f = x$ w.r.t. $G = \{r\}$ with $r = x - x^2$ without any trouble considering termination:

$$(1-x)x - r = 0.$$

The main idea behind the definition of a normal form is to get a characterization of a standard basis. Moreover, this turns out to be strongly related to what we want to be understood as a "reduction":

Definition 1.7.15. Let $f, h \in \mathcal{M}$, let M denote the set of all monomials in $\mathcal{M}$, let G be a sequence in $\mathcal{M}$, $g \in G$.

(1) *f top–reduces to h w.r.t. G* if there exist $g \in G$, $m \in M$ such that $\mathrm{lm}(f) = m\,\mathrm{lm}(g)$ and $h = f - \frac{\mathrm{lc}(f)}{\mathrm{lc}(g)} mg$.

(2) *f reduces to h w.r.t. G* if there exist $g \in G$, a term t in the power series expansion of f, $m \in M$ such that $\mathrm{lm}(t) = m\,\mathrm{lm}(g)$ and $h = f - \frac{\mathrm{lc}(t)}{\mathrm{lc}(g)} mg$.

We also use the more implicit notations *f is (top–)reducible (w.r.t. G)* in the respective cases above.

Lemma 1.7.16. *Let $0 \neq f \in \mathcal{M}$, let G be a sequence in $\mathcal{M}$.*

(1) If f is not reducible w.r.t. G, then f is in normal form w.r.t. G, i.e. $f = \eta(f, G)$.

(2) *If f has a standard representation w.r.t. G, then f is top–reducible w.r.t. G.*

(3) *f has a standard representation w.r.t. G if and only if $\eta(f, G) = 0$.*

Proof. Clear. □

This gives us a neat characterization of a standard basis. We see in the next section that the normal form algorithms form the main part of a standard basis computation.

Theorem 1.7.17. *Let $G = \{g_1, \ldots, g_r\}$ be a finite subset in $\mathcal{M}$. G is a standard basis if and only if each $0 \neq f \in M = \langle G \rangle$ has a standard representation w.r.t. G.*

Proof. If G is a standard basis for M, then for every $0 \neq f \in M$ it holds that $\eta(f, G) = 0$. If $0 \neq f \in M$ has a standard representation w.r.t. G, then there exists $g \in G$ such that $\operatorname{lm}(g) \mid \operatorname{lm}(f)$. Thus G is a standard basis for M. □

The main problem of the characterization in Theorem 1.7.17 is that it does not outline any idea of how to compute a standard basis using a termination–ensured algorithm.

1.8 The basic standard basis algorithm

Until now we have defined what a standard basis G for a submodule M in $\mathcal{M}$ is and we have already found a nice characterization of standard bases in Theorem 1.7.17, requiring any element $0 \neq f \in M$ to have a standard representation w.r.t. G. The problem is that there are infinitely many elements in M, thus we are still missing an algorithmic way to compute a standard basis given any submodule M.

In this section we introduce the notion of *s–vectors*. Those are a special linear combination of two module elements, which enable us to give an algorithmic characterization of standard bases.

Note that we are only interested in *how to compute* a standard basis in this chapter. The question of computing them *efficiently* using various kinds of optimizations is postponed to the following chapters.

Definition 1.8.1. Let $f, g \in \mathcal{M}\backslash\{0\}$ such that $\operatorname{lm}(f) = \boldsymbol{x}^\alpha e_i$ and $\operatorname{lm}(g) = \boldsymbol{x}^\beta e_j$. Let $G = \{g_1, \ldots, g_r\}$ be a finite subset of M.

(1) We define the *least common multiple of f and g* by

$$\operatorname{lcm}\left(\operatorname{lm}(f), \operatorname{lm}(g)\right) := \boldsymbol{x}^\gamma,\ \gamma = \left(\max\{\alpha_1, \beta_1\}, \ldots, \max\{\alpha_n, \beta_n\}\right) \in \mathbb{N}^n.$$

Moreover, we introduce a shorthand notation $\tau(f, g) := \operatorname{lcm}\left(\operatorname{lm}(f), \operatorname{lm}(g)\right)$.

(2) Analogously, we define the *greatest common divisor of f and g* by

$$\operatorname{gcd}\left(\operatorname{lm}(f), \operatorname{lm}(g)\right) := \boldsymbol{x}^\gamma,\ \gamma = \left(\min\{\alpha_1, \beta_1\}, \ldots, \min\{\alpha_n, \beta_n\}\right) \in \mathbb{N}^n.$$

(3) We define the *s–vector of f and g* by

$$\mathcal{S}(f,g) := \begin{cases} \operatorname{lc}(g)\frac{\tau(f,g)}{\boldsymbol{x}^\alpha} f - \operatorname{lc}(f)\frac{\tau(f,g)}{\boldsymbol{x}^\beta} g & \text{if } i = j, \\ 0 & \text{else.} \end{cases}$$

If $f, g \in \mathcal{P}$ are polynomials we also call $\mathcal{S}(f,g)$ the *s–polynomial* of f and g

(4) We say that $\mathcal{S}(f,g)$ has a *standard representation w.r.t. G* if

$$\mathcal{S}(f,g) = \sum_{i=1}^{r} p_i g_i, \; p_i \in \mathcal{P},$$

such that

$$\max\{\operatorname{lm}(p_i g_i) \mid 1 \le i \le r\} < \tau(f,g).$$

Remark 1.8.2.

(1) For $f, g \in \mathcal{P}$ think of $\mathcal{P} \cong \mathcal{P}^1$, i.e. two polynomials can be assumed to always have $i = j$ in the above definition of an s–vector.

(2) The definition of a standard representation of $\mathcal{S}(f,g)$ is strongly connected to those given in Definition 1.7.12. To see this, note that $\operatorname{lm}\big(\mathcal{S}(f,g)\big) < \tau(f,g)$.

Theorem 1.8.3 (Buchberger's Criterion)**.** *Let $G = \{g_1, \ldots, g_r\}$ be a subset in $\mathcal{M}$. Then the following are equivalent:*

(1) G is a standard basis.

(2) For all $i, j \in \{1, \ldots, r\}$, $\mathcal{S}(g_i, g_j)$ has a standard representation w.r.t. G.

Intuitive idea of proof sketched.

(1) ⇒ (2) By Theorem 1.7.17 every $0 \neq f \in M$ has a standard representation w.r.t. G. For any two elements $g_i, g_j \in G$ it holds that $\mathcal{S}(g_i, g_j) \in M$, thus clearly (2) holds.

(2) ⇒ (1) Remember that if $\mathcal{S}(f,g)$ has a standard representation w.r.t. G. This is equivalent to $\eta\big(\mathcal{S}(f,g), G\big) = 0$ by Lemma 1.7.16. Any element $g \in \langle G\rangle$ can be written as

$$\begin{aligned} g &= \sum_{i}^{r} p_i g_i, \; p_i \in \mathcal{P} \\ &= \sum_{i}^{r} \Big(\sum_{k}^{\text{finite}} a_k m_k \Big) g_i, \; a_k \in \mathcal{K}, m_k \in \operatorname{Mon}(x_1, \ldots, x_n). \end{aligned}$$

For an intuitive understanding let us assume the special situation that

$$g = a_1 m_1 g_1 - a_2 m_2 g_2.$$

If we want to compute the normal form of g two situations can arise:

(1) $a_1 m_1 \operatorname{lm}(g_1) \neq a_2 m_2 \operatorname{lm}(g_2)$, say $a_1 m_1 \operatorname{lm}(g_1) > a_2 m_2 \operatorname{lm}(g_2)$. Then we can reduce g to zero by

$$\eta(g, G) = \underbrace{a_1 m_1 g_1 - a_2 m_2 g_2}_{g} - \underbrace{a_1 m_1 g_1}_{\text{1st reducer}} + \underbrace{a_2 m_2 g_2}_{\text{2nd reducer}} = 0.$$

(2) $a_1 m_1 \operatorname{lm}(g_1) = a_2 m_2 \operatorname{lm}(g_2)$. From Definition 1.8.1 it follows that

$$\tau(g_1, g_2) \mid m_1 \operatorname{lm}(g_1).$$

Thus there exist $m' \in \operatorname{Mon}(x_1, \dots, x_n)$ and $a' \in \mathcal{K}$ such that

$$a' m' \tau(g_1, g_2) = a_1 m_1 \operatorname{lm}(g_1).$$

This enables us to rewrite $a_1 m_1 g_1 - a_2 m_2 g_2$:

$$a_1 m_1 g_1 - a_2 m_2 g_2 = a' m' \mathcal{S}(g_1, g_2).$$

By our assumption $\eta\left(\mathcal{S}(g_i, g_j), G\right) = 0$, thus $\eta(g, G) = 0$.

□

Remark 1.8.4.

(1) Note that restricting the number of s–vectors in Theorem 1.8.3 (2) by assuming that $i > j$ is no problem:
   a) $\mathcal{S}(g_i, g_i) = 0$ for all $i \in \{1, \dots, r\}$.
   b) $\mathcal{S}(g_i, g_j) = -\mathcal{S}(g_i, g_j)$ for all $i, j \in \{1, \dots, r\}$.

(2) In some textbooks the combination of Theorem 1.7.17 and Theorem 1.8.3 is called Buchberger's Criterion. We divide this into two parts, as Theorem 1.8.3 is the main part we are interested in from the computational point of view in this section.

This enables us to compute standard bases in finitely many steps.

Remark 1.8.5. If $<$ is a well–order Std is also known as *Buchberger's Algorithm*. This special case of the above presented algorithm was published first in Bruno Buchberger's PhD thesis ([34]).

Two notions appearing the first time in Algorithm 4 are important for our further investigations.

Definition 1.8.6. The set P defined in Line 2 of Algorithm 4 is called *pair set*. The tuples $(f, g) \in P$ are called *critical pairs*. The *degree of the critical pair* (f, g) is defined to be $\deg(\tau(f, g))$.

Let us proof why Std is an algorithm computing a standard basis.

Theorem 1.8.7. *Let $F \subset \mathcal{M}$ be the input of Std. Then Std is an algorithm computing a standard basis G of $\langle F \rangle$ w.r.t. $<$.*

Algorithm 4 Standard basis computation w.r.t. $<$ (Std)

Input: $F = \{f_1, \ldots, f_r\}$ a subset of $\mathcal{M}$, NF a normal form
Output: G a standard basis for $\langle F \rangle$ w.r.t. $<$

1: $G \leftarrow F$
2: $P \leftarrow \{(f_i, f_j) \mid i > j\}$
3: **while** $(P \neq \varnothing)$ **do**
4: Choose (f, g) from P.
5: $P \leftarrow P \backslash \{(f, g)\}$
6: $h \leftarrow \mathcal{S}(f, g)$
7: $h \leftarrow \mathrm{NF}(h, G)$
8: **if** $(h \neq 0)$ **then**
9: $P \leftarrow P \cup \{(g, h) \mid g \in G\}$
10: $G \leftarrow G \cup \{h\}$
11: **return** G

Correctness and Termination of Algorithm 4.

(1) Correctness follows from Theorem 1.8.3 as well as the fact that all normal form algorithms (GNF, $\mathrm{GNF}_{\mathrm{red}}$, and LNF) compute correct normal forms.

(2) If $h \neq 0$ in Line 8, then by Property (2) of Definition 1.7.12. $\mathrm{lm}(h) \notin L(G)$. Thus whenever such an element h is added to G, then $L(G)$ strictly increases, i.e. $L(G) \subsetneq L\big(G \cup \{h\}\big)$. As M is Noetherian we know by Lemma 1.1.15 (3) that this chain of increasing modules has to become stationary. This means that at some point no new elements h are added to G, and thus no new critical pairs are added to P. So $P = \varnothing$ and Std terminates after finitely many steps.

□

Remark 1.8.8. Note that the normal form used in Line 7 in Algorithm 4 is not explicitly given: Depending on the order and the variant of standard basis we want to receive this choice varies:

(1) If $<$ is a global order and there are no further requirements on the basis, we use GNF (Algorithm 1).

(2) If $<$ is global and we want to get a reduced standard basis, we use Algorithm 2. Be careful, in this situation the return value G of Std is **not** a reduced standard basis. For this we have to delete every element $g \in G$ such that there exists $g' \in G$ with $\mathrm{lm}(g') \mid \mathrm{lm}(g)$ at the end, and we have to normalize the remaining elements.

(3) If $<$ is non–global, we use the normal form described in Algorithm 3 to prevent a loop of infinite reduction steps.

At this point, we finish the main part of our introductory chapter with an example of a standard basis computation for an ideal w.r.t. a global order. We describe every step in detail, although many computations turn out to be "useless". Exactly these steps are the main issue with Std and need to be avoided as much as possible.

Example 1.8.9. Let $F = \{p_1, p_2, p_3\}$ be a finite set of polynomials in $\mathcal{K}[x, y, z]$,

$$\begin{aligned} p_1 &= xy - 1, \\ p_2 &= x^2 - 1, \\ p_3 &= y^2 - xz. \end{aligned}$$

Let $<_{\mathrm{dp}}$ be the graded reverse lexicographical order on $\mathcal{K}[x, y, z]$. For the choice in Line 4 we use the rule *first in, first out*. In this example we use the reduced normal form, thus we stick to Algorithm 2. We set $G := \{p_1, p_2, p_3\}$. Initially P is set in the following way:

$$P := \{(p_2, p_1), (p_3, p_1), (p_3, p_2)\}.$$

The computations start with $\mathcal{S}(p_2, p_1)$:

$$\begin{aligned} P &:= P \setminus \{(p_2, p_1)\} \\ \mathcal{S}(p_2, p_1) &= yp_2 - xp_1 = x^2y - y - x^2y + x \\ &= x - y. \end{aligned}$$

Clearly, $h := \eta(x - y) = x - y$, thus we need to add h to G:

$$\begin{aligned} p_4 &:= x - y \\ P &:= P \cup \{(p_4, p_1), (p_4, p_2), (p_4, p_3)\} \\ G &:= G \cup \{p_4\} \end{aligned}$$

Next we go on with $\mathcal{S}(p_3, p_1)$:

$$\begin{aligned} P &:= P \setminus \{(p_3, p_1)\} \\ \mathcal{S}(p_3, p_1) &= xp_3 - yp_1 = xy^2 - x^2z - xy^2 + y \\ &= -x^2z + y, \\ \eta(-x^2z + y) &= -x^2z + y + \underbrace{x^2z - z}_{zp_2} \\ &= y - z. \end{aligned}$$

Thus the element $p_5 := y - z$ has to be added for further computations:

$$\begin{aligned} P &:= P \cup \{(p_5, p_1), (p_5, p_2), (p_5, p_3), (p_5, p_4)\} \\ G &:= G \cup \{p_5\} \end{aligned}$$

Next pair to be computed:

$$\begin{aligned} P &:= P \setminus \{(p_3, p_2)\} \\ \mathcal{S}(p_3, p_2) &= x^2p_3 - y^2p_2 = x^2y^2 - x^3z - x^2y^2 + y^2 \\ &= -x^3z + y^2, \\ \eta(-x^3z + y^2) &= -x^3z + y^2 + \underbrace{x^3z - xz}_{xzp_2} \underbrace{- y^2 + xz}_{p_3} \\ &= 0. \end{aligned}$$

So nothing new has to be added and we can go on with the next pair in P:

$$\begin{aligned}
P :&= P \backslash \{(p_4, p_1)\} \\
\mathcal{S}(p_4, p_1) &= y p_4 - p_1 = xy - y^2 - xy + 1 \\
&= -y^2 + 1, \\
\eta(-y^2+1) &= -y^2 + 1 + \underbrace{y^2 - yz}_{y p_5} + \underbrace{yz - z^2}_{z p_5} \\
&= -z^2 + 1.
\end{aligned}$$

Before adding this element to G, multiply it by -1 and add new pairs with this element to P:

$$\begin{aligned}
p_6 &:= z^2 - 1 \\
P &:= P \cup \{(p_6, p_1), (p_6, p_2), (p_6, p_3), (p_6, p_4), (p_6, p_5)\} \\
G &:= G \cup \{p_6\}
\end{aligned}$$

Next pair to be computed:

$$\begin{aligned}
P :&= P \backslash \{(p_4, p_2)\} \\
\mathcal{S}(p_4, p_2) &= x p_4 - p_2 = x^2 - xy - x^2 + 1 \\
&= -xy + 1, \\
\eta(-xy+1) &= -xy + 1 + \underbrace{xy - 1}_{p_1} \\
&= 0.
\end{aligned}$$

It turns out that for each pair remaining in P the normal form of the corresponding s–polynomial is zero. Thus we get a Gröbner basis

$$G = \{p_1, p_2, p_3, p_4, p_5, p_6\}.$$

To receive a reduced Gröbner basis we have to remove some of the elements. Easily one sees that the uniquely defined reduced Gröbner basis of I is given by

$$G' = \{p_4, p_5, p_6\}.$$

Remark 1.8.10. Note that all 15 s–polynomials mentioned in Example 1.8.9 are generated in Algorithm 4 and most of the normal form computations consist of various reduction steps. This is very time consuming: Together with the two reductions to zero we have explicitly done above, we have 12 s–polynomials altogether, which are computed and reduced, but which do not give any new information for the standard basis we are searching for.
So let us have a more detailed look at the overhead Std has computed in this example:

(1) 15 s–polynomials are generated and their normal forms are computed.

(2) 3 of them add new polynomials, which need to be added to receive a standard basis for $\langle F \rangle$ in the end.

(3) 12 of them, i.e. $\frac{4}{5}$th of the investigated data is just useless.

In Chapter 2 we give possible optimizations to compute as much as possible useful data only.

1.9 On the complexity of standard basis computations

As a last step in our introduction to standard bases let us give a small insight to the area of the *complexity of standard basis computations*. For this, we restrict ourselves to the polynomial case.

For algorithms one mostly measures the complexity in two different types: *time complexity* and *space complexity*. Let us introduce the so–called *Landau–notation*:

Definition 1.9.1. Let $g : \mathbb{N} \to \mathbb{R}$ be a function on the natural numbers. An algorithm A has a complexity of $\mathcal{O}\big(g(n)\big)$ if and only if a Turing machine can compute the result of an input of A in $c \cdot g(n)$ steps for constant $c, n \in \mathbb{N}$.

Example 1.9.2. Assume that the time complexity $T(n)$ of an algorithm is given by $T(n) = 687n^5 + 123n^4 + 12$. Then we write $T(n) = \mathcal{O}(n^5)$.

In complexity theory one defines so–called *complexity classes* to group algorithms. For us, two of them are important:

(1) A problem is known to be P if the corresponding algorithm solving it, is $\mathcal{O}\big(g(n)\big)$, where g is a polynomial in n.

(2) A problem is known to be Expspace if the corresponding algorithm solving it, is $\mathcal{O}\big(2^{g(n)}\big)$, where g is a polynomial in n.

Let us try to parametrize the computation of standard bases based on the corresponding input. There exists a quite natural setting of parameters to define a finite set $F = \{f_1, \ldots, f_r\}$ of polynomials in $\mathcal{P}$:

(1) the number n of variables in $\mathcal{P}$,

(2) the number $r = \#(F)$ of elements in F,

(3) the maximal degree $d_{\max} := \max\{\deg(f_i) \mid f_i \in F\}$, and

(4) the maximal coefficient $c_{\max} := \max\{$ all coefficients of $f_i \mid f_i \in F\}$.

Having this as input data, the complexity of computing the standard basis G relies on them:

(1) The maximal degree during the computation is bounded by a function in n, r and $d_{\max}$.

(2) Also $\#(G)$ is bounded by a function in n, r and $d_{\max}$.

(3) The maximal coefficient appearing during the computation is bounded by a function in n, r, $d_{\max}$ and $c_{\max}$.

In some special situation these upper bounds can be given, e.g. [17, 86–88, 123, 125]. We do not want to discuss this in detail and just give a feeling for "how hard" this problem really is in general.

In [41, 42] it is shown by Caniglia, Galligo and Heintz that the complexity of computing a Gröbner basis w.r.t. the graded reverse lexicographical order $<_{\mathrm{dp}}$ for the input set $F = \{f_1, \ldots, f_r\}$ is

(1) $\mathcal{O}\big(d_{\max}^{n^2}\big)$ if $\#\big(\{a \in \mathcal{K}^n \mid f_i(a) = 0 \text{ for all } f_i \in F\}\big) < \infty$, and

(2) $\mathcal{O}\big(d_{\max}^{n}\big)$ if the solutions at infinity are also finite.

On the other hand, the same computations w.r.t. the lexicographical order $<_{\mathrm{lp}}$ lead to computations with a complexity of $d_{\max}^{\mathcal{O}(n^3)}$.

In 1990 Doubé has presented an upper bound for the degree of elements in the reduced standard basis. As already discussed above, it strongly depends on the input data.

Theorem 1.9.3 (Dubé)**.** *Let $I = \langle f_1, \ldots, f_r \rangle \subseteq \mathcal{P}$ be an ideal, $<$ any monomial order on $\mathcal{P}$. The degree of polynomials in the reduced standard basis for I has the upper bound*

$$D := 2\left(\frac{d_{\max}^2}{2} + d_{\max}\right)^{2^{n-1}}.$$

Proof. See [53]. □

This means that we have a doubly–exponential bound on the degrees of the elements in the standard basis. In [124] the following is shown:

Theorem 1.9.4 (Mayr)**.** *Given an ideal I in a polynomial ring of n variables, generated by finitely many polynomials, the reduced standard basis G for I w.r.t. a monomial order $<$ can be computed in* EXPSPACE.

All in all we have to state the following:

Remark 1.9.5. The complexity of standard basis computations can be *doubly–exponential in the number of solutions of the polynomial system.*

Thus the problem in focus of this thesis can be characterized as being "not so easy".

With this premises in mind, it makes sense to think about how to improve the computations of standard bases. This is the content of the following chapters.

2 Ways to improve standard basis computations

In Chapter 1 we have introduced the topic of standard basis computations. Closing Section 1.8 with an example of a Gröbner basis computation for an ideal we have seen that, also the standard algorithm is quite easy to understand, it has one huge drawback: *redundant computations*. Most of the computations in Example 1.8.9 generated new critical pairs not needed for the Gröbner basis. Their corresponding s–vectors are computed and then the reduction process starts. In the end, these redundant reductions just tell us that the s–vector we investigate already fulfills the Buchberger Criterion (Theorem 1.8.3) and we do not need to add something new to the intermediate Gröbner basis. This is not what we want to have. We would like to only compute *new data needed* for the basis. Thus one needs to think about ways to distinguish the useful and the useless data.

In this chapter we present ideas how to improve Std. This can be done not only by adding some criteria for the critical pairs, but also by different implementations of Std. The main differences can be found in the decisions one has to take during the computations. Other ideas related to speed–up the computational time of standard basis algorithms are given, too.

Besides considering criteria to reject useless critical pairs there have been other ideas developed which try to improve the following important steps of STD:

(1) In Line 4 of Algorithm 4 we have to choose the next critical pair STD shall investigate. In the case of homogeneous elements as input one can always sort the pair set P by the degree of the critical pairs, whereas in the inhomogeneous case the *sugar degree* gives a good choice. It can be understand as "the degree the corresponding element would have if we start with homogeneous elements as input" and was presented first in [85]. Thus we get some strategies *how to order and how to choose critical pairs from* P.

(2) In Section 2.3 we present the two main criteria to pick out useless critical pairs in advance, which go back to Buchberger, [34, 35, 109]. Different attempts using these criteria where developed. Subsequently we present the one being most influentual, the *Gebauer–Möller implementation* ([81]) in Section 2.4.

(3) Another important point of STD is the normal form computation of the s–vectors. In [114, 117] Lazard presented a new way describing the reduction steps with the Sylvester matrix. This method was improved in [61] by Faugère in 1999.

(4) Also the second possible choice in STD is investigated: If we have several possible reducers, which one should be chosen? Some recent work is done in [61], but a much more comprehensive discussion can be found in [31]: The main idea is to store the intermediate reduced elements if they have some nice properties which can be useful for upcoming reductions.

(5) One can use the Hilbert(–Poincaré) series to speed up the standard basis computation, as shown in [154]: Compute the standard basis G_1 w.r.t. a "nice" order $<_1$ and get the Hilbert series. Then one can use the Hilbert series to get bounds for the number of elements in intermediate versions of G_2, the wanted standard basis w.r.t. $<_2$. This idea is discussed in Section 2.7.

(6) Other ideas are related to the monomial order the computations are based on: If you want to compute a standard basis G_1 in a given order $<_1$, but this is too hard a problem, try to compute the standard basis G_2 of your input w.r.t. some related order $<_2$. Afterwards, compute G_1 using G_2. There are several attempts of doing this ([4, 38, 44, 67, 152, 159]). We discuss the most efficient ones in Section 2.8

(7) A standard basis computation over the rationals as ground field $\mathcal{K}$ could be reduced to several computations of the same input over ground fields of characteristic $p < \infty$, p a prime number. Those ideas have been evolved a lot over the last years, due to the fact that those *modular computations* benefit from multi–core processors, which are designed to compute several independent workloads in parallel ([6, 103, 153]). We give a deeper insight in this topic in Section 2.9.

(8) A complete different approach to compute standard bases is done using *involutive* methods ([11, 24, 25]). Using a slightly different definition of the term "division"

instead of the normal one leads to a new normal form computation. Based on this so–called *involutive bases* can be computed, which are related to standard bases.

The intention of this chapter is to make the reader aware of how to improve standard basis algorithms in the classical sense, i.e. without using signatures. The knowledge and understanding of these ideas is essential to grasp the benefits and problems of the signature–based approach.

2.1 The problem of zero reductions

The main drawback we see for the standard basis algorithm Std as presented in Section 1.8 is the vast number of reductions to zero. These reductions are in some sense uselss for the standard basis computation.

Firstly, let us define what the term "useless" means in our setting.

Definition 2.1.1. Let $f, g \in \mathcal{M}$. A critical pair (f, g) in Std is called *useless* if and only if $\eta\left(\mathcal{S}(f, g)\right) = 0$. If a pair is not useless we call it *useful*.

The concept behind this notation is the following: We want to compute a standard basis, i.e. we need to get a set $G = \{g_1, \ldots, g_s\}$ such that $\mathcal{S}(g_i, g_j) = 0$ for all $i, j \in \{1, \ldots, s\}$ with $i > j$:

Std starts with a set of elements, say $F = \{f_1, \ldots, f_r\}$. Some of the s–vectors $\mathcal{S}(f_i, f_j)$ might reduce to zero w.r.t. F, others may not. On the one hand, those, which do not reduce to zero, are important for us, for example, assume that $\mathcal{S}(f_i, f_j)$ reduces to an element $f_{i,j} \neq 0$ w.r.t. F. We need to ensure that it reduces to zero, if we want to receive a standard basis by Theorem 1.8.3. Thus the only possibility to achieve this situation is to enlarge F, $F' = F \cup \{f_{i,j}\}$. By construction it is clear that $\mathcal{S}(f_i, f_j)$ reduces to zero w.r.t. F'. Iterating this process over all s–vectors, in the end, we receive a standard basis G. On the other hand, those s–vectors reducing to zero w.r.t. F are not important at all. At the end of their reduction we do neither enlarge F nor do we generate new critical pairs. Thus, nothing in our data set has changed when $\eta\left(\mathcal{S}(f_k, f_l)\right) = 0$. We could compute G without even considering the s–vector, its reduction is useless for our task.

In bigger computations more than 90 percent of the reduction steps done in Std lead to zero reductions, even in Example 1.8.9, a very small example we have done by hand, 80 percent of the computed data does not influence the computation of G and produce computational overhead. We want to avoid this; for bigger problems we even have to do so as otherwise the standard basis is not computable even on super computers respectively compute servers.

2.2 Selection strategies for critical pairs

We have seen that there are two different choices in the standard basis algorithm Std:

(1) How to choose the next critical pair (f, g) from the pair set P?

(2) How to choose the reducer in NF if there are different possible ones?

A discussion giving some answers and heuristics for the second question can be found in Section 2.6.

Here we focus our attention on the question how to choose the next critical pair from P efficiently. What we want to do in terms of the pseudo code of Std is to improve Line 4 from Algorithm 4. Instead of just picking a critical pair from P we want to select a special subset of P and sort the included elements, such that the received order, in which to compute and reduce s–vectors, is hopefully more efficient for standard basis computations. We call the algorithm to select a special subset of P Select, instead of Line 4 in Algorithm 4 we have to be a bit more explicit:

Algorithm 5 Standard basis algorithm including selection strategy (Std)

Input: $F = \{f_1, \ldots, f_r\}$ a subset of $\mathcal{M}$, NF a normal form
Output: G, a standard basis for $\langle F\rangle$ w.r.t. $<$

1: $G \leftarrow F$
2: $P \leftarrow \big\{(f_i, f_j) \mid i > j\big\}$
3: **while** $(P \neq \varnothing)$ **do**
4: $\quad P' \leftarrow \textsc{Select}(P)$
5: $\quad P \leftarrow P \backslash P'$
6: $\quad$ **while** $(P' \neq \varnothing)$ **do**
7: $\quad\quad (f, g) \leftarrow$ First element of P'
8: $\quad\quad P' \leftarrow P' \backslash \big\{(f, g)\big\}$
9: $\quad\quad h \leftarrow \mathcal{S}(f, g)$
10: $\quad\quad h \leftarrow \mathrm{NF}(h, G)$
11: $\quad\quad$ **if** $(h \neq 0)$ **then**
12: $\quad\quad\quad P \leftarrow P \cup \big\{(g, h) \mid g \in G\big\}$
13: $\quad\quad\quad G \leftarrow G \cup \{h\}$
14: **return** G

In Line 4 a special, ordered subset $P' \subseteq P$ is chosen. Next, only the elements of P' are taken into account (Line 6), whereas all elements of $P \backslash P'$ are held back. So the "magic" happens in Select. Let us give some examples of possible selection strategies for P':

Assume the following situation: We have just computed the normal form of $\mathcal{S}(f, g)$, possibly added a new element $h = \eta\left(\mathcal{S}(f, g)\right) \neq 0$ to G, generated new critical pairs with h and added them to P. What is our next choice considering P? One could just choose the oldest element from P, i.e. the element added to P before all other elements currently in P have been added. Other choices could be: the youngest element, the element of lowest lcm, the element of highest lcm, etc.

The choice of these elements is very important for the performance of STD: The critical pairs possibly become new elements in G, and thus new reducers for further normal form computations. For example, assuming $<$ to be a global order, it is not helpful to get a new element h from a normal form computation of an s–vector with $\deg(h) = 8$, if there are hundreds of critical pairs left in P, whose s–vectors have degree < 8.

(1) Let us start with a special situation: Assume for the moment that the input $F = \{f_1, \ldots, f_r\}$ of STD is homogeneous, i.e. all f_i are homogeneous. We note that the s–vectors $\mathcal{S}(f_i, f_j)$ for any two elements $f_i, f_j \in F$ are homogeneous, too, by construction. Computing the normal form, we only have homogeneous reducers, thus again, $\eta\left(\mathcal{S}(f_i, f_j)\right)$ is homogeneous. So no inhomogeneous elements are added to $G = \{g_1, \ldots, g_s\}$ during the computations of STD. Moreover, the following important equation holds:

$$\deg\left(\tau(g_k, g_l)\right) = \deg\left(\mathcal{S}(g_k, g_l)\right).$$

If $\eta\left(\mathcal{S}(g_k, g_l)\right) \neq 0$ then even the following holds:

$$\deg\left(\tau(g_k, g_l)\right) = \deg\left(\mathcal{S}(g_k, g_l)\right) = \deg\left(\eta\left(\mathcal{S}(g_k, g_l)\right)\right). \tag{2.2.1}$$

One of the most common and natural strategies to choose critical pairs is the *normal selection*: SELECT(P) takes all pairs such that the degree of their lowest common multiple is minimal, removes them from P and adds them to P'. Next P' is either ordered by increasing (resp. decreasing) lcm, or by the indices of the elements generating the pair (i.e. the point, when the pair has been added to P), or just left unordered. If the input of STD is homogeneous, with each selection the degree of elements in P' increases. This strategy was defined at first by Buchberger in [35].

(2) Assuming the normal selection if the input of STD is inhomogeneous, we see some drawbacks of this method: Equation 2.2.1 does no longer hold, but only the inequalities

$$\deg\left(\tau(g_k, g_l)\right) \geq \deg\left(\mathcal{S}(g_k, g_l)\right) \geq \deg\left(\eta\left(\mathcal{S}(g_k, g_l)\right)\right).$$

Thus the situation $d := \deg\left(\tau(g_k, g_l)\right) > \deg\left(\eta\left(\mathcal{S}(g_k, g_l)\right)\right)$ is rather possible. This implies that the lcm of pairs generated by $g' := \eta\left(\mathcal{S}\left(g_k, g_l\right)\right)$ can be lower than d. Thus selecting the next subset of P, P' can consist of elements of degree lower than d. As the drop of the degree during the reduction process of s–vectors can be rather big, we end up computing lots of elements of lower degree *after* the computation of elements of higher degree. This is not really efficient as explained in the above discussion. Thus to avoid the processing of elements of a degree d before elements of degree $< d$ are treated, in the inhomogeneous case, one just selects one element from P at a time, namely the one whose lcm fulfills the following property:

$$\tau(g_k, g_l) = \min\{\tau(g_i, g_j) \mid (g_i, g_j) \in P\}.$$

Although this increases the likelihood of taking the element of lowest possible degree, it still has a big disadvantage: Assume a lexicographic order in the inhomogeneous case. Then STD could compute two elements with the variable x_1 eliminated.

In this situation the algorithm always processes the pairs generated by these two elements until a standard basis of the ideal generated by these two elements is computed. Not until this point the other elements are taken into account. This could have a really bad impact on the runtime and the memory consumption of the algorithm.

Note that the normal strategy is still much more efficient than just choosing any element of P arbitrarily. Grouping pairs of the same degree has a great impact on the performance of the algorithm.

A commonly used idea to cope with those bad behaving inhomogeneous input is to homogenize it as explained in Section 1.5. Computing the standard basis G of the homogenized input[1] with the normal strategy is safe and efficient. In the end, one has to dehomogenize G and receives a standard basis G' of the initially inhomogeneous input. Again, there is a big downside of this approach: G can be a lot larger than G', i.e. a lot of overhead / useless data for the original problem is computed.

Another possibility to handle such inhomogeneous input is to compute a standard basis w.r.t. another order and deduce the basis one searches for from it. A short overview of corresponding methods is given in Section 2.8.

A solution of this problem using just a different selection strategy is explained next.

(3) The so–called *sugar selection* was presented 1991 in [85]. The main idea is to equip each critial pair with another degree, the so–called *sugar degree* which is the degree the pair would have if we would have homogenized the input in the beginning. The crucial point is now to order the pair set P in 3 different steps by the following properties (in the given order):

   a) sugar degree,
   b) usual degree (w.r.t. the given order),
   c) indices of the generators of the critical pair.

   This enables us to sort the pairs as they would be sorted in the case of a homogenization, but without the drawbacks of the overhead a real homogenization would raise.

Let us define the sugar degree of a critical pair explicitly as it is also important for signature–based algorithms computing with inhomogeneous data. We show the close affiliation between the sugar degree and the signature of an element in Section 7.1.

Definition 2.2.1. Let the finite subset $F = \{f_1, \ldots, f_r\}$ of elements f_i in $\mathcal{M}$ be the input for Std, let $t \in \mathcal{P}$ be a term.

(1) The *sugar degree of an initial* f_i is defined by

$$\text{s-deg}(f_i) := \deg(f_i).$$

(2) For any element $g \in G$ generated during the computations of Std and any term $t \in \mathcal{M}$ we define

$$\text{s-deg}(tg) := \deg(t) + \text{s-deg}(g).$$

[1]Note, only the generators of F are homogenized, we do not homogenize $I = \langle F \rangle$.

(3) Moreover, for any two elements $g, h \in G$ we define

$$\text{s-deg}(g+h) := \max\{\text{s-deg}(g), \text{s-deg}(h)\}.$$

The above definition ensures that the sugar degree is the corresponding degree of the computed elements, if we homogenize the input before STD starts its computations.

To end this section, let us state one last, nice fact about homogeneous standard basis computations:

Definition 2.2.2. A finite set $G = \{g_1, \ldots, g_s\}$ in $\mathcal{M}$ of homogeneous elements g_i is called a *d–standard basis* if for all $i, j \in \{1, \ldots, s\}$ with $\deg\big(\tau(g_i, g_j)\big) \le d$ the corresponding s–vectors $\mathcal{S}(g_i, g_j)$ have a standard representation w.r.t. G.

Proposition 2.2.3. *Let F be a finite set of homogeneous elements in $\mathcal{M}$, equip STD with the normal selection strategy. Denote the intermediate standard basis by G, let $P' \subset P$ be the subset of all pairs of degree d during the computations of STD. At the moment all pairs of P' are treated, i.e. $P' = \emptyset$, G is a d–standard basis for F.*

Proof. By construction, all s–vectors $\mathcal{S}(g_i, g_j)$ with $\deg\big(\tau(g_i, g_j)\big) \le d$ have standard representation w.r.t. G. □

We postpone the discussion of selecting a "good" reducer to Section 2.6 as this problem is strongly related to the topics of sections 2.3 – 2.5.

2.3 Buchberger's criteria

Next we discuss the most obvious improvement of STD one can think of: Try to compute the normal form of as few as possible s–vectors. The problem is that if we take not enough of them into account, or the wrong ones, we do not receive a standard basis at the end of STD's computations.

Note that we give proofs (or at least sketches of them) for the two criteria stated, although they can be found in any introductory textbook about computer algebra. The reason for this is again that the reader should be able to compare the classical criteria to the signature–based ones. It turns out that proving the correctness of signature–based standard basis algorithms is much harder than for their classical counterparts. This is due to the fact that all attempts presented in this chapter are only based on investigating the critical pairs resp. s–vectors themselves, whereas one has to take care of much more structure in the signature–based situation.

For an easier notation we restrict ourselves to considering only polynomials in $\mathcal{P}$ w.r.t. a well–order in this section.

The easiest criterion is *Buchberger's 1st Criterion*[2] stated by Buchberger in [34]. It depends only on the two elements f and g generating the s–vector $\mathcal{S}(f, g)$, but not on any

[2] Buchberger's 1st Criterion is also known as *Product Criterion*.

other element in P. The crucial point is that if $\operatorname{lm}(f)$ and $\operatorname{lm}(g)$ have nothing in common, then the normal form of the corresponding s–vector is zero.

Lemma 2.3.1 (Buchberger's 1st Criterion)**.** *Let $f, g \in \mathcal{P}$ be two elements such that $\tau(f,g) = \operatorname{lm}(f)\operatorname{lm}(g)$. Then $\mathcal{S}(f,g)$ has a standard representation w.r.t. $\{f,g\}$.*

Proof. If $\tau(f,g) = \operatorname{lm}(f)\operatorname{lm}(g)$, then we get

$$\mathcal{S}(f,g) = \operatorname{lc}(g)\operatorname{lm}(g)f - \operatorname{lc}(f)\operatorname{lm}(f)g.$$

As the leading terms of the two summands cancel each other by construction we get

$$\mathcal{S}(f,g) = \operatorname{lc}(g)\operatorname{lm}(g)\operatorname{tail}(f) - \operatorname{lc}(f)\operatorname{lm}(f)\operatorname{tail}(g).$$

Now we can compute the normal form of $\mathcal{S}(f,g)$ in two steps:

(1) For all terms s_i in $\operatorname{tail}(f)$ we subtract $s_i g$ from $\mathcal{S}(f,g)$. In the end we get a first intermediate normal form η' w.r.t. $\{g\}$:

$$\begin{aligned}\eta'\left(\mathcal{S}(f,g)\right) &= \underbrace{\operatorname{lc}(g)\operatorname{lm}(g)\operatorname{tail}(f) - \operatorname{tail}(f)g}_{\operatorname{tail}(f)\operatorname{tail}(g)} - \operatorname{lc}(f)\operatorname{lm}(f)\operatorname{tail}(g)\\ &= \operatorname{tail}(g)f.\end{aligned}$$

(2) Clearly, we reduce this to zero by subtracting $t_i f$ for all terms t_i in $\operatorname{tail}(g)$.

All in all we have $\eta\left(\mathcal{S}(f,g)\right) = 0$. □

Using Buchberger's 1st Criterion in Example 1.8.9 would delete the following critical pairs:

$$(p_3,p_2),(p_4,p_3),(p_5,p_2),(p_5,p_4),(p_6,p_1),\ldots,(p_6,p_5).$$

Thus 9 of the 12 s–polynomials which lead to zero are not computed if we use Buchberger's 1st Criterion in Std when building new critical pairs, i.e. in lines 2 and 9. Checking the greatest common divisor resp. least common multiple of $\operatorname{lm}(f)$ and $\operatorname{lm}(g)$ can be done in much less computational steps than any reduction step in the example. Thus this is a big improvement of Std.

Needless to say, there are still 3 critical pairs left in our example, which reduce to zero. In practice, Buchberger's 1st Criterion is an improvement of Std, but it does not find nearly all useless critical pairs.

Their detection can be optimized by

(1) improving Buchberger's 1st Criterion and

(2) adding another criterion to Std.

For Case (1) we can define an easy extension of Lemma 2.3.1:

Corollary 2.3.2 (Extended version of Buchberger's 1st Criterion)**.** *Let $f, g \in \mathcal{P}$, m a monomial in $\operatorname{Mon}(x_1,\ldots,x_n)$ such that for all $t \in \operatorname{supp}(f) \cup \operatorname{supp}(g)$ it holds that $m \mid t$. If $\tau\left(\frac{f}{m},\frac{g}{m}\right) = \frac{\operatorname{lm}(f)}{m}\frac{\operatorname{lm}(g)}{m}$, then $\mathcal{S}(f,g)$ has a standard representation w.r.t. $\{f,g\}$.*

Sadly, in Example 1.8.9 this extended version does not detect any other useless critical pair besides the ones also detected by the one presented in Lemma 2.3.1.

Remark 2.3.3. Note that the extended version of Buchberger's 1st Criterion does not need much additional computations compared to the usual one: For any new element f we need to compute the greatest common divisor of all terms $t \in \operatorname{supp}(f)$ once. Even assuming dense[3] elements the time needed to compute the gcd is very small compared to a reduction step when computing, for example, a normal form.

For Case (2) we use the idea developed in [34,109]: Buchberger stated a second criterion, and most of the classical standard basis algorithms are based on it.

Lemma 2.3.4 (Buchberger's 2nd Criterion[4]). *Let $f, g, h \in \mathcal{P}$, F a finite subset of $\mathcal{P}$. Assume that*

(1) $\operatorname{lm}(g) \mid \tau(f,h)$, *and*

(2) $\mathcal{S}(f,g)$ *and* $\mathcal{S}(g,h)$ *have a standard representation w.r.t. F. Then* $\mathcal{S}(f,h)$ *has a standard representation w.r.t. F.*

Proof. W.l.o.g. we can assume that $\operatorname{lc}(f) = \operatorname{lc}(g) = \operatorname{lc}(h) = 1$. As $\operatorname{lm}(g) \mid \tau(f,h)$ there exist monomials m_f, m_h in $\mathcal{P}$ such that

$$m_f \tau(f,g) = \tau(f,h) = m_h \tau(g,h).$$

This gives us a rewriting for $\mathcal{S}(f,h)$:

$$\begin{aligned}
\mathcal{S}(f,h) &= \frac{\tau(f,h)}{\operatorname{lm}(f)} f - \frac{\tau(f,h)}{\operatorname{lm}(h)} h \\
&= m_f \frac{\tau(f,g)}{\operatorname{lm}(f)} f \underbrace{- m_f \frac{\tau(f,g)}{\operatorname{lm}(g)} g + m_g \frac{\tau(g,h)}{\operatorname{lm}(g)} g}_{=0} - m_g \frac{\tau(g,h)}{\operatorname{lm}(h)} h \\
&= m_f \mathcal{S}(f,g) - m_g \mathcal{S}(g,h).
\end{aligned}$$

By assumption $\mathcal{S}(f,g)$ and $\mathcal{S}(g,h)$ have a standard representation w.r.t. F, thus $\mathcal{S}(f,h)$ has a standard representation w.r.t. F, too. $\square$

Reconsidering Example 1.8.9 again we could use Buchberger's 2nd Criterion to see that (p_3, p_2) need not be computed: $\operatorname{lm}(p_2) \mid \tau(p_3, p_1)$ and the normal forms of $\mathcal{S}(p_2, p_1)$ and $\mathcal{S}(p_3, p_1)$ are already computed. Thus by Lemma 2.3.4 we can securely remove (p_3, p_2) from P.

Remark 2.3.5. Note that the efficiency (and also the correctness) of Buchberger's 2nd Criterion depends highly on the order in which the critical pairs are checked. Thus its implementation is not as easy as the one of Buchberger's 1st Criterion. A very good implementation of both criteria is given by Gebauer and Möller in [81]. This is discussed in more detail in Section 2.4

[3] We have not defined this explicitly until now, just think of elements consisting of lots of terms.

[4] Buchberger's 2nd Criterion is also known as *Chain Criterion*.

Let us define some more notations.

Convention.

(1) In the situation of Lemma 2.3.1 we say that (f, g) *is detected by Buchberger's 1st Criterion*.

(2) Similarly, in the situation of Lemma 2.3.4 we say that (f, h) *is detected by Buchberger's 2nd Criterion*.

Using Corollary 2.3.2 and Lemma 2.3.4 we can conclude an improved version of Theorem 1.8.3:

Corollary 2.3.6. *Let $G = \{g_1, \ldots, g_r\}$ be a subset in $\mathcal{P}$. Then the following are equivalent:*

(1) G is a standard basis.

(2) For all $i > j \in \{1, \ldots, r\}$ one of the following hold:

a) $\mathcal{S}(g_i, g_j)$ has a standard representation w.r.t. G.

b) $\mathcal{S}(g_i, g_j)$ is detected by Buchberger's 1st Criterion.

c) $\mathcal{S}(g_i, g_j)$ is detected by Buchberger's 2nd Criterion.

Also the statement of Corollary 2.3.6 follows easily from the above discussion, implementing it efficiently is a hard task. Next we discuss a highly optimized implementation of a standard basis algorithm using Buchberger's criteria.

2.4 The Gebauer–Möller implementation

In this section we show in detail how the criteria presented in Section 2.3 can be implemented in Std. Whereas Buchberger's 1st Criterion is no problem, his 2nd Criterion is a bit harder to integrate in Std. As in the last section, we describe the polynomial situation w.r.t. a well–order only.

One possible problem is a two–out–of–three deletion: Assume $f, g, h \in \mathcal{P}$ such that $\mathrm{lm}(g) \mid \tau(f, h)$. If two of the three lowest common multiples involved are equal, e.g. $\tau(f, g) = \tau(f, h)$, then one can choose which critical pair to be removed:

(1) Remove (f, h), but compute $\mathcal{S}(f, g)$ and $\mathcal{S}(g, h)$, or

(2) remove (f, g), but compute $\mathcal{S}(f, h)$ and $\mathcal{S}(g, h)$.

Both choices are possible, the problem is not to remove both at the same time, (f, h) by g and then (g, h) by f. One way would be to always check that two of the corresponding critical pairs have been investigated, before removing the third one. This generates some overhead in the algorithm, but ensures the correctness. As we are interested in efficient standard basis algorithms, this is not an adequate solution.

In the following we state the so–called *Gebauer–Möller implementation* of a standard basis algorithm ([81]). It removes critical pairs as early as possible, doing the criteria checks in different steps. To understand its correctness we need the two following, easy statements:

Lemma 2.4.1. *Let $f, g, h \in F \subset \mathcal{P}$. Then the following are equivalent:*

(1) $\mathrm{lm}(f) \mid \tau(g, h)$.

(2) $\tau(f, g) \mid \tau(g, h)$.

(3) $\tau(f, h) \mid \tau(g, h)$.

Proof. *(1)* $\Rightarrow$ *(2)* and *(2)* $\Rightarrow$ *(3)* are clear. Assuming that $\tau(f, h) \mid \tau(g, h)$, there exist monomials $\lambda, \lambda_f \in \mathcal{P}$ such that

$$\begin{aligned} \lambda\tau(f, h) &= \tau(g, h) \\ \lambda\lambda_f \,\mathrm{lm}(f) &= \tau(g, h). \end{aligned}$$

Thus *(3)* $\Rightarrow$ *(1)*. □

Corollary 2.4.2. *Let $f, g, h \in F \subset \mathcal{P}$, λ_g, λ_h be two monomials in $\mathcal{P}$ such that $\lambda_g > 1$, $\lambda_h > 1$ and*

$$\begin{aligned} \lambda_g\tau(f, g) &= \tau(g, h), \\ \lambda_h\tau(f, h) &= \tau(g, h). \end{aligned}$$

Then $\tau(f, g) \nmid \tau(f, h)$ and $\tau(f, h) \nmid \tau(f, g)$.

Proof. Assume that $\tau(f, g) \mid \tau(f, h)$. Then by Lemma 2.4.1 $\mathrm{lm}(g) \mid \tau(f, h)$. Moreover, $\tau(g, h) \mid \tau(f, h)$. This contradicts the assumption that $\lambda_h > 1$. The second statement follows analogously. □

The Gebauer–Möller implementation, presented in the follwing, consists of two separate algorithms:

The only difference to Algorithm 5 from Section 2.2 is the usage of another algorithm, called Update, when new elements are added: In Update the criteria of Section 2.3 are used to check which pairs should enter the set of critical pairs P.

Let us have a closer look at Algorithm 7. Buchberger's criteria are checked in 4 steps:

(1) In Line 1 all critical pairs (f, g) not being generated by h are checked by Buchberger's 2nd Criterion w.r.t. h. But only those pairs are deleted where $\tau(f, g) \neq \tau(f, h)$ and $\tau(f, g) \neq \tau(g, h)$. Only in this step elements of P_{old} can be removed from the set of critical pairs. Later on only elements including h are checked and possibly deleted.

(2) In Line 5 we search in the set P' of all critical pairs including h for pairs $(f, h), (g, h)$, whose least common multiples are multiples of each other: $\tau(f, h) \mid \tau(g, h)$. In this situation we remove the pair (g, h) from P'. Note that by Corollary 2.4.2 for those $(f, h), (g, h)$ the corresponding (f, g) is not deleted in the first step.

Algorithm 6 Improved standard basis computation w.r.t. $<$ (GM)

Input: $F = \{f_1, \ldots, f_r\}$ a subset of $\mathcal{P}$, NF a normal form
Output: G, a standard basis for $\langle F \rangle$ w.r.t. $<$

1: $G \leftarrow f_1$
2: $P \leftarrow \varnothing$
3: **for** $(i = 2, \ldots, r)$ **do**
4: $\quad P \leftarrow \text{Update}(P, G, f_i)$
5: $\quad G \leftarrow G \cup \{f_i\}$
6: $l \leftarrow r$
7: **while** $(P \neq \varnothing)$ **do**
8: $\quad P' \leftarrow \text{Select}(P)$
9: $\quad P \leftarrow P \backslash P'$
10: $\quad$ **while** $(P' \neq \varnothing)$ **do**
11: $\quad\quad (f, g) \leftarrow$ First element of P'
12: $\quad\quad P' \leftarrow P' \backslash \{(f, g)\}$
13: $\quad\quad h \leftarrow \mathcal{S}(f, g)$
14: $\quad\quad h \leftarrow \text{NF}(h, G)$
15: $\quad\quad$ **if** $(h \neq 0)$ **then**
16: $\quad\quad\quad f_{l+1} \leftarrow h$
17: $\quad\quad\quad P \leftarrow \text{Update}(P, G, f_{l+1})$
18: $\quad\quad\quad G \leftarrow G \cup \{f_{l+1}\}$
19: $\quad\quad\quad l \leftarrow l + 1$
20: **return** G

(3) The third step is very similar to the second one: In Line 10 we start deleting all those (g, h) from P' where $\tau(f, h) = \tau(g, h)$. With the same argument as above, the corresponding pair (f, g) is not deleted in step 1.

(4) As the last step, we check all remaining pairs in P' by Buchberger's 1st Criterion and delete those detected.

Remark 2.4.3. Note that a crucial point of any implementation of Buchberger's 1st and 2nd Criterion is to check the 2nd Criterion first, the 1st one later. Why is this so important? Deleting one useless critical pair from the algorithm the 2nd Criterion needs three pairs, say (f, g), (f, h) and (g, h). Assume that we have already considered (f, g) and (f, h), then we can delete (g, h). Now let furthermore $\tau(f, g) = \text{lm}(f)\,\text{lm}(g)$. Then we can also remove (f, g) and only need to compute (f, h). Doing this process the other way around, we check the 1st Criterion first, which means that we delete (f, g) before we check the 2nd Criterion. Thus (f, h) and (g, h) are left and we cannot remove any of them. This should illustrate that the Gebauer–Möller implementation is highly efficient checking Buchberger's 1st Criterion as last step in (4).

This implementation is a very efficient one: It does not depend on the selection strategy of the pairs, it does not depend on the order $<$. Moreover, it checks critical pairs in the moment they are generated and does not keep them until they are selected; this saves memory

Algorithm 7 Updating the set of critical pairs (UPDATE)

Input: P_{old} a set of critical pairs, G a subset of $\mathcal{P}$, $h \in F$
Output: P_{new} a set of critical pairs

for all $(f, g) \in P_{\text{old}}$ **do**
  if $\left(\text{lm}(h) \mid \tau(f, g) \text{ and } \tau(f, h) \neq \tau(f, g) \text{ and } \tau(g, h) \neq \tau(f, g)\right)$ **then**
    $P_{\text{old}} \leftarrow P_{\text{old}} \setminus \{(f, g)\}$
$P' \leftarrow \{(f, h) \mid f \in G\}$
for all $(f, h) \in P'$ **do**
  Fix $(f, h) \in P'$.
  for all $\left((g, h) \in P' \setminus \{(f, h)\}\right)$ **do**
    if $\left(\exists \lambda > 1 \text{ s.t. } \tau(f, h) = \lambda \tau(g, h)\right)$ **then**
      $P' \leftarrow P' \setminus \{(g, h)\}$
for all $(f, h) \in P'$ **do**
  Fix $(f, h) \in P'$.
  for all $\left((g, h) \in P' \setminus \{(f, h)\}\right)$ **do**
    if $\left(\tau(f, h) = \tau(g, h)\right)$ **then**
      $P' \leftarrow P' \setminus \{(g, h)\}$
for all $(f, h) \in P'$ **do**
  if $\left(\tau(f, h) = \text{lm}(f)\,\text{lm}(h)\right)$ **then**
    $P' \leftarrow P' \setminus \{(f, h)\}$
$P_{\text{new}} \leftarrow P_{\text{old}} \cup P'$
return P_{new}

and overhead in the computational point of view. SINGULAR's standard basis algorithm is based on a highly optimized version of the Gebauer–Möller implementation together with a lot of computational tricks.

2.5 Normal form computations and their relation to Gaussian elimination

In this section we show how normal form computations are related to Gaussian eliminations. We give a very short overview of the main ideas as the topic is not in the focus of this thesis. Nevertheless, every signature–based standard basis algorithm can be equipped with a so–called *F4–ish reduction*, thus the importance of the knowledge of the main ideas should be self–evident.

Note that we explain the main ideas only in terms of ideals resp. polynomials w.r.t. a well–order on $\mathcal{P}$, to keep this introduction as easy as possible and to not confuse the reader with overwhelming notations.

In the late 1970s Lazard was the first who discovered a relationship between the computation of a standard basis and the computation of the resultant of the Sylvester matrix ([114, 116, 150]). In these days elimination theory got some new life and bigger examples started to be computable.

In these first approaches the computation of the resultant was restricted to two polynomials in only 1 variable. Having two polynomials $f = \sum_{i=0}^{k} a_i x^i, g = \sum_{j=0}^{l} b_j x^j \in \mathcal{K}[x]$ the Sylvester matrix of f and g is defined to be

$$\begin{array}{r} l \text{ times}\left\{\begin{array}{c} \\ \\ \\ \\ \end{array}\right. \\ k \text{ times}\left\{\begin{array}{c} \\ \\ \\ \\ \end{array}\right. \end{array}\begin{pmatrix} a_0 & \dots & a_k & 0 & & \dots & 0 \\ 0 & a_0 & \dots & a_k & 0 & \dots & 0 \\ & & & \ddots & & & \\ 0 & & \dots & 0 & a_0 & \dots & a_k \\ b_0 & \dots & b_l & 0 & & \dots & 0 \\ 0 & b_0 & \dots & b_l & 0 & \dots & 0 \\ & & & \ddots & & & \\ 0 & & \dots & 0 & b_0 & \dots & b_l \end{pmatrix}$$

where k need not be equal to l. If we denote the above matrix by $\mathrm{Syl}(f,g)$ we have the property that $\gcd(f,g)$ is not constant if and only if $\det\big(\mathrm{Syl}(f,g)\big) = 0$. The main problems of this method are:

(1) It is only usable in the univariate case.

(2) It is only usable for two polynomials. One can use this method recursively on more than two polynomials, but then the degree of the generated polynomials increases exponentially, thus the performance is very bad.

A generalization of the Sylvester matrix is the *Macaulay matrix*, discovered first in [120]: This solves the above mentioned drawbacks of the Sylvester matrix: It can be used in the multivariate case and for finitely many polynomials at the same time. Thus solving algebraic systems was possible with this construction.

Sadly, this was still not optimal, and STD has a way better performance computing Gröbner bases to prepare the resolving of systems of algebraic equations. In 1999, Faugère presented the F4 Algorithm([61]). The main differences to STD are:

(1) F4 does several normal form computations simultaneously.

(2) F4 uses Gaussian elimination to compute the normal forms of s–polynomials.

(3) F4 precomputes all possible reducers for a bunch s–polynomials before any reduction step takes place.

F4 transforms the polynomial data into rows of matrices. The reduction process itself is nothing else but a special Gaussian elimination[5] without swapping columns. We present the pseudo code of F4 in Algorithm 8.

Let us assume $F = \{f_1, \dots, f_r\}$ as input for F4. We want to compute the standard basis G for $\langle F \rangle$.

[5] In the following we always use "Gaussian elimination" as short notation for "Gaussian elimination without column swaps".

Algorithm 8 Faugère's F4 Algorithm (F4)

Input: $F = \{f_1, \dots, f_r\}$ a subset of $\mathcal{P}$ w.r.t. $<$
Output: G, a standard basis for $\langle F \rangle$ w.r.t. $<$

1: $G' \leftarrow \varnothing, H \leftarrow \varnothing, M \leftarrow \varnothing, P \leftarrow \varnothing$
2: $G \leftarrow f_1$
3: **for** $(i = 2, \dots, r)$ **do**
4: $\quad P \leftarrow \textsc{Update}(P, G, f_i)$
5: $\quad G \leftarrow G \cup \{f_i\}$
6: $l \leftarrow r$
7: **while** $(P \neq \varnothing)$ **do**
8: $\quad P' \leftarrow \textsc{Select}(P)$
9: $\quad P \leftarrow P \backslash P'$
10: $\quad (H, M) \leftarrow \textsc{SymPre}(P', G)$
11: $\quad G' \leftarrow \textsc{F4Reduction}(H, M)$
12: $\quad$ **while** $(G' \neq \varnothing)$ **do**
13: $\qquad h \leftarrow$ First element of G'
14: $\qquad P \leftarrow \textsc{Update}(P, G, h)$
15: $\qquad G \leftarrow G \cup \{h\}$
16: $\qquad G' \leftarrow G' \backslash \{h\}$
17: **return** G

(1) First of all we build critical pairs (f_i, f_j), precomputing the corresponding multipliers for the s–polynomial construction, i.e. terms t_i, t_j in $\mathcal{P}$ such that $t_i \operatorname{lt}(f_i) = t_j \operatorname{lt}(f_j)$. Note that using the algorithm Update defined in Section 2.4 we can avoid some useless pairs.

(2) After having computed these data we select a bunch of critical pairs P' out of the pair set P by some selection strategy. Next the so–called *symbolic prepocessing* starts in Line 10. The pseudo code of this part is given in Algorithm 9.

There we need to prepare the data set a bit (Lines 1 – 8): We compute the multiples for generating the s–polynomials corresponding to the critical pairs (Line 7) and store the corresponding elements in a set H. Besides this, we store all included monomials in another set called M (Lines 8 and 17). Note that the leading monomials are added to M at the end of the algorithm: We do not need to search for reducers of $\lambda_g \operatorname{lm}(g)$ and $\lambda_f \operatorname{lm}(f)$, they reduce each other as they are equal (see also Remark 2.5.1 (3)). Nevertheless, we need these monomials in M to be able to construct the matrix A as explained in the following.

In Line 13 we search for reducers in G whose leading monomials divide any monomial of the multiplied generators $\lambda_{f_i} \cdot f_i$ of the critical pairs, which are stored in the set H (Line 7), or the already found and multiplied reducers $t_{\text{red}} g_{\text{red}}$, also stored in H (Line 15).

In the end we return both, H and M, the sets of all multiplied polynomials necessary for the reduction (w.r.t. G) of the critical pairs in P'.

(3) This reduction process takes place in an algorithm called F4Reduction: We use the cardinalities of the sets H and M to define a matrix $A \in \mathcal{K}^{\#(H)\times\#(M)}$ including all data necessary. For all $i \in \{1,\dots,\#(H)\}$ and $j \in \{1,\dots,\#(M)\}$ the entry

$$A_{i,j} = \begin{cases} 0 & \text{if } m_j \notin \text{m-supp}(h_i), \\ \operatorname{lc}(t_k) & \text{if } t_k \in \operatorname{supp}(h_i) \text{ and } \operatorname{lm}(t_k) = m_j. \end{cases}$$

We can think of A as the matrix consisting of the coefficients of all polynomials in H, whereas each row corresponds to a polynomial in H and each column corresponds to a monomial in M.

At that point the normal form computations of the s–polynomials corresponding to the critical pairs in P' are nothing else but the Gaussian elimination of A without column swapping. This means that we compute the row echelon form of A.

In Lines 10 – 15 of Algorithm 10 we retransform the rows of A' to polynomials. From the standard basis point of view we are only interested in those rows resp. polynomials g_i s.t. $\operatorname{lm}(g_i) \notin L(G)$.

(4) In the end, back in Algorithm 8, we generate new critical pairs, again using Buchberger's criteria to find useless ones, and start again selecting a new bunch of critical pairs in Line 8.

Remark 2.5.1.

(1) Algorithm 8 is not the basic F4 Algorithm presented in [61]: The basic version of F4 does not include the algorithm Update to detect useless pairs. Neither is Algorithm 8 equivalent to the *improved* F4 from [61]: This improved version includes another optimization, the algorithm Simplify, which tries to choose better reducers. The discussion of this is postponed to the next section.

(2) Again note that in F4Reduction we *are not allowed* to swap columns when processing the Gaussian elimination for A. This would change the monomial order (the monomials labelling the columns would not be in decreasing order w.r.t. $<$ anymore).

(3) Also note that we only need to search for reducers of the monomials in $\text{m-supp}\left(f - \operatorname{lm}(f)\right)$ resp. $\text{m-supp}\left(g - \operatorname{lm}(g)\right)$ in SymPre due to the fact that the leading monomials are equal, $\lambda_f \operatorname{lm}(f) = \lambda_g \operatorname{lm}(g)$. Thus the corresponding leading coefficients are in one column. It follows that they reduce themselves and no further reducer must be searched in G. This is similar to the usual normal form computation: We first build the s–polynomial. By this, the leading terms already cancel out each other, and we search for reducers of the terms left in $\mathcal{S}(f,g)$.

A similar argument holds for the intermediate added reducers in Line 16 of Algorithm 9. Of course, these leading monomials need to be added to M to determine the number of columns of A correctly in F4Reduction (Line 17 in Algorithm 9).

The following theorem is proven in [61]:

Algorithm 9 Symbolic preprocessing of possible reducers (SymPre)

Input: P a set of critical pairs, G a set of reducers
Output: H a set of polynomials, M a set of monomials

$H \leftarrow \varnothing$, $M \leftarrow \varnothing$, $D \leftarrow \varnothing$
while $(P \neq \varnothing)$ **do**
  $(f, g) \leftarrow$ First element of P
  $P \leftarrow P \backslash \{(f, g)\}$
  $\lambda_f \leftarrow \frac{\tau(f,g)}{\mathrm{lm}(f)}$
  $\lambda_g \leftarrow \frac{\tau(f,g)}{\mathrm{lm}(g)}$
  $H \leftarrow H \cup \{\mathrm{lc}(g)\lambda_f f, \mathrm{lc}(f)\lambda_g g\}$
  $M \leftarrow M \cup \left\{\lambda_f m_f \mid m_f \in \text{m-supp}\left(f - \mathrm{lt}(f)\right)\right\} \cup \left\{\lambda_g m_g \mid m_g \in \text{m-supp}\left(g - \mathrm{lt}(g)\right)\right\}$
while $(M \neq \varnothing)$ **do**
  Choose $m \in M$
  $M \leftarrow M \backslash \{m\}$
  $D \leftarrow D \cup \{m\}$
  if $(\exists h \in G$ s.t. $\mathrm{lm}(h) \mid m)$ **then**
    $\lambda_h \leftarrow \frac{m}{\mathrm{lm}(h)}$
    $H \leftarrow H \cup \{\lambda_h h\}$
    $M \leftarrow M \cup \left\{\lambda_h m_h \mid m_h \in \text{m-supp}\left(h - \mathrm{lt}(h)\right)\right\}$
$M \leftarrow D \cup \{\mathrm{lm}(h) \mid h \in H\}$
Sort M w.r.t. $<$, decreasing leading monomials.
return (H, M)

Theorem 2.5.2. *Let $F \subset \mathcal{P}$ be the input of* F4. *Then* F4 *is an algorithm computing a standard basis G of $\langle F \rangle$ w.r.t. $<$.*

Let us investigate the main differences between Std resp. GM and F4 a bit more closely:

(1) If we select only one critical pair at a time in Line 8 in Algorithm 8, F4 behaves very similar to GM: It reduces one s–polynomial, possibly adds new data to P and G, and then goes on to the next critical pair. The difference lies in the symbolic preprocessing: On the one hand, F4 searches for reducers and starts the reduction process (Gaussian elimination) after all possible reducers have been found. GM, on the other hand, searches for a reducer, and if one is found, the reduction immediately takes place. Then the next reducer is searched for, and so on.
F4 needs to precompute all reducers first, since otherwise one would not know the size of the matrix A. Other than that, it is a nice distinction between the different steps of the inner loop of a standard basis algorithm:

a) Generate critical pairs.

b) Search for reducers.

c) Reduce all preprocessed data.

Algorithm 10 Reduction process in F4 (F4REDUCTION)

Input: G and $H = \{h_1, \ldots, h_r\}$ sets of polynomials, $M = \{m_1, \ldots, m_s\}$ a set of monomials
Output: G' a set of polynomials

$F' \leftarrow \varnothing$
$A \leftarrow 0_{r \times s}$
for $\big(i = 1, \ldots, r\big)$ **do**
  for $\big(j = 1, \ldots, s\big)$ **do**
    if $(\mathrm{lm}(h_i) = m_j)$ **then**
      $A_{i,j} \leftarrow \mathrm{lc}(h_i)$
      $h_i \leftarrow h_i - \mathrm{lt}(h_i)$
$H \leftarrow \varnothing$, $M \leftarrow \varnothing$
$A' \leftarrow \textsc{Gauss}(A)$
for $\big(i = 1, \ldots, r\big)$ **do**
  $g_i \leftarrow 0$
  for $\big(j = 1, \ldots, s\big)$ **do**
    if $(A_{i,j} \neq 0)$ **then**
      $g_i \leftarrow g_i + A_{i,j} m_j$
  $F' \leftarrow F' \cup \{g_i\}$
$G' \leftarrow \{g_i \in F' \mid \mathrm{lm}(g_i) \notin L(G)\}$
return G'

(2) The real idea is *not* to select only one pair, but to get a subset P' of P including several elements and to precompute the whole reducer data beforehand, doing only one Gaussian elimination for all these elements.

Of course, one needs to be careful with the size of P': If P' is too big the resulting coefficient matrix A could be too large to be computed, possibly even to be stored on a computer. One needs to find a good choice which pairs should be taken. A more detailed discussion on this is given in Section 2.2. The best selection strategies for a wide range of examples are the normal selection and the sugar selection.

Let us give a small example of how F4 works, using the normal selection strategy:

Example 2.5.3. Let $F = \{f_1, f_2\} \subset \mathcal{K}[x, y, z]$ where

$$f_1 = xy - z^2$$
$$f_2 = y^2 - z^2.$$

We equip $\mathcal{K}[x, y, z]$ with the graded reverse lexicographical order $<_{\mathrm{dp}}$. Clearly, $G = \{g_1, g_2\}$ where $g_i = f_i$ for $i \in \{1, 2\}$. We generate the only possible critical pair and add it to P:

$$P := \big\{(g_2, g_1)\big\}.$$

Clearly, $P' = P$ and we enter the symbolic preprocessing: We first compute the multipliers of g_1 and g_2, and generate the sets H and M:

$$\begin{aligned}
\lambda_{g_1} g_1 &= y g_1 = xy^2 - yz^2 \\
\lambda_{g_2} g_2 &= x g_2 = xy^2 - xz^2 \\
\Rightarrow H &:= \{xy^2 - xz^2, xy^2 - yz^2\} \\
M &:= \{xy^2, xz^2, yz^2\}.
\end{aligned}$$

Note that no reducer is found in G for xz^2 as well as yz^2, thus we compute the matrix A:

$$A_1 := \begin{array}{c} \begin{array}{ccc} xy^2 & xz^2 & yz^2 \end{array} \\ \left(\begin{array}{ccc} 1 & -1 & 0 \\ 1 & 0 & -1 \end{array}\right) \end{array} \begin{array}{l} \\ xg_2 \\ yg_1 \end{array}$$

Computing the row echelon form A_1' of A_1 we get a new polynomial g_3 for G:

$$A_1' := \begin{array}{c} \begin{array}{ccc} xy^2 & xz^2 & yz^2 \end{array} \\ \left(\begin{array}{ccc} 1 & -1 & 0 \\ 0 & 1 & -1 \end{array}\right) \end{array} \begin{array}{l} \\ xg_2 \\ yg_1 - xg_2 \end{array}$$

Retransforming the two rows we get the set F' consisting of

$$\begin{aligned}
g_1' &= xy^2 - xz^2 \\
g_2' &= xz^2 - yz^2
\end{aligned}$$

We see that $\operatorname{lm}(g_1') \in L(G)$, whereas $\operatorname{lm}(g_2') \notin L(G)$. Thus renaming $g_3 := g_2'$ we get $G' = \{g_3\}$. Only one new critical pair is generated, (g_3, g_1), as $\tau(g_3, g_2) = \operatorname{lm}(g_3)\operatorname{lm}(g_2)$ and thus the pair (g_3, g_2) is detected being useless by Buchberger's 1st Criterion.
Again we have only one element in $P' = \{(g_3, g_1)\}$. Computing as in the first iteration we get

$$\begin{aligned}
\lambda_{g_1} g_1 &= z^2 g_1 = xyz^2 - z^4 \\
\lambda_{g_3} g_3 &= y g_3 = xyz^2 - y^2z^2 \\
\Rightarrow H &:= \{xyz^2 - y^2z^2, xyz^2 - z^4\} \\
M &:= \{xyz^2, y^2z^2, z^4\}.
\end{aligned}$$

This time there exists a reducer, namely g_2, since $\operatorname{lm}(g_2) = y^2 \mid y^2z^2$. This means that we add z^2g_2 to H and the monomial part of $z^2\big(g_2 - \operatorname{lt}(g_2)\big) = -z^4$ to M. At this point no further reducers are found, thus we start again with F4Reduction:

$$A_2 := \begin{array}{c} \begin{array}{ccc} xyz^2 & y^2z^2 & z^4 \end{array} \\ \left(\begin{array}{ccc} 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & -1 \end{array}\right) \end{array} \begin{array}{l} \\ yg_3 \\ z^2g_1 \\ z^2g_2 \end{array}$$

$$\Rightarrow \quad A_2' := \begin{array}{c} \begin{array}{ccc} xyz^2 & y^2z^2 & z^4 \end{array} \\ \left(\begin{array}{ccc} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{array}\right) \end{array} \begin{array}{l} \\ yg_3 \\ z^2g_1 - yg_3 \\ z^2g_2 - (z^2g_1 - yg_3) \end{array}$$

We see that none of the retransformed polynomials has a leading monomial not already in $L(G)$. So, no new critical pairs are generated, $P = \emptyset$ and F4 terminates. We have computed the standard resp. Gröbner basis

$$G = \{xy - z^2, y^2 - z^2, xz^2 - yz^2\}.$$

Remark 2.5.4.

(1) The F4 Algorithm is implemented in several computer algebra systems: The initial implementation is done in Faugère's FGB package. This package can be used in the MAPLE system ([127]), too. Moreover, there exists a very efficient implementation of F4 by Allan Steel for MAGMA ([29]). Besides these low–level implementations various high–level implementations in interpreted languages are available, one of the most recent ones by Daniel Cabarcas ([36]).

(2) One of the most important advantages of F4 these days is the natural way of parallelizing the Gaussian elimination. In [69] Faugère and Lacharte present a strategy how to parallelize the matrix operations in polynomial rings over finite fields, e.g. grouping the matrix into different blocks, using different attempts for sparse, semi–sparse, and dense blocks[6], etc. For an 8–core architecture they achieve a speed–up between 6–8 in big examples like `Katsura-13` in characteristic 65,521.
Of course, one can also parallelize polynomial arithmetic, as shown in [128,129,156]. To achieve a good parallelization, this means a nearly linear one, in terms of the vast majority of polynomial storage structures is much harder than to achieve a similar result in F4.

In Section 5.1 we discuss signature–based criteria. We can easily change the algorithm UPDATE in Algorithm 8 to achieve a combination of signature–based standard basis algorithms and linear algebra reduction processes. Moreover, the order of the rows in the matrix A has a strong connection to the signatures attached to the polynomials.

Let us finish the discussion of the basic ideas of F4 by giving a transition to the next section: Although F4 already has an advantage over GM by using linear algebra to compute normal forms of s–polynomials, the real enhancement can be found in the selection of the best possible reducer in F4. This is done using the algorithm SIMPLIFY. The selection of reducers is a very important topic, it is not only used in F4, but it can be applied to GM and other variants of STD, too. It has a big impact on the performance of the algorithms, not only considering timings, but also focussing on memory usage.

[6] Again just a sloppy note: The sparsity of a block B of a matrix A is defined by the difference of the number of elements in B and the number of zero entries in B.

2.6 Picking a good reducer

In Section 2.5 we have seen how to use linear algebra to reduce several s–polynomials at the same time using matrices. When starting the retransformation step in F4Reduction we only use those g' whose leading monomials are not already in $L(G)$ for further computations (see Line 16 in Algorithm 10). All the other computations which are done during the Gaussian elimination of A are not used in any further step for the computation of the standard basis, thus those steps are in some sense useless. The topic of this chapter is to give an overview of ideas how to prevent algorithms from useless computations.

This time we do not try to delete those useless computations beforehand, but try to make them useful for furhter reduction steps. Again, we restrict ourselves to the polynomial situation w.r.t. a well–order on $\mathcal{P}$.

Let us see how one can reuse already reduced elements, and how to choose a "good" reducer. We introduce this topic using F4, but one easily sees that this is a general problem of standard basis algorithms. There exist lots of papers on this topic, we restrict ourselves to the ideas given by Faugère in [61], and as a follow–up by Brickenstein in [30, 31]. This restriction is justified by the fact that the above methods are known to be applicable for the most part to signature–based standard basis algorithms.

Let us start with a closer look at F4's reduction process (Algorithm 10):

(1) We start storing all critical pairs and all possible reducers in the coefficient matrix A.

(2) Next we compute the row echelon form A' of A.

(3) At the end we investigate all retransformed polynomials from F':
   a) If $\operatorname{lm}(h) \notin L(G)$ we use g in the following.
   b) If $\operatorname{lm}(h') \in L(G)$ we do not use g' any more.

Step (3)b is the problematic one: h' is some, possibly reduced, multiple of an element $g_i \in G = \{g_1, \ldots, g_s\}$, i.e.

$$h' = \sum_{i=1}^{s} u_i g_i, \text{ where } \operatorname{lm}(h') = \operatorname{lm}(u_{k_0}) \operatorname{lm}(g_{k_0}) \text{ for some } k_0 \in \{1, \ldots, s\}. \tag{2.6.1}$$

It is very probable that in an upcoming reduction, i.e. Gaussian elimination, $\operatorname{lm}(h')$ is needed to reduce. Then F4 would add the multiple $\operatorname{lm}(u_{k_0})g_{k_0}$ in Algorithm SymPre to the set of reducers. But what happens next? SymPre adds all monomials of the product $\operatorname{lm}(u_{k_0})\,(g_{k_0} - \operatorname{lt}(g_{k_0}))$ to the set M (Line 16). Thus in the following also for these monomials appropriate reducers are searched for. Most of these reducers will be the very same as those already in the representation of h' given in 2.6.1. This means that we redo lots of reduction steps we have already done in the Gaussian elimination, from whose resulting matrix A' the polynomial h' was extracted.

Thus using h' as reducer instead of $\operatorname{lm}(u_{k_0})g_{k_0}$ saves us from doing stuff twice! Moreover, we only need to add the new reducers of h' which are possibly available at this time of the algorithm. All the other reducers are not added to H and thus not to A. This means that we have two very important optimizations:

(1) We re–use already known reductions, thus as few as possible reduction steps are done all in all. Moreover, reductions with the same reducers are preveneted as much as possible from taking place multiple times during the computations of F4.

(2) Another big advantage is the fact that we construct and store a much smaller matrix A in the following. Each new reducer added to H adds a new row to A. Using h' instead of $\mathrm{lm}(u_{k_0})g_{k_0}$ and all its reducers g_i means that we add only 1 line instead of several ones. Moreover, it is possible that h' is sparser than g_{k_0}, which means that it includes less monomials than $\mathrm{lm}(u_{k_0})g_{k_0}$. Thus even more $\#(M)$ does not increase as much using h' as it would otherwise. This leads to less reducer searching in SymPre and less columns in A.

This idea of reusing as much as possible already done reduction steps is included in the *improved version of* F4[7]. We mark the changes from F4 to the improved F4 in Algorithms 11 – 13 in the pseudo code; Algorithm 14 is completely new.

We need to add some more bookkeeping elements to the algorithms enabling F4 to keep track of those elements in Algorithm 10 which are in $F' \backslash G'$. We need to store these in a set B and check them for being possibly better reducers in the following steps. The structure of B can be characterized as follows:

(1) $\#(B) = \#(G)$, this means that for every element $g \in G$ there exists a corresponding element $B_g \in B$.

(2) Every B_g itself is a set containing tuples of the type (m_g, p_g), where m_g is a monomial in $\mathcal{P}$, p_g a polynomial. In general for an element $g \in G$ we define the map $\varphi_g : \mathcal{P} \to \mathcal{P}$ by

$$\varphi_g(m_g) = \begin{cases} p_g & \text{if } (m_g, p_g) \in B_g, \\ 0 & \text{else.} \end{cases}$$

(3) For every such element (m_g, p_g) the following holds:

a) $m_g g$ was considered in some previous Gaussian elimination as input row of A.

b) $m_g g$ reduced to p_g where $m_g \mathrm{lm}(g) = \mathrm{lm}(p_g)$.

Thus, whenever the Gaussian elimination in Algorithm 13 has been finished we check all elements $g_i \in \mathcal{P}$ (retransformed from A') whether their leading terms are already in $L(G)$ or not. Based on this they are either added to G' (Line 17) or the multiplier $m_g = \frac{\mathrm{lm}(g_i)}{\mathrm{lm}(g)}$ is computed and the tuple (m_g, g_i) is added to B_g, where $\mathrm{lm}(g) \mid \mathrm{lm}(g_i)$ (Line 20).

We see that, besides storing information of previously done reductions in the set B, the algorithm Simplify is called in SymPre. This is the main optimization, described in Algorithm 14.

[7] In the following chapters we always consider the improved version of F4. Thus we keep the already introduced notation for all algorithms being part of F4.

Algorithm 11 Improved F4 Algorithm (F4)

Input: $F = \{f_1, \dots, f_r\}$ a subset of $\mathcal{P}$ w.r.t. $<$
Output: G, a standard basis for $\langle F \rangle$ w.r.t. $<$

$B \leftarrow \varnothing$, $G' \leftarrow \varnothing$, $H \leftarrow \varnothing$, $M \leftarrow \varnothing$, $P \leftarrow \varnothing$
$G \leftarrow f_1$
for $(i = 2, \dots, r)$ **do**
    $P \leftarrow \textsc{Update}(P, G, f_i)$
    $G \leftarrow G \cup \{f_i\}$
$l \leftarrow r$
while $(P \neq \varnothing)$ **do**
    $P' \leftarrow \textsc{Select}(P)$
    $P \leftarrow P \setminus P'$
    $(H, M) \leftarrow \textsc{SymPre}(P', G, B)$
    $(G', B) \leftarrow \textsc{F4Reduction}(H, M, B)$
    while $(G' \neq \varnothing)$ **do**
        $h \leftarrow$ First element of G'
        $P \leftarrow \textsc{Update}(P, G, h)$
        $G \leftarrow G \cup \{h\}$
        $G' \leftarrow G' \setminus \{h\}$
return G

Let us explain the important steps of Simplify: It receives three arguments, namely a monomial m, a polynomial f and the set B. m and f have been selected by SymPre to be added to the list of elements which build the coefficient matrix A in F4Reduction. At this point we do not add $m \cdot f$ to H and all monomials in $m\big(f - \mathrm{lt}(f)\big)$ to M, but we search for possibly further reduced elements corresponding to $m \cdot f$ (Lines 7, 8, and 17 in Algorithm 12).

So, how does this search works? It just implements our ideas from this section's introduction: We search for all possible divisors u of m and check if $\varphi_f(u) \neq 0$, i.e. the tuple $(u, p) \in B_f$, where $p = \varphi_f(u)$ (Line 4). If this is the case we replace m resp. f by $\frac{m}{u}$ resp. p (Line 6). This process goes on recursively until no new replacement can be done[8].

Besides the clear *improvement* of

(1) doing less reductions multiple times,

(2) using possible reducers with sparser tails, and

(3) having smaller matrices A

we also need to understand the *drawbacks* of the improved version of F4:

(1) Sometimes the replacement of a reducer is not sparser at all. Thus lots of checks in SymPre have to be done and the column size of A does not strictly decrease.

[8] Of course, this is not an efficient way to implement it. The description focusses on the explanation of the idea.

Algorithm 12 Improved Symbolic preprocessing of possible reducers (SymPre)

Input: P a set of critical pairs, G a set of reducers, B a set of sets of polynomal tuples
Output: H a set of polynomials, M a set of monomials

$H \leftarrow \emptyset$, $M \leftarrow \emptyset$, $D \leftarrow \emptyset$
while $(P \neq \emptyset)$ **do**
  $(f, g) \leftarrow$ First element of P
  $P \leftarrow P \backslash \{(f, g)\}$
  $\lambda_f \leftarrow \frac{\tau(f,g)}{\operatorname{lm}(f)}$
  $\lambda_g \leftarrow \frac{\tau(f,g)}{\operatorname{lm}(g)}$
  $f' \leftarrow \textsc{Simplify}(\lambda_f, f, B)$
  $g' \leftarrow \textsc{Simplify}(\lambda_g, g, B)$
  $H \leftarrow H \cup \{\operatorname{lc}(g) f', \operatorname{lc}(f) g'\}$
  $M \leftarrow M \cup \{m_{f'} \mid m_{f'} \in \text{m-supp}\,(f' - \operatorname{lt}(f'))\} \cup \{m_{g'} \mid m_{g'} \in \text{m-supp}\,(g' - \operatorname{lt}(g'))\}$
while $(M \neq \emptyset)$ **do**
  Choose $m \in M$
  $M \leftarrow M \backslash \{m\}$
  $D \leftarrow D \cup \{m\}$
  if $(\exists h \in G \text{ s.t. } \operatorname{lm}(h) \mid m)$ **then**
    $\lambda_h \leftarrow \frac{m}{\operatorname{lm}(h)}$
    $h' \leftarrow \textsc{Simplify}(\lambda_h, h, B)$
    $H \leftarrow H \cup \{h'\}$
    $M \leftarrow D \cup \{m_{h'} \mid m_{h'} \in \text{m-supp}\,(h' - \operatorname{lt}(h'))\}$
$M \leftarrow M \cup \{\operatorname{lm}(h') \mid h' \in H\}$
Sort M w.r.t. $<$, elements decreasing.
return (H, M)

(2) Even though the matrices A are mostly of a smaller size, the memory consumption of the algorithm can increase: Now we need to store all the data of the old matrices in B. There is a lot of data in B that is possibly never used during the computations of F4. This can even result in the incomputability of examples due to memory overflow, whereas these examples can be worked out by the basic F4 Algorithm.

(3) Sometimes the replacement chosen by Simplify is not the best one (w.r.t. sparsity, coefficient growth, etc.). The problem is that the improved F4 Algorithm is not able to dynamically choose another reducer depending on the actual data it is just computing.

In [30, 31] Brickenstein discovered some improved version of the selection method, which reducers to be used, during a deeper inspection of F4. His ideas are implemented as the algorithm SlimGB in Singular ([49]) and PolyBoRi ([32]).
The intention is to check more properties, or different properties being related to the actual problem (characteristic of the underlying field, sparsity of polynomials, degree, etc.), of the possible reducers to decide which one is the best. Let us describe SlimGB in more detail,

Algorithm 13 Reduction process in the improved F4 (F4REDUCTION)

Input: G and $H = \{h_1, \ldots, h_r\}$ sets of polynomials, $M = \{m_1, \ldots, m_s\}$ a set of monomials, B a set of sets of polynomial tuples
Output: G' a set of polynomials, B a set of sets of polynomial tuples

1: $F' \leftarrow \varnothing$, $K \leftarrow \varnothing$
2: $A \leftarrow 0_{r \times s}$
3: **for** $(i = 1, \ldots, r)$ **do**
4:     **for** $(j = 1, \ldots, s)$ **do**
5:         **if** $(\mathrm{lm}(h_i) = m_j)$ **then**
6:             $A_{i,j} \leftarrow \mathrm{lc}(h_i)$
7:             $h_i \leftarrow h_i - \mathrm{lt}(h_i)$
8: $H \leftarrow \varnothing$, $M \leftarrow \varnothing$
9: $A' \leftarrow \mathrm{GAUSS}(A)$
10: **for** $(i = 1, \ldots, r)$ **do**
11:     $g_i \leftarrow 0$
12:     **for** $(j = 1, \ldots, s)$ **do**
13:         **if** $(A_{i,j} \neq 0)$ **then**
14:             $g_i \leftarrow g_i + A_{i,j} m_j$
15:     $F' \leftarrow F' \cup \{g_i\}$
16:     **if** $(\mathrm{lm}(g_i) \notin L(G))$ **then**
17:         $G' \leftarrow G' \cup \{g_i\}$
18:     **else**
19:         $m_g \leftarrow \frac{\mathrm{lm}(g_i)}{\mathrm{lm}(g)}$ where $g \in G$
20:         $B_g \leftarrow B_g \cup \{m_g, g_i\}$
21: **return** (G', B)

giving its pseudo code and explaining the replacement strategies of polynomials.

Remark 2.6.1. We use the pseudo code of F4 as basis for SLIMGB. On the one hand, this is due to the fact that SLIMGB also reduces multiple s–polynomials at the same time. On the other hand, SLIMGB does not use linear algebra for the reduction, but an updated version of the normal form algorithms presented in Section 1.7. We give the pseudo code of this variant, called SLIMNF based on the code of GNF. Of course, one can use the ideas given here also updating $\mathrm{GNF}_{\mathrm{red}}$ or LNF, but for the purpose of this section we want to keep notation as easy as possible and focus on the choice of polynomials in the normal form. Again we highlight the lines which are newly inserted or updated w.r.t. the Algorithm 11. SLIMNF itself is completely new based on the fact that we do not only reduce several polynomials at the same time, but also check for replacements.

We see the main change in SLIMGB starting in Line 11: Besides using Algorithm 16 for the normal form computations of the set H of selected s–polynomials, it returns two different values:

(1) A set G' of polynomials reduced w.r.t. G: Those elements generate new critical pairs and are added to G afterwards.

Algorithm 14 Simplifying the reduction process in F4 (Simplify)

Input: m a monomial in $\mathcal{P}$, f a polynomial in $\mathcal{P}$, B a set of sets of polynomial tuples
Output: h a polynomial in $\mathcal{P}$

$D \leftarrow \{\text{divisors of } m\}$
while $(u \in D)$ **do**
  $D \leftarrow D \setminus \{u\}$
  if $\varphi_f(u) \neq 0$ **then**
    if $(u \neq m)$ **then**
      return Simplify $\left(\frac{m}{u}, \varphi_f(u), B\right)$
    else
      return $\varphi_f(u)$
return $m \cdot f$

(2) A set E of polynomial tuples for exchanging polynomials by better ones found during the computations of SlimNF: For all these tuples (h, p) it holds that $h \in G \cup R$ and $\mathrm{lm}(h) \mid \mathrm{lm}(p)$. Two different situations can happen:

  a) If $\deg(h) = \deg(p)$, then $\mathrm{lm}(h) = \mathrm{lm}(p)$. This means that we exchange the element $h \in G \cup R$, since p has better properties than h (Line 16 of Algorithm 15).

  b) If $\deg(h) < \deg(p)$, then we add p to the list of reducers R (Line 18 of Algorithm 15).

Remark 2.6.2.

(1) The set R is only used for reduction purposes in SlimNF, we do not build any new critical pairs with an element from R. For any $r \in R$ there exists a $g \in G$ such that $\lambda = \frac{\mathrm{lm}(r)}{\mathrm{lm}(g)}$. Assuming the situation of reducing an element h in SlimNF such that $\mathrm{lm}(h) = \lambda \,\mathrm{lm}(g)$ one should try to use r as a reducer instead of λg.

(2) Again, note that the presented pseudo code does not focus on efficiency, but on educational aspects. Of course, the way r is chosen in Line 5 of Algorithm 16 should be implemented in the vein of Algorithm 14.

The three main differences between the ideas of the improved version of F4 and SlimGB are:

(1) Not only the already computed s–polynomials are checked for replacements, even the generators of the corresponding critical pairs are replaced. In later steps, critical pairs generated by new polynomials use the replaced element, not the old one. This sometimes leads to a better performance of the algorithm.

(2) Algorithm 17, used in Line 1 of SlimNF, does not just compare r and h depending on *when they are computed*, but on more properties, even fitted to the requirements of the given input the standard basis should be computed of. See Example 2.6.3 for more details.

Algorithm 15 SlimGB Algorithm computing a standard basis w.r.t. $<$ (SlimGB)

Input: $F = \{f_1, \dots, f_r\}$ a subset of $\mathcal{P}$ w.r.t. $<$
Output: G a standard basis for $\langle F \rangle$ w.r.t. $<$

$E \leftarrow \varnothing$, $G' \leftarrow \varnothing$, $P \leftarrow \varnothing$, $R \leftarrow \varnothing$
$G \leftarrow f_1$
for $(i = 2, \dots, r)$ **do**
  $P \leftarrow \textsc{Update}(P, G, f_i)$
  $G \leftarrow G \cup \{f_i\}$
$l \leftarrow r$
while $(P \neq \varnothing)$ **do**
  $P' \leftarrow \textsc{Select}(P)$
  $P \leftarrow P \backslash P'$
  $H \leftarrow \{\mathcal{S}(f, g) \mid (f, g) \in P'\}$
  $(G', E) \leftarrow \textsc{SlimNF}(H, G, R)$
  while $(E \neq \varnothing)$ **do**
    $(h, p) \leftarrow$ First element of E
    $E \leftarrow E \backslash \{(h, p)\}$
    if $\big(\deg(h) = \deg(p)\big)$ **then**
      $h \leftarrow p$
    else
      $R \leftarrow R \cup \{p\}$
  while $(G' \neq \varnothing)$ **do**
    $h \leftarrow$ First element of G'
    $P \leftarrow \textsc{Update}(P, G, h)$
    $G \leftarrow G \cup \{h\}$
    $G' \leftarrow G' \backslash \{h\}$
return G

(3) Moreover, SlimGB only stores new reducers, if they are really better and necessary. F4 on the contrary stores all data not having new leading monomials. Thus a huge amount of memory has to be allocated, whereas only a small part of it is really useful.

As we see in Algorithm 17 the whole check whether to replace an element resp. to add a new reducer depends on the comparison of some property of the polynomials. This property can be determined problem–oriented. Let us give some possible and useful examples.

Example 2.6.3. Assume the polynomial $p \in \mathcal{P}$. The following properties can be of interest when computing standard bases:

(1) Length strategy: $\textsc{Property}(p) = \#\big(\operatorname{supp}(p)\big)$,

(2) Coefficient–length strategy: $\textsc{Property}(p) = \operatorname{lc}(p) \cdot \#\big(\operatorname{supp}(p)\big)$,

(3) Elimination strategy:

$$\textsc{Property}(p) = \sum_{m \in \text{m-supp}(p)} \Big(1 + \max\Big\{\deg(m) - \deg\big(\operatorname{lm}(p)\big), 0\Big\}\Big).$$

Algorithm 16 Normal form w.r.t. G of SlimGB (SlimNF)

Input: $H \subset \mathcal{P}$ a finite sequence, $G \subset \mathcal{P}$ a finite sequence, $R \subset \mathcal{P}$ a finite sequence
Output: $G' \subset \mathcal{P}$ a finite sequence, E a set of polynomial tuples

```
E ← ∅, G' ← ∅
while (H ≠ ∅) do
    h ← First element of H
    while (h ≠ 0 and D_h ← {r ∈ G ∪ R ∪ H | lm(r) | lm(h)} ≠ ∅) do
        Choose any r ∈ D_h.
        if (r ∈ G ∪ R and Replace?(r, h)) then
            E ← E ∪ {(r, h)}
        h ← h − (lt(h)/lt(r)) r
    if (h ≠ 0) then
        G' ← G' ∪ {h}
return (G', E)
```

Algorithm 17 Replacement check for SlimGB (Replace?)

Input: f, g polynomials in $\mathcal{P}$
Output: TRUE if a replacement should happen, FALSE otherwise

```
if (Property(f) > Property((lt(f)/lt(g)) g)) then
    return TRUE
return FALSE
```

(4) Coefficient–elimination strategy:

$$\textsc{Property}(p) = \mathrm{lc}(p) \cdot \sum_{m \in \text{m-supp}(p)} \left(1 + \max\left\{ \deg(m) - \deg\left(\mathrm{lm}(p)\right), 0\right\}\right).$$

Of course, other strategies are possible, but these are useful for a wide range of examples, for example, the Coefficient–length strategy gives a huge speed–up for computations over function fields and the Elimination strategy improves computations w.r.t. lexicographical orders up to a factor of 1000 compared to the usage of the Length–strategy.

Let us give a last example, which tries to convince the reader, that the replacement of polynomials in SlimGB really is advanced to the usage of Simplify in F4.

Example 2.6.4. Let us give a short comparison on the behaviour of the two replacement strategies presented in this section: Assume the point of the computations at which we want to reduce an element h. We see that $\mathrm{lm}(h) = \lambda_1\lambda_2 \,\mathrm{lm}(g_k)$ for some monomials λ_1, λ_2 and a possible reducer g_k. In F4, it could be possible that Simplify changes $\lambda_1\lambda_2 g_k$ to $\lambda_1 g_l$, where $\mathrm{lm}(g_l) = \lambda_2 \,\mathrm{lm}(g_k)$, although g_l is not sparser or better reduced than $\lambda_2 g_k$. On top of that, there can exist a much better element g_m with $\mathrm{lm}(g_m) = \lambda_1 \,\mathrm{lm}(g_k)$, which is blocked by g_l. This situation is nearly impossible to achieve in SlimGB, whereas it is rather possible in F4.

Remark 2.6.5.

(1) Of course, one can combine both attempts of optimizing polynomial data in the reduction process. This should be done to get highly optimized standard basis algorithms. In the end, the level of optimization mostly depends on the input data, i.e. the ideal, the order, etc. Based on this one has to decide which is the best strategy to be used and which reducers are replaced. As the behaviour of a standard basis computation is highly not predictable, heuristics must be implemented.

(2) One must be aware that the check and possible storage of a new element in Line 1 in SlimNF together with the whole bookkeeping done in the lines 12 – 18 in SlimGB produces an overhead in memory and timings. Based on the input, this can lead to a decline instead of an improvement. Thus a good heuristic is needed to decide, when to use which replacement and how strong the criteria for choosing the right reducer should be.

One last note on optimizing the choice of a possible reducer in the case of using LNF: In Algorithm 3 the ecart is used to ensure termination of the normal form computations, even if $<$ is a local order. In [92] it is mentioned that using a *weighted ecart* to choose the next reducer can speed up the computations for some good choice of the weight. As this section is restricted to the polynomial case, we also give the following definition in this setting. It should be clear how to generalize the definition to the more arbitrary situation of modules.

Definition 2.6.6. Let $w \in \left(\mathbb{R}^+\right)^n$ where $m \leq n$. Let $p = \sum_\alpha c_\alpha \boldsymbol{x}^\alpha \in \mathcal{P}$ be a polynomial.

(1) We define the *weighted degree of p w.r.t. w* by

$$\deg_w(p) := \max\left\{\sum_{i=1}^n w_i \cdot \alpha_i \mid c_\alpha \neq 0\right\}.$$

(2) Moreover, we define the *weighted ecart of p w.r.t. w* by

$$\mathrm{ecart}_w(p) := \deg_w(p) - \deg_w\left(\mathrm{lm}(p)\right).$$

This finishes our discussion of improving the usage of reducers during a standard basis computation. The research in this field is of high importance for signature–based standard basis algorithms, since we see in Chapter 5 that there the freedom of choice is restricted.

2.7 Using the Hilbert–Poincaré series

In this section we present the idea of how to use the Hilbert polynomial to improve standard basis computations. In Section 1.6 we have introduced the notions of the Hilbert–Poincaré series and shown its connection to the Hilbert polynomial. Here we discuss the so–called *Hilbert–driven standard basis algorithm*, which was presented first by Traverso in [154]. Note that the ideas discussed in Sections 2.8 and 2.9 are also influenced by the Hilbert–Poincaré series. All of these attempts have in common that one needs to know the Hilbert–Poincaré series beforehand to take advantage of it in an upcoming computation. For example, some work of how to achieve it can be found in [20–22].

Again, we restrict ourselves to the situation of computing a standard basis for an ideal I in $\mathcal{P}$.

In some special situations, we even know the Hilbert–Poincaré series without any further computations:

Theorem 2.7.1. *Let $<$ be an order on $\mathcal{P}$, $I \subset \mathcal{P}$ a homogeneous ideal. Then*

$$\mathrm{HP}_{\mathcal{P}/I}(t) = \mathrm{HP}_{\mathcal{P}/L(I)}(t).$$

Proof. See for example Section 5.2 in [97]. □

Using the above theorem one can conclude the following nice statement.

Corollary 2.7.2. *Let $I \in \mathcal{P}$ be an ideal, let $<$ a global order, and let $G = \{g_1, \ldots, g_s\} \subset I$. Then the following properties for the corresponding Hilbert functions hold:*

(1) $\mathrm{H}_{\mathcal{P}/L(I)}(d) \leq \mathrm{H}_{\mathcal{P}/L(G)}(d)$ for all d.

(2) If $\mathrm{H}_{\mathcal{P}/L(I)}(d) = \mathrm{H}_{\mathcal{P}/L(G)}(d)$ for all d, then G is a Gröbner basis for I.

Proof.

(1) This follows from the fact that $L(G) \subset L(I)$.

(2) Having $L(G) \subset L(I)$ the equality of the Hilbert functions follows from the equality of the leading ideals, i.e. $L(G) = L(I)$. But this is just the definition of G being a Gröbner basis for I.

□

Corollary 2.7.3. *Let $<_1$ and $<_2$ be two global orders on $\mathcal{P}$, $I \subset \mathcal{P}$ an ideal.*

(1) If I is homogeneous, then $\mathrm{H}_{\mathcal{P}/L_{<_1}(I)}(d) = \mathrm{H}_{\mathcal{P}/I}(d) = \mathrm{H}_{\mathcal{P}/L_{<_2}(I)}(d)$ for all d.

(2) If I is inhomogeneous, then $\mathrm{H}_{\mathcal{P}/L_{<_1}(I)}(d) = \mathrm{H}_{\mathcal{P}/I}(d) - \mathrm{H}_{\mathcal{P}/I}(d-1) = \mathrm{H}_{\mathcal{P}/L_{<_2}(I)}(d)$ for all d.

Proof. See for example [154]. □

Another very nice corollary from [154] gives us the possibility to use the Hilbert–Poincaré series even for improving the computations of Gröbner bases for inhomogenous input ideals. Whenever we have already computed a Gröbner basis G_1 for I for an order $<_1$, we can compute a Gröbner basis G_2 for I w.r.t. $<_2$ without caring for any degree drop during the computations.

Corollary 2.7.4. *Let $I \subset \mathcal{P}$ be an ideal, $<_1$ and $<_2$ global orders on $\mathcal{P}$, and let G_1 be a Gröbner basis for I w.r.t. $<_1$. Starting the computation of a Gröbner basis G_2 for I w.r.t. $<_2$ with G_1 as input, we can use the following variant of a standard basis algorithm:*

(1) Consider critical pairs by increasing degree.

(2) During a reduction step: Whenever the degree is decreased, the reduced element can be deleted and the next pair can be computed.

This is useful, considering that the computation of a Gröbner basis w.r.t. $<_1$ could be much easier than the computation w.r.t. $<_2$. Thus using the easier computation as basis for the harder computation enables us to improve the hard computation by applying the variant described in Corollary 2.7.4.

Remark 2.7.5. The usage of the equality of the Hilbert function in different global orders is a narrowed variant of the basic ideas behind the improvements of standard basis algorithms presented in Section 2.8: Compute the standard basis w.r.t. to an easier order and try to transform it into a standard basis w.r.t. the requested order without doing the complete standard basis computation again. This is just a combination of corollaries 2.7.3 and 2.7.4.

Using the notations from Section 1.6 we can present the crucial statement from [154].

Theorem 2.7.6. *Let I and J be two homogeneous ideals in $\mathcal{P}$ such that $J \subset I$. By Theorem 1.6.4 there exist polynomials $p(t) = \sum_{i=0}^{v} p_i t^i$, $q(t) = \sum_{j=0}^{w} q_j t^j$ such that the corresponding Hilbert–Poincaré series are*

$$\mathrm{HS}_{\mathcal{P}/I}(t) = \frac{p(t)}{(1-t)^n} \quad \textit{and} \quad \mathrm{HS}_{\mathcal{P}/J}(t) = \frac{q(t)}{(1-t)^n}.$$

Then the following conditions are equivalent:

(1) $\mathrm{H}_{\mathcal{P}/I}(t) = \mathrm{H}_{\mathcal{P}/J}(t)$ for all $1 \le t \le d-1$ and $\mathrm{H}_{\mathcal{P}/I}(d) < \mathrm{H}_{\mathcal{P}/J}(d)$.

(2) $p(i) = q(i)$ for $1 \le i \le d-1$ and $p(d) < q(d)$.

Definition 2.7.7.

(1) The *height of a prime ideal Q in $\mathcal{P}$* is defined by

$$\mathrm{ht}(Q) = \sup\{\mathrm{length}(C) \mid C \text{ are chains of prime ideals contained in } Q\}.$$

(2) The *height of an ideal I in $\mathcal{P}$* is defined by

$$\mathrm{ht}(I) = \inf\{\mathrm{ht}(Q) \mid Q \supset I, Q \text{ prime}\}.$$

(3) Let $I = \langle f_1, \ldots, f_r \rangle$ in $\mathcal{P}$ such that $r \le n$ and all f_i are homogeneous of degree d_i. If the ideal has height r, then the so–called *vanishing set*

$$V(I) = \{a \in \mathcal{K}^n \mid f_i(a) = 0 \text{ for all } i\}$$

of I is called a *complete intersection*.

(4) Let $(p_1, \ldots, p_r)$ be a squence in $\mathcal{P}$, F a finitely generated module in $\mathcal{M}$. We say that the sequence $(p_1, \ldots, p_r)$ is *regular (for $\mathcal{M}$)* if for each $1 \le i \le r$ it holds that

$$p_i \text{ is not a zerodivisor in } F/\langle p_1, \ldots, p_{i-1}\rangle F.$$

Example 2.7.8. Geometrically one can think of a complete intersection in the following way: Let $I = \langle f_1, \ldots, f_r\rangle$ be an homogeneous ideal in $\mathcal{P}$. $V(I) := \{a \in P^{n-1} \mid f_i(a) = 0 \text{ for all } 1 \le i \le r\}$ where P^{n-1} denotes the $(n-1)$–dimensional projective space. Similarly one can define $V(f_i) := \{a \in P^{n-1} \mid f_i(a) = 0\}$ for all $1 \le i \le r$. Now we say that $V(I)$ is a complete intersection if and only if $V(I) = \cap_{i=1}^r V(f_i)$. Thus the intersection of all those hypersurfaces $V(f_i)$ in P^{n-1} contains $V(I)$ and nothing else.

Remark 2.7.9. In [60] it is shown that if $V(I)$ is a complete intersection for $I = \langle f_1, \ldots, f_r\rangle$, then $(f_1, \ldots, f_r)$ is a regular sequence. We see in the following chapters that signature–based standard basis algorithms are in a strong connection to regular sequences. Furthermore, if the input of such an algorithm is a regular sequence, it is ensured that no zero reduction takes place.

The nice property of a complete intersection $V(I)$ is that we know the corresponding Hilbert–Poincaré series of I without the need of computing a standard basis for I beforehand:

Lemma 2.7.10. *If $V(I)$ is a complete intersection for $I = \langle f_1, \ldots, f_r\rangle$ where f_i is homogeneous of degree* $\deg(f_i) = d_i$ *for all $1 \le i \le r$, then the Hilbert–Poincaré series is*

$$\mathrm{HS}_{\mathcal{P}/I}(t) = \frac{\prod_{i=1}^r (1 - t^{d_i})}{(1-t)^n}.$$

Next we describe the *Hilbert–driven standard basis algorithm*. Using Theorem 2.7.6 one can improve standard basis computations. For this, we give the pseudo code based on the one of the Gebauer–Möller implementation (see Section 2.4): It is restricted to homogeneous input with the ideas presented here incorporated. Those new parts of Algorithm 18 are again highlighted.

Let $<_1$ be an order on $\mathcal{P}$, and let I be the ideal we want to compute the standard basis for. Assume furthermore that we already know the Hilbert function $\mathrm{H}_{\mathcal{P}/I}(t)$. This could be achieved by

(1) a previous Gröbner basis computation for I w.r.t. some other global order $<_2$,

(2) the fact that I corresponds to a complete intersection (see Lemma 2.7.10), or

(3) the fact that $I = \langle f_1, \ldots, f_r\rangle$, where $r \le n$. Then we can use the Hilbert–Poincaré series $\mathrm{HS}_{\mathcal{P}/I}(t) = \frac{\prod_{i=1}^r (1-t^{d_i})}{(1-t)^n}$ as an upper bound[9].

[9] If at some degree step in the Gröbner basis computation the bound does not hold any longer, the Gröbner basis computation goes on without any additional checks of the Hilbert function.

Algorithm 18 Hilbert–driven variant of GM w.r.t. a global order $<$ (HGM)

Input: $F = \{f_1, \ldots, f_r\}$ a subset of $\mathcal{P}$ of homogeneous elements, NF a normal form, $\mathrm{H}(t) := \mathrm{H}_{\mathcal{P}/\langle F \rangle}(t)$ the Hilbert function of $\langle F \rangle$
Output: G a standard basis for $\langle F \rangle$ w.r.t. $<$

$k := \infty$
$G \leftarrow f_1$
$P \leftarrow \varnothing$
$d' \leftarrow 0$
for $(i = 2, \ldots, r)$ **do**
  $P \leftarrow \textsc{Update}(P, G, f_i)$
  $G \leftarrow G \cup \{f_i\}$
$l \leftarrow r$
while $(P \neq \varnothing)$ **do**
  $d \leftarrow \min\left\{d \mid d = \deg\left(\mathcal{S}(f,g)\right), (f,g) \in P\right\}$
  $P' \leftarrow \left\{(f,g) \in P \mid \deg\left(\mathcal{S}(f,g)\right) = d'\right\}$
  $P \leftarrow P \backslash P'$
  while $(P' \neq \varnothing$ and $k > 0)$ **do**
    $(f,g) \leftarrow$ First element of P'
    $P' \leftarrow P' \backslash \{(f,g)\}$
    $h \leftarrow \mathcal{S}(f,g)$
    $h \leftarrow \mathrm{NF}(h, G)$
    if $(h \neq 0)$ **then**
      $f_{l+1} \leftarrow h$
      $P \leftarrow \textsc{Update}(P, G, f_{l+1})$
      $G \leftarrow G \cup \{f_{l+1}\}$
      $l \leftarrow l + 1$
      $k \leftarrow k - 1$
  if $(\mathrm{H}_{\mathcal{P}/L(G)}(t) = \mathrm{H}(t)$ for all $t)$ **then**
    return G
  else
    $d' \leftarrow \min\left\{t \in \mathbb{N} \mid \mathrm{H}_{\mathcal{P}/L(G)}(t) > \mathrm{H}(t)\right\}$
    $k \leftarrow \mathrm{H}_{\mathcal{P}/L(G)}(d') - \mathrm{H}(d')$
    $P'' \leftarrow \left\{(f,g) \in P \mid \deg\left(\mathcal{S}(f,g)\right) < d'\right\}$
    $P \leftarrow P \backslash P''$
return G

Let us denote $\mathrm{H}(t) = \mathrm{H}_{\mathcal{P}/I}(t)$ the known Hilbert function of I. In the following situations one can use the information stored in $\mathrm{HS}_{\mathcal{P}/I}(t)$.

(1) If I is homogeneous and $<_1$ is a global order on $\mathcal{P}$, then we can assume that we compute the Gröbner basis G for I by increasing degree (see Section 2.2). Thus, let us assume that we have already computed an intermediate Gröbner basis G, a d–Gröbner basis for I for some degree $d \geq 0$. At this point we compute the Hilbert

function $\mathrm{H}_{\mathcal{P}/L(G)}(t)$. It holds that

$$\mathrm{H}_{\mathcal{P}/L(G)}(t) = \mathrm{H}(t) \text{ for all } t \leq d.$$

Furthermore, we have the correspondence that

$$\mathrm{H}_{\mathcal{P}/L(G)}(t) = \mathrm{H}(t) + m_t \text{ for all } t, m_t \geq 0.$$

In the pseudo code the variable k is equivalent to m_t. In the beginning it is set to infinity as we do not have any information about G (Line 1). After the first degree step of reductions is done, the Hilbert function $\mathrm{H}_{\mathcal{P}/L(G)}(t)$ is computed and k is possibly adjusted (Line 28).

a) If $m_t = 0$ for all t, then G is a Gröbner basis for I (Line 25).

b) Otherwise we know from Theorem 2.7.6 that there exists some $d' > d$ such that $m_t = 0$ for all $t < d'$ and $m_{d'} \neq 0$. This means that in order to become a d'–Gröbner basis for I G needs $m_{d'}$ more elements in degree d'. Thus we know that exactly $m_{d'}$ critical pairs are useful. If we have added $m_{d'}$ elements of degree d' to G, we can stop treating any more critical pairs of degree d'. In Algorithm 18, k is checked whenever a new critical pair is treated for reduction purpose from P' (Line 13). As long as $k > 0$ the computations go on, otherwise enough critical pairs of degree d' are found and we can finish this degree step reduction.
After adding those $m_{d'}$ elements to G we recompute $\mathrm{H}_{\mathcal{P}/L(G)}(t)$ and go on with the next higher degree.

(2) If I is inhomogeneous and $<_1$ is a global order on $\mathcal{P}$, we cannot really use the ideas presented in Situation (1). If one has already computed a Gröbner basis G_2 of I w.r.t. another global order $<_2$ on $\mathcal{P}$, then the ideas of Corollary 2.7.4 can be used.
Another idea would be to homogenize the generators of I, compute the Gröbner basis G^{h} of the homogenized input according to Situation (1), and dehomogenize G^{h} in the end in order to receive the requested Gröbner basis $(G^{\mathrm{h}})^{\mathrm{deh}}$.

(3) If I is inhomogeneous and $<_1$ is a local order on $\mathcal{P}$, one can again homogenize the generators of I and go on as in Situation (2). Other ideas, again using the Hilbert–Poincaré series, can be found in Chapter 5 of [154].

Remark 2.7.11.

(1) Note that in Situation (1)b from above the improvements highly depends on the order, in which the critical pairs are computed: If the first critical pairs are the useful ones, the optimization is best. If those are at the end of the list of pairs to be reduced we still compute the zero reductions of the useless pairs investigated before them.

(2) One can think of the Hilbert–driven standard basis algorithm in the homogeneous case as a wrapper around any standard basis algorithm we already know: We select a bunch of critical pairs of lowest degree, compute all the normal forms of their corresponding s–polynomials as usual. Afterwards we compare the Hilbert functions. From this we get to know how many new elements of the next degree step we need to compute.

Using the presented ideas to switch from the computations over the rationals to computations over fields of finite characteristic p is postponed to Section 2.9. There we give an in–depth introduction to that topic.

2.8 Going the indirect way

In Section 2.7 we have seen that sometimes it has some benefits to compute a standard basis G_2 w.r.t. an order $<_2$ different from the one the origin problem is based on, say $<_1$. There we have used the Hilbert function to improve the computations w.r.t. the initial order. Moreover, we have seen in Corollary 2.7.4 that for computing G_1 w.r.t. $<_1$ starting from G_2 we can leave out some steps of the standard basis algorithm.

Moreover, recall the crucial differences in the complexity of the algorithms when computing w.r.t. lexicographical order resp. graded reverse lexicographical order we have seen in Section 1.9.

In this section we present more general attempts of this idea: In [133, 143] the notion of a *Gröbner fan* is introduced. All algorithms discussed here have one base frame: *They compute a Gröbner basis G_1 w.r.t. a given order $<_1$ by computing a Gröbner basis G_2 w.r.t. another order $<_2$ first and transform G_2 to G_1.*

Remark 2.8.1. We have noted in Section 1.8 that for some orders the computation of a standard basis can be done much faster and easier than for others. On the one hand, the order $<_{\mathrm{dp}}$ is much better for standard basis computation than $<_{\mathrm{lp}}$. On the other hand, a standard basis w.r.t. $<_{\mathrm{lp}}$ can be used way better in further applications than the one computed w.r.t. $<_{\mathrm{dp}}$. Thus the usage of the following algorithms is quite clear: We want to compute a standard basis w.r.t. an order for which the computation itself is pretty hard. Instead of going the direct way, we calculate the basis w.r.t. a much easier order and then transforming the result to a basis w.r.t. the requested order.

Thus the transformation at the end should not cost too much, otherwise the benefit of the computation w.r.t. the better order is lost.

Convention. Although there are attempts defining Gröbner fans for modules ([13]), we just want to explain the basic ideas behind the presented algorithms, which can be done much easier in the polynomial case with a well–order on $\mathcal{P}$, thus we restrict ourselves to this situation.

In [38] Caboara introduced a first attempt changing the examined order dynamically during ongoing Gröbner basis computations. We state the pseudo code of this algorithm based on the Gebauer–Möller implementation, called DGM, which stands for *dynamic Gebauer–Möller implementation*.

We see that Caboara's idea is plainly to dynamically adjust the order w.r.t. which the normal form of the next considered s–polynomial is computed. This adjustment is done in the very beginning (Line 1) and after each addition of a new element to G (Line 20). Both

Algorithm 19 Dynamic variant of GM w.r.t. a global order < (DGM)

Input: $F = \{f_1, \ldots, f_r\}$ a subset of $\mathcal{P}$ of homogeneous elements, NF a normal form, $\mathrm{H}(t) := \mathrm{H}_{\mathcal{P}/\langle F\rangle}(t)$ the Hilbert function of $\langle F\rangle$

Output: G a standard basis for $\langle F\rangle$ w.r.t. σ_0

$\sigma \leftarrow \textsc{InitialOrder}(f_1, \ldots, f_r)$
$G \leftarrow f_1$
$P \leftarrow \varnothing$
for $(i = 2, \ldots, r)$ **do**
  $P \leftarrow \textsc{Update}(P, G, f_i)$
  $G \leftarrow G \cup \{f_i\}$
$l \leftarrow r$
while $(P \neq \varnothing)$ **do**
  $P' \leftarrow \textsc{Select}(P)$
  $P \leftarrow P \backslash P'$
  while $(P' \neq \varnothing)$ **do**
    $(f, g) \leftarrow$ First element of P'
    $P' \leftarrow P' \backslash \{(f, g)\}$
    $h \leftarrow \mathcal{S}(f, g)$
    $h \leftarrow \mathrm{NF}(h, G)$
    if $(h \neq 0)$ **then**
      $f_{l+1} \leftarrow h$
      $P \leftarrow \textsc{Update}(P, G, f_{l+1})$
      $G \leftarrow G \cup \{f_{l+1}\}$
      $\sigma \leftarrow \textsc{NewOrder}(\sigma, G)$
      $l \leftarrow l + 1$
$G \leftarrow \textsc{Reduce}(G, \sigma_0)$
return G

of these algorithms, InitialOrder and NewOrder, try to keep the expected values of the Hilbert function $\mathrm{H}_{\mathcal{P}/L_\sigma(G)}(t)$ as small as possible. Let us discuss this for a moment:

Let $I := \langle F\rangle$. It always hold that $\mathrm{H}_{\mathcal{P}/L_\sigma(G)}(t) \geq \mathrm{H}_{\mathcal{P}/I}(t)$. So the idea is to keep the values of $\mathrm{H}_{\mathcal{P}/L_\sigma(G)}(t)$ as low as possible by choosing a good order σ. Thus for an input order σ for InitialOrder resp. NewOrder a new order, say τ, is returned, with the relation $L_\tau(G) = L_\sigma(G)$. For the idea of keeping the Hilbert function minimal we note two things:

(1) What is meant by *minimal*? One can think of minimal in terms of lexicographically minimal when considering $\mathrm{H}_{\mathcal{P}/L_\sigma(G)}(t)$ as a function. Another way would be to think about the Hilbert polynomial, demanding it to have minimal degree.
Caboara suggests a mix of these possibilities using heuristics, but is not giving a clear implementation of that.

(2) Note that whenever we change σ, not only $\mathrm{H}_{\mathcal{P}/L_\sigma(G)}(t)$ changes, but clearly also $\mathrm{H}_{\mathcal{P}/I}(t)$ changes. Thus the behaviour of algorithm NewOrder selecting σ to always minimize $\mathrm{H}_{\mathcal{P}/L_\sigma(G)}(t)$ in the above mentioned sense is not predictable, and can be

even worse than the behaviour of the basic Gebauer–Möller algorithm.

Considering the second remark above, one needs to find a good heuristic for the choice of σ. With the presented one we could at least hope for a better detection of useless critical pairs by UPDATE, which should lead to less zero reductions.

In the end, we compute the reduced Gröbner basis G w.r.t. σ_0. As all changes between different orders σ used during the computations preserve the leading ideal, it is enough to just reduce all elements $g \in G$ w.r.t. $G\backslash\{g\}$, throw away multiples and normalize all leading coefficients. This is done in REDUCE in Line 22. With this the Gröbner basis computation of G w.r.t. σ_0 finishes.

As mentioned, DGM's performance does highly depend on the choice of the next order, whose impact on the subsequent computations cannot be predicted besides some, rather naive, heuristics on the Hilbert function. A much more generalized variant of Gröbner basis computations using different orders is presented in the so–called *Gröbner walk*, first mentioned in [44]. For this we need some more notation, largely of plain combinatorial character:

Definition 2.8.2.

(1) Let $V := \{v_1, \ldots, v_r\} \subset \mathbb{R}^n$ be a finite set of vectors. The set

$$C(V) := \{\sum_{i=1}^{r} a_i v_i \mid a_i \in R^+, v_i \in V\}$$

is called a *(convex polyhedral) cone in* $\mathbb{R}^n$.

(2) The *dimension* $\dim(C)$ *of a cone* C is the dimension of the linear space C spans.

(3) The *dual of a cone* C is defined by

$$\check{C} := \{w \in \mathbb{R}^n \mid \langle w, v\rangle \geq 0, \text{ for all } v \in C\}$$

where $\langle \cdot, \cdot \rangle$ denotes the *dual pairing*.

(4) A *face* τ *of a cone* C is defined by

$$\tau := \{v \in C \mid \langle u, v\rangle = 0\}$$

for some $u \in \check{C}$. A face τ of C with dimension $\dim(\tau) = \dim(C) - 1$ is called *facet*.

(5) A *fan* Δ is a finite collection of (convex polyhedral) cones such that the following properties hold:

a) If $C \in \Delta$ and τ is a face of C, then $\tau \in \Delta$.

b) If $C_1, C_2 \in \Delta$, then $C_1 \cap C_2$ is a face of C_1 and of C_2.

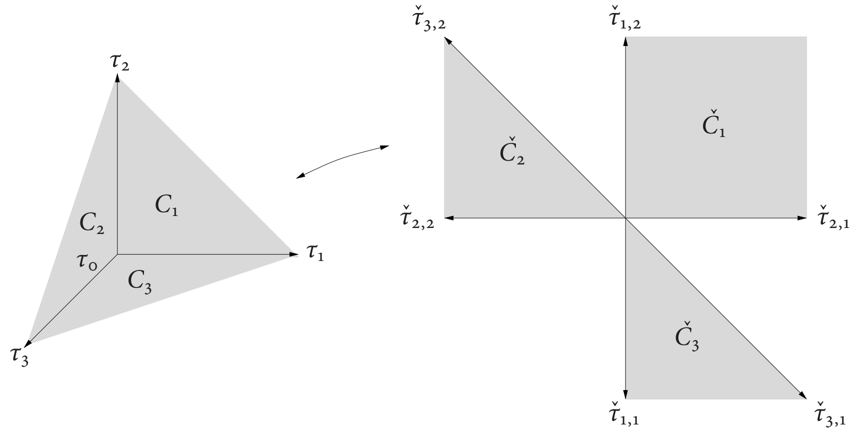

Figure 2.8.1: An example of fans, cones and faces

Example 2.8.3. We give an easy example in $\mathbb{R}^2$. Let $v_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $v_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, and $v_3 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$, illustrated in Figure 2.8.1. We have the fan Δ consisting of the cones C_1, C_2, and C_3, where

$$\begin{aligned} C_1 &:= \{av_1 + bv_2 \mid a, b \in \mathbb{R}^+\}, \\ C_2 &:= \{av_2 + bv_3 \mid a, b \in \mathbb{R}^+\}, \\ C_3 &:= \{av_1 + bv_3 \mid a, b \in \mathbb{R}^+\}. \end{aligned}$$

Moreover, we have the following faces:

$$\begin{aligned} \tau_0 &:= \{\vec{0}\}, \\ \tau_1 &:= \{av_1 \mid a \in \mathbb{R}^+\}, \\ \tau_2 &:= \{av_2 \mid a \in \mathbb{R}^+\}, \\ \tau_3 &:= \{av_3 \mid a \in \mathbb{R}^+\}. \end{aligned}$$

Clearly, the faces τ_1, τ_2 resp. τ_3 are facets of the corresponding cones C_1, C_2 resp. C_3 including them, whereas $\dim(\tau_0) = \dim(C_i) - 2$ for $i \in \{1, 2, 3\}$. However, treating τ_0 as a face of the other τ_i it is a facet of each τ_i.

On the right side of the picture we see the dual structures. Having the vectors $w_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$,

$w_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, $w_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $w_4 = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$, $w_5 = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$, and $w_6 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ we see that

$$\begin{aligned}
\check{C}_1 &:= \{aw_1 + bw_2 \mid a, b \in \mathbb{R}^+\},\\
\check{C}_2 &:= \{aw_3 + bw_4 \mid a, b \in \mathbb{R}^+\},\\
\check{C}_3 &:= \{aw_5 + bw_6 \mid a, b \in \mathbb{R}^+\},\\
\check{\tau}_{1,1} &:= \{aw_5 \mid a \in \mathbb{R}^+\},\\
\check{\tau}_{1,2} &:= \{aw_2 \mid a \in \mathbb{R}^+\},\\
\check{\tau}_{2,1} &:= \{aw_1 \mid a \in \mathbb{R}^+\},\\
\check{\tau}_{2,2} &:= \{aw_4 \mid a \in \mathbb{R}^+\},\\
\check{\tau}_{3,1} &:= \{aw_6 \mid a \in \mathbb{R}^+\},\\
\check{\tau}_{3,2} &:= \{aw_3 \mid a \in \mathbb{R}^+\}.
\end{aligned}$$

All in all, we can build the corresponding dual faces for τ_1, τ_2, and τ_3, and get

$$\begin{aligned}
\check{\tau}_1 &:= \check{\tau}_{1,1} \cup \check{\tau}_{1,2},\\
\check{\tau}_2 &:= \check{\tau}_{2,1} \cup \check{\tau}_{2,2},\\
\check{\tau}_3 &:= \check{\tau}_{3,1} \cup \check{\tau}_{3,2}.
\end{aligned}$$

Proposition 2.8.4. *The following statements hold:*

(1) *Any face is a convex polyhedral cone.*

(2) *Any intersection of faces is a face.*

(3) *Any face of a face is a face.*

(4) *Any proper face is contained in a facet.*

There is a wide range of good literature covering these structures, namely in the field of *toric geometry*, for example, see [75, 145, 147]. We refer to these for the reader interested in more details about those geometric structures and focus on our purpose of improving Gröbner basis computations using the above definitions.

Definition 2.8.5. Let $v, w \in \mathbb{R}^n$, $<$ a well–order on $\mathcal{P}$, $p = \sum_\alpha c_\alpha \boldsymbol{x}^\alpha$ a polynomial in $\mathcal{P}$ and $F = \{f_1, \ldots, f_r\} \subset \mathcal{P}$ finite.

(1) The *initial monomial of p w.r.t. v* is defined by $\mathrm{in}_v(p) = \max\{\deg_v(\boldsymbol{x}^\alpha) \mid c_\alpha \neq 0\}$.

(2) p is called *v–homogeneous* if $p = \mathrm{in}_v(p)$.

(3) The *initial ideal of F* is given by $\mathrm{in}_v(F) := \left\langle \mathrm{in}_v(f_1), \ldots, \mathrm{in}_v(f_r) \right\rangle$.

(4) $\overline{vw} := \left\{(1-\lambda)v + \lambda w\right\}$ denotes the *line segment between v and w*.

(5) The order $(v,<)$ on $\mathcal{P}$ defined by

$$\begin{aligned} \boldsymbol{x}^\alpha (v,<) \boldsymbol{x}^\beta :\Longleftrightarrow\ & \deg_v(\boldsymbol{x}^\alpha) <_{\text{nat}} \deg_v(\boldsymbol{x}^\beta) \text{ or} \\ & \deg_v(\boldsymbol{x}^\alpha) =_{\text{nat}} \deg_v(\boldsymbol{x}^\beta) \text{ and } \boldsymbol{x}^\alpha < \boldsymbol{x}^\beta \end{aligned}$$

is a *refinement of v by $<$*. $(v,<)$ refines v in the sense that whenever $\deg_v(\boldsymbol{x}^\alpha) < \deg_v(\boldsymbol{x}^\beta)$, then $\boldsymbol{x}^\alpha(v,<)\boldsymbol{x}^\beta$.

Remark 2.8.6.

(1) Note that a v–homogeneous polynomial p is homogeneous for $v = (1,\dots,1)$.

(2) There is a strong connection between $L_<(F)$ and $\text{in}_v(F)$ as we have already noted a strong connection between vectors in $\mathbb{R}^n$ resp. matrices in $\text{GL}(n,\mathbb{R})$ and monomial orders in Lemma 1.3.9. Note that whereas every element in $L_<(F)$ is a monomial this need not be true for the elements in $\text{in}_v(F)$. For example, let $F = \{z^4 - xy + 1\} \in \mathcal{K}[x,y,z,]$ and $v = (2,2,1)$, then $\text{in}_v(F) = \langle z^4 - xy\rangle$.

With this we are able to define the cones resp. fans we are interested in.

Definition 2.8.7. Let $<$ be a well–order on $\mathcal{P}$, $F = \{f_1,\dots,f_r\} \subset \mathcal{P}$ finite, $I = \langle F\rangle$, and let $G = \{g_1,\dots,g_s\} \subset \mathcal{P}$ be the reduced Gröbner basis for I w.r.t. $<$.

(1) For F we define the cone

$$C_<(F) := \left\{w \in (\mathbb{R}^+)^n \mid L_<\big(\text{in}_w(f)\big) = L_<(f) \text{ for all } f \in F\right\}.$$

The *Gröbner cone of G w.r.t.* $<$ is defined by $C_<(G)$.

(2) The *Gröbner fan* is the fan Δ_G consisting of a collection of $C_<(F)$ where $<$ runs over all well–orders on $\mathcal{P}$.

Remark 2.8.8. Note that there are only finitely many well–orders not equivalent to each other, see [133]. Thus the above definition of a Gröbner fan is well–defined.

The following lemma enables us to do a walk in the Gröbner fan without losing previously computed data.

Lemma 2.8.9. *Let I be an ideal in $\mathcal{P}$.*

(1) Let $<_1, <_2$ be two well–orders on $\mathcal{P}$, $G_{<_1}$ the reduced Gröbner basis for I w.r.t. $<_1$. Then $C_{<_1}(I) = C_{<_2}(I)$ if and only if $\text{lm}_{<_1}(g) = \text{lm}_{<_2}(g)$ for all $g \in G_{<_1}$.

(2) Let $u, v \in \mathbb{R}^n$ such that the well–order $<$ refines v. Then there exists a $w \in \overline{uv}$ such that $\overline{uw} \subseteq C_{(v,<)}(I)$.

In particular, Statement (2) of Lemma 2.8.9 enables us to walk around in the different cones of the Gröbner fan Δ_G, and in the special situation of w being on a facet of two cones, we can move into an adjacent cone.

Lemma 2.8.10. *Let $<_1, <_2$ be two different well–orders on $\mathcal{P}$, let $I = \langle f_1, \ldots, f_r \rangle$ be an ideal in $\mathcal{P}$, and let $w \in \mathbb{R}^n$ such that $w \in C_{<_1}(I) \cap C_{<_2}(I)$. Then $\mathrm{in}_w(I)$ is not a monomial ideal.*

Proof. On the one hand, there must exist at least one generator f_i of I such that $\mathrm{lm}_{<_1}(f_i) \neq \mathrm{lm}_{<_2}(f_i)$. On the other hand, $\mathrm{lm}_{<_1}\left(\mathrm{in}_w(f_i)\right) = \mathrm{lm}_{<_1}(f_i)$ as well as $\mathrm{lm}_{<_2}\left(\mathrm{in}_w(f_i)\right) = \mathrm{lm}_{<_2}(f_i)$. It follows that $\mathrm{in}_w(f_i)$ must consist of at least two monomials, namely $\mathrm{lm}_{<_1}(f_i)$ and $\mathrm{lm}_{<_2}(f_i)$. □

Convention. In the following rStd denotes any of the Gröbner basis algorithms we have already discovered with the condition that it returns the unique, reduced Gröbner basis at the end.

Algorithm 20 Gröbner walk to compute a reduced Gröbner basis (GBWalk)

Input: $G_{<_1} = \{g_1, \ldots, g_r\}$ a reduced Gröbner basis (for some ideal I) w.r.t. $<_1$, $u \in C_{<_1}(I)$ current weight vector, $v \in C_{<_2}(I)$
Output: $G_{<_2}$ a reduced Gröbner basis (for the ideal I) w.r.t. $<_2$

1: Compute $\lambda_0 \in \mathbb{R}$ such that $(1 - \lambda_0)u + \lambda_0 v \in C_{<_1}(I) \cap C_{<_2}(I)$.
2: $w \leftarrow (1 - \lambda_0)u + \lambda_0 v$
3: Refine w by $<_2 \Rightarrow (w, <_2)$
4: $M_w \leftarrow \textsc{rStd}\left(\mathrm{in}_w(G_{<_1}), (w, <_2)\right)$ such that $m_j = \sum_{i=1}^r h_{ij}\,\mathrm{in}_w(g_i)$ for all $m_j \in M_w$
5: $p_j \leftarrow \sum_{i=1}^r h_{ij} g_i$ such that $\mathrm{in}_w(p_j) = m_j$ for all j
6: $G' \leftarrow \{p_1, \ldots, p_{\#(M_w)}\}$
7: $G_{(w,<_2)} \leftarrow \textsc{Reduce}\left(G', (w, <_2)\right)$
8: Convert $G_{(w,<_2)}$ to a reduced Gröbner basis $G_{<_2}$.
9: **return** $G_{<_2}$

We assume that we have already computed a reduced Gröbner basis $G_{<_1}$ for some ideal I w.r.t. $<_1$. Thus we have a weight vector $u \in C_{<_1}(I)$ given and want to enter the adjacent cone $C_{<_2}(I)$. There we have a weight vector v, already precomputed. Doing this we need to cross the border of the two cones: We compute a weight vector $w \in C_{<_1}(I) \cap C_{<_2}(I)$ which is in both cones, as illustrated in Figure 2.8.2. It follows by Lemma 2.8.10 that $\mathrm{in}_w(G_{<_1})$ is not a monomial ideal. At this point we compute the reduced Gröbner basis M_w for $\mathrm{in}_w(G_{<_1})$ w.r.t. $(w, <_2)$ in Line 4. Note that all elements in $\mathrm{in}_w(G_{<_1})$ are w–homogeneous, thus all generated s–polynomials and all computed normal forms are so, too. Thus we find w–homogeneous elements h_{ij} which fulfill that

$$m_j = \sum_{i=1}^{r} h_{ij}\,\mathrm{in}_w(g_i) \text{ for all } m_j \in M_w.$$

After that we can easily get a Gröbner basis for I w.r.t. $(w, <_2)$ out of M_w by replacing all $\mathrm{in}_w(g_i)$ by g_i (Line 5). This has to be reduced to $G_{(w,<_2)}$ and then further transformed to receive the reduced Gröbner basis $G_{<_2}$ for I w.r.t. $<_2$.

Remark 2.8.11.

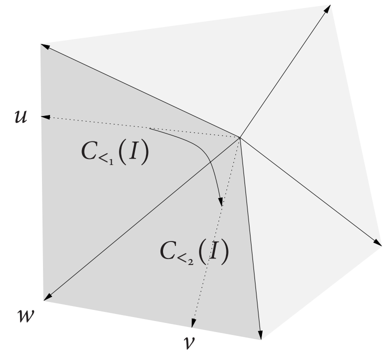

Figure 2.8.2: Crossing the border of two Gröbner cones

(1) The crucial point of GBWalk is the Gröbner basis computation for $\text{in}_w(G_{<_1})$ in Line 4: The assumption is that $\text{in}_w(G_{<_1})$ surely is not a monomial ideal, but quite near to it, this means that most of the generators should have very few monomials. Thus the computation of M_w should be quite fast and lightweight. If this is not the case, then we have a bottleneck. The idea is that if one chooses w rather generic, then it is quite possible that we get a "good" initial ideal $\text{in}_w(G_{<_1})$. There are attempts improving this step by Fukuda et al. ([73]). Other improvements of dynamic Gröbner basis algorithms are ongoing, see, for example, [89].

(2) Note that Algorithm 20 is just a part of the computations that have to be done for computing the requested, reduced Gröbner basis with a Gröbner walk algorithm. We just present how to come from one cone $C_{<_1}(I)$ to the adjacent cone $C_{<_2}(I)$. In a real computation, one has to make several crossings, depending on the starting order $<_1$ and the target order $<_2$. As this is the most difficult part of the algorithm and the other Gröbner basis computations before and after the crossing are clear, we focus on this.

(3) Moreover, Figures 2.8.1 and 2.8.2 should be understood as the easiest possible geometric interpretation of the problem. In general, having more than two variables in your polynomial ring the search for a good path from one order to another can be quite hard. As this not in the focus of this thesis, we remain with the presentation of the basic idea and keep the problems being not directly linked to Gröbner basis computations out of our way.

(4) In [107] Kalkbrener shows that for the conversion of a Gröbner basis $G_{<_1}$ to an adjacent (w.r.t. the Gröbner fan) Gröbner basis $G_{<_2}$ the maximal degree of elements in $G_{<_2}$ is bounded by

$$D(G_{<_2}) < 2 \cdot D(G_{<_1})^2 + (n+1) \cdot D(G_{<_1}).$$

This is a huge improvement to the possible doubly–exponential growth of degree when transforming between two non–adjacent Gröbner bases.

(5) A complete software package dealing with Gröbner cones and Göbner fans is GFAN by Anders Jensen ([105]). The area of fans and cones also has a strong connection to toric and tropical geometry, for example, they are used for the computation of tropical varieties ([27]).

Another method for computing a Gröbner basis $G_{<_1}$ for an ideal I w.r.t. a well–order $<_1$ and transform it to a Gröbner basis $G_{<_2}$ for the same ideal w.r.t. a well–order $<_2$ is the FGLM Algorithm by Faugère, Gianni, Lazard and Mora ([67]). Instead of the attempt of GBWALK, FGLM does not need to pass each adjacent cone $C_<(I)$, but it gives us a direct transformation of $G_{<_1}$ to $G_{<_2}$, regardless whether $C_{<_1}(I) \cap C_{<_2}(I) = \emptyset$ or not.

The main idea of FGLM is to define 3 different sets w.r.t. a leading ideal $L(I)$:

Definition 2.8.12. Let I be an ideal I in $\mathcal{P}$. Then we can define the following sets:

(1) $N(I) := \{m \in \text{Mon}(x_1, \ldots, x_n) \mid m \notin L(I)\}$, the set of *all monomials, which are not reducible by* $L(I)$.

(2) $E(I) := \{m \in L(I) \mid \text{ for all } x_i \text{ such that } x_i \mid m, \frac{m}{x_i} \notin L(I)\}$, the set of *edges of* $L(I)$.

(3) $S(I) := \{m \in L(I) \mid \exists x_i, x_j \text{ such that } x_i \mid m, x_j \mid m, \frac{m}{x_i} \notin L(I), \frac{m}{x_j} \in L(I)\}$, the set of *sides of* $L(I)$.

(4) The disjoint union $B(I) = E(I) \,\dot\cup\, S(I)$ is called the *boundary of* $L(I)$.

Having a Gröbner basis G for I, the sets $N(I)$, $E(I)$ and $S(I)$ can be computed easily. Let us give an example for this.

Example 2.8.13. Assume the ideal $I = \langle x^2y^2 - x^2, x^4 - y^3 \rangle \subset \mathcal{K}[x, y]$. A Gröbner basis for I w.r.t. $<_{\text{dp}}$ is

$$G = \{x^2y^2 - x^2, x^4 - y^3, y^5 - x^4\}.$$

Then we can illustrate $N(G)$, $E(G)$, and $S(G)$ easily in Figure 2.8.13.

Remark 2.8.14. The nice fact is that $N(G) \cap E(G) \cap S(G) = \emptyset$ and any monomial $m \in \text{Mon}(x_1, \ldots, x_n)$, which is not a proper[10] mulitple of an element of $L(G)$, is in exactly one of those three sets. In the following we also talk about elements outside these three sets, thus we introduce the following notation:

$$O(G) := \text{Mon}(x_1, \ldots, x_n) \backslash \big(N(G) \cup E(G) \cup S(G)\big).$$

The idea of FGLM is to compute the three sets N, E, and S to receive the corresponding reduced Gröbner basis.

Proposition 2.8.15. *Let I be an ideal in $\mathcal{P}$, G a Gröbner basis for I w.r.t. $<$. If G is reduced, then* $\langle E(G) \rangle = L(G)$.

[10] In the sense that $\frac{m}{x_i} \notin L(G)$ for all variables x_i.

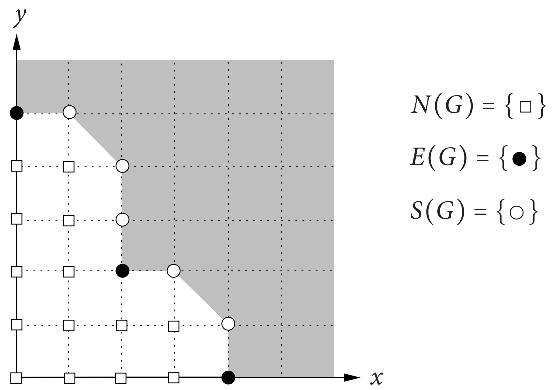

Figure 2.8.3: The classification of $\mathcal{K}[x,y]$ by N, E and S

Proof. This is clear by the definition of $E(G)$. $\square$

For the algorithm presented in the following it is of great importance that the sets $N(G)$ and $E(G)$ are finite. This has consequences on the ideals, for which the reduced Gröbner basis can be computed by FGLM.

Definition 2.8.16. Let I be an ideal in $\mathcal{P}$. We say that *I is zero–dimensional* if and only if the vector space dimension $\dim_{\mathcal{K}}(\mathcal{P}/I) < \infty$.

Proposition 2.8.17. *Let $I = \langle f_1, \ldots, f_r \rangle$ be an ideal generated by polynomials in $\mathcal{P}$. The following statements are equivalent:*

(1) I is zero–dimensional.

(2) $\#\Big(\big\{a \in \mathbb{C}^n \mid f_i(a) = 0 \text{ for } 1 \leq i \leq r\big\}\Big) < \infty.$

(3) For each $i \in \{1, \ldots, n\}$ there exists an element $k_i \in \mathbb{N}\backslash\{0\}$ such that $x_i^{k_i} \in L(I)$.

(4) $\dim_{\mathcal{K}}(\mathcal{P}/I) = \#\big(N(G)\big)$ *for a Gröbner basis G for I w.r.t. <.*

Proof. For example, see Theorem 15 of [159]. $\square$

Proposition 2.8.17 states one big drawback of FGLM, whose pseudo code is presented in Algorithm 21: Since FGLM is constructing the sets N and E successively its termination is based on the fact that $\#(N) < \infty$. Thus it follows that the basic version of the algorithm presented in [67] (and also given in Algorithm 21) can only be applied to zero–dimensional ideals.

Let us describe the functioning of FGLM: Assume an already computed Gröbner basis $G_{<_1}$ for an ideal I w.r.t. $<_1$. Now we switch all computations to be done w.r.t. the desired order $<_2$. We begin to check all possible monomials $m \in \operatorname{Mon}(x_1, \ldots, x_n)$. If $m \mid m'$ and

$m \neq m'$ for some $m' \in E(G_{<_2})$, then $m > m'$ and $m \in S(G_{<_2}) \cup O(G_{<_2})$. Those m are not interesting for us (Line 4). Thus, in Line 5, we can decide whether they are part of $N(G_{<_2})$ or $E(G_{<_2})$. Whenever we find an element for $E(G_{<_2})$ we also compute the corresponding polynomial h, which fulfills $\mathrm{lm}(h) = m$, and add h to $G_{<_2}$. After adding all multiples $x_i m$ to M, we choose the minimal element of M w.r.t. $<_2$ for the next iteration round. In the end, $G_{<_2}$ is the reduced Gröbner basis for I w.r.t. $<_2$, and $E(G_{<_2}) = L_{<_2}(I)$.

Let us have a closer look at how we decide whether m is in $N(G_{<_2})$ or $E(G_{<_2})$: How do we know where to put m by just checking the linear independency in Line 5?

Assume there exist such constants c_λ such that not all $c_\lambda = 0$ and

$$\mathrm{NF}_{\mathrm{red}}(m, G_{<_1}) + \sum_{\lambda \in N} c_\lambda \, \mathrm{NF}_{\mathrm{red}}(p, G_{<_1}) = 0. \tag{2.8.1}$$

Clearly, $h = m + \sum_{\lambda \in N} c_\lambda \lambda$ is an element of I, thus in this situation we also know that there exists an element $g \in G_{<_1}$ with $\mathrm{lm}(g) \mid \mathrm{lm}(h)$, this means that $\mathrm{lm}(h) \in L(I)$. As we compute the monomials m by increasing order, $\mathrm{lm}(h) = m$ and m is added to the set E, which is equal to $L(I)$, when FGLM terminates. On the other hand, if Equation 2.8.1 holds only if $c_\lambda = 0$ for all $\lambda \in N$, then clearly $m \notin L(I)$ and thus m is added to N.

Algorithm 21 Gröbner basis conversion algorithm (FGLM)

Input: $G_{<_1}$ a Gröbner basis for an ideal I in $\mathcal{P}$ w.r.t. $<_1$
Output: $G_{<_2}$ the reduced Gröbner basis for I in $\mathcal{P}$ w.r.t. $<_2$

1: $G_{<_2} \leftarrow \emptyset, E \leftarrow \emptyset, N \leftarrow \emptyset, M \leftarrow \emptyset$
2: $M \leftarrow M \cup \{1\}$
3: **while** $M \neq \emptyset$ **do**
4:    **if** $\left(\nexists m' \in E \text{ such that } m' \mid m \text{ and } m' < m\right)$ **then**
5:       **if** $\left(\exists c_\lambda \in \mathcal{K} : \ \mathrm{NF}_{\mathrm{red}}(m, G_{<_1}) + \sum_{\lambda \in N} c_\lambda \, \mathrm{NF}_{\mathrm{red}}(\lambda, G_{<_1}) = 0 \text{ and not all } c_\lambda = 0\right)$ **then**
6:          $h \leftarrow m + \sum_{\lambda \in N} c_\lambda \lambda$
7:          $G_{<_2} \leftarrow G_{<_2} \cup \{h\}$
8:          $E \leftarrow E \cup \{m\}$
9:       **else**
10:          $N \leftarrow N \cup \{m\}$
11:          $M \leftarrow M \cup \{x_i m \mid 1 \leq i \leq n\}$
12:    $m \leftarrow \min_{<_2}\{m' \in M\}$
13: **return** $G_{<_2}$

All in all, any monomial neither in E nor in N is a proper multiple of an element of E. Thus,

(1) the normal form of an element w.r.t. $G_{<_2}$ is a linear combination of elements of N;

(2) the normal form of an element from $\langle G_{<_1} \rangle$ is zero.

It follows that $G_{<_2}$ is a Gröbner basis for I w.r.t. $<_2$. Since no multiple of elements of E are considered, it is even the corresponding, reduced Gröbner basis.

Besides the above discussion, we do not give a proof of correctness and termination of FGLM as this is not in focus of this thesis and can be found in [67]. From the pseudo code presented it should be clear that termination strongly depends on the fact that N is a finite set. As mentioned before, this restricts the class of considered ideals to zero–dimensional ones.

The transformation process presented in Algorithm 21 is pretty fast, which means that computing a Gröbner basis for I in a "good" well–order $<_1$ and then using FGLM can be much faster and less memory consuming than a direct computation of the Gröbner basis w.r.t. $<_2$.

Some last remarks on FGLM and its impact on the field of computer algebra in the last years.

Remark 2.8.18.

(1) In [67] it is shown that due to the low complexity of the conversion algorithm FGLM the complexity of the computation of a Gröbner basis w.r.t. $<_{\mathrm{lp}}$ can be lowered from $d^{\mathcal{O}(n^3)}$ to $d^{\mathcal{O}(n^2)}$.

(2) In [159] Wichmann generalized the FGLM Algorithm to be useable also if the ideals are not zero–dimensional. For this he uses Hilbert functions to determine various bounds for the computation. The problem with this attempt is that one needs to check the Buchberger Criterion (Theorem 1.8.3) manually to get a criterion for termination. It is clear that this testing leads to a way worse performance than the initial FGLM Algorithm.

(3) In [67] they did not just present the above algorithm, but showed how to reduce the check of linear dependency of polynomials to just linear algebra with vector and matrix computation. Another improvement, which can be understood as an impact for the ideas incorporated in F4 by Faugère (see Section 2.5).

(4) Recently, Faugère and Mou presented new ideas for order–changing Gröbner basis algorithms (again restricted to the zero–dimenstional case) with sparse multiplication matrices in [70].

With this we finish our discussion on order–changing, dynamic, and indirect Gröbner basis algorithms. We have seen that using these attempts one has to deal with the drawback of some restrictions (well–order, zero–dimensional ideals), but one can get a performance–improved way of computing Gröbner bases, where these restrictions are fulfilled anyway. Which approach to be used is highly depending on the initial data, using these techniques without good heuristics can lead to bad results, hence they should be used with care.

2.9 Modular standard basis computations

Coefficient growth during the computation of standard bases over a field of characteristic zero has a very strong influence on the overall computation. In each single reduction step, the leading coefficient c_1 of the reducer p_1 must be adjusted to match the leading coefficient c_2 of the element to be reduced. For this not only the fraction $\frac{c_2}{c_1}$ must be computed, but also every coefficient in p_1 must be multiplied by this fraction. This can lead to enormous numbers, whose calculations slow down the standard basis computation tremendously. In this section we discuss *modular standard basis computations*, influenced by [28, 54] and initially presented 1988 by Traverso in [153] and Winkler in [160].

The idea is to *not* compute one standard basis over a field of characteristic zero, but to compute many standard bases over fields of prime characteristic $p < \infty$. In the end, combined with algorithms for the reconstruction of rational numbers ([45, 110, 136, 157, 158]), we merge these modular standard bases together and lift the coefficients using the Chinese Remainder Theorem (Theorem 1.1.26).

Also the ideas are rather old, these days the method becomes the fashion again due to the development of multicore resp. multiprocessor computers, on which the independent modular computations can be done in parallel ([5, 6, 103]). Even in the area of algebraic cryptanalysis modular Gröbner basis computations are on vogue these days ([106]).

Convention. We are working over the rationals, thus let us assume $\mathcal{P} = \mathbb{Q}[x_1, \ldots, x_n]$ for the whole of this section. As in the previous sections we restrict ourselves to the polynomial case. Moreover, let $<$ be local or global, but not mixed.

For our task to give a description of a modular standard basis algorithm we need to define some more tools in the following.

Definition 2.9.1. Let $N > 0$ be an integer.

(1) The set of *m–Farey fractions* F_m is defined by

$$F_m := \left\{ \frac{a}{b} \mid \gcd(a, b) = 1, 0 \le a \le m, 0 < |\, b \,| \le m \right\}.$$

(2) The *m–Farey rational map* φ_m is defined by

$$\begin{aligned} \varphi_m : F_m &\longrightarrow \mathbb{Z}_p \\ \frac{a}{b} &\longmapsto (a + m\mathbb{Z})(b + m\mathbb{Z})^{-1} \end{aligned}$$

for some prime number p.

Proposition 2.9.2. *Assuming the same notation as in Definition 2.9.1 the m–Farey rational map $\varphi_m : F_m \to \mathbb{Z}_p$ is bijective if and only if m is the largest integer satisfying $m \le \sqrt{\frac{p-1}{2}}$.*

Proof. See [110]. □

Definition 2.9.3. Let $I = \langle f_1, \ldots, f_r \rangle \subset \mathcal{P}$ be an ideal, G a standard basis of I w.r.t. $<$. Moreover, let p be any prime number in $\mathbb{N}$ such that p does not divide the denominator of any coefficient of f_i for $i \in \{1, \ldots, r\}$.

(1) The ideal $I_p = \langle f_1 + p\mathbb{Z}, \dots, f_r + p\mathbb{Z}\rangle \subset \mathbb{Z}_p[x_1, \dots, x_n]$ is the ideal[11] corresponding to I modulo p.

(2) $G_p \subset \mathbb{Z}_p[x_1, \dots, x_n]$ denotes the standard basis for I_p.

(3) A prime number p is called *lucky for I* if and only if $L(G_p) = L(G)$.

(4) A prime number p is called *Hilbert–lucky for I* if and only if $\mathrm{H}_I = \mathrm{H}_{I_p}$.

Lemma 2.9.4. *For any prime p, any ideal I in $\mathcal{P}$ and any degree d it holds that*

$$\mathrm{H}_I(d) \leq \mathrm{H}_{I_p}(d).$$

Proof. See Theorem 5.3 in [6]. □

Now we are ready to describe the workings of modStd (Algorithm 22) in detail. Assume in the following the task to compute a standard basis G for $I = \langle f_1, \dots, f_r\rangle$ in $\mathcal{P}$:

(1) First of all we generate a set Q of prime numbers p which do not divde the denominator of any coefficient of the elements f_i (Line 2).

(2) For each $p \in Q$ we compute the modular standard basis G_p for I_p in $\mathbb{Z}_p[x_1, \dots, x_n]$ (Line 7).

(3) After these modular standard bases are computed and stored in $\mathcal{G}$, we search in algorithm RemoveNotLucky for those G_p whose p are clearly not lucky for I (Line 9). As we do not know G at this point, we cannot use the definition of "luckiness" from 2.9.3. Thus we have to choose the lucky ones with a high probability out of $\mathcal{G}$. For this we build sets S_p in the following way:

Take the first element p of Q. Then we define

$$S_p := \{q \in Q \mid L(G_p) = L(G_q)\}.$$

Next we choose the first element $p' \in Q$ which is not in S_p. We build the set $S_{p'}$ analogously to S_p. This process goes on until all elements of Q are added to exactly one set S_p. Let S be the set containing all these S_p. Then we keep in $\mathcal{G}$ only those standard bases G_p, whose index prime is in the set $S_{p_0} \in S$ where

$$\#(S_{p_0}) \geq \#(S_p) \text{ for all } S_p \in S.$$

With this we get the standard bases corresponding to lucky primes for I with a high probability. A wrong decision here is trapped in the tests in Step (5).

Assume $\mathcal{G} = \{G_{p_1}, \dots, G_{p_s}\}$ after this step.

(4) Then we lift the results in two steps:

a) Using the Chinese Remainder Theorem we get a standard basis G_N in the polynomial ring $\mathbb{Z}_N[x_1, \dots, x_n]$ where $N = \prod_{i=1}^{s} p_i$:

[11] Possibly the f_i are previously multiplied by the least common multiple of all denominators of all coefficients.

$$\begin{array}{ccccccc}\mathbb{Z}_{p_1}[x_1,\dots,x_n] & \times & \dots & \times & \mathbb{Z}_{p_s}[x_1,\dots,x_n] & \longrightarrow & \mathbb{Z}_N[x_1,\dots,x_n] \\ G_{p_1} & \times & \dots & \times & G_{p_s} & \longmapsto & G_N.\end{array}$$

b) From $\mathbb{Z}_N[x_1,\dots,x_n]$ we get back to $\mathcal{P}$ using the *Farey rational map* φ_k, where $k \leq \sqrt{\frac{N-1}{2}}$.

These computations are done in the algorithm Lift (Line 10) which returns the set G.

(5) Next we need to check if G really is a standard basis of I w.r.t. $<$. This has to be done, since we do not know whether we have computed enough modular standard bases G_p or not.

Of course, there exists an upper bound for the number of primes to be considered: Assume that the Gröbner basis G for I w.r.t. $<$ would have been already computed beforehand. Then the primes $p \in Q$ would be enough if

$$\prod_{p\in Q} p \geq \max\left\{2\cdot \mid c \mid^2 \mid c \text{ any coefficient of an element } g \in G\right\}.$$

Sadly we do not know G beforehand, as our task plainly is to compute G. Thus we do not know, at which point of the computations we have enough modular standard bases computed and lifted. Thus we need to test if the set G constructed in Step (4) is the requested standard basis or not. For this, $G = \{g_1,\dots,g_t\}$ must pass 3 different tests in the algorithm Test (Line 11):

a) We choose some prime q randomly such that q does not divide the numerator or denominator of any coefficient of the generating polynomials f_i for I such that $q \notin Q$. The test is passed if $\{g_1 + q\mathbb{Z},\dots,g_t + q\mathbb{Z}\}$ is a standard basis for I_q. Note that this test is not sufficient for checking if G is a standard basis of I, but it is very fast compared to the following two necessary tests. If G does not pass this test, we can go on with more modular computations, without the need of losing too much time doing the next two, very expensive tests.

b) Next we check if $I \subset \langle G\rangle$.

c) Last we check if G is a standard basis for $\langle G\rangle$. Note that this test is done in $\mathcal{P}$ and it can be very expensive to test this if we have not considered enough modular standard bases G_p.

(6) If G passes Test, then G is the standard basis for I w.r.t. $<$ and modStd terminates. Otherwise, we need to consider more primes and compute more modular standard bases. We are back at Step (2).

Remark 2.9.5.

(1) Of course, the pseudo code presented in Algorithm 22 is not optimized, but focusses on the general idea of modStd. In a real implementation one re–uses the already computed standard bases G_p and the standard basis G_N, already lifted by the Chinese Remainder Theorem. Thus, if Test does not return a positive answer, in the next round of modular computations we do compute only those G_q where $q \in R$.

Algorithm 22 Modular standard basis computation (modStd)

Input: I an ideal in $\mathcal{P}$, $<$ an order on $\mathcal{P}$
Output: G a standard basis for I in $\mathcal{P}$ w.r.t. $<$

$\mathcal{G} \leftarrow \emptyset$, $b \leftarrow 1$
$Q \leftarrow \{p$ prime numbers $\mid p$ chosen heuristically $\}$
while $(b = 1)$ **do**
  while $(Q \neq \emptyset)$ **do**
    Choose p from Q.
    $Q \leftarrow Q \setminus \{p\}$
    $G_p \leftarrow \textsc{Std}(I_p, \mathrm{NF})$
    $\mathcal{G} \leftarrow \mathcal{G} \cup \{G_p\}$
  $\textsc{RemoveNotLucky}(\mathcal{G})$
  $G \leftarrow \textsc{Lift}(\mathcal{G})$
  if $\big(\textsc{Test}(G, I, Q)\big)$ **then**
    return G
  $R \leftarrow \{p$ prime numbers $\mid p \notin Q$ and p chosen heuristically $\}$
  $Q \leftarrow Q \cup R$

(2) A highly optimized version of the presented variant of modStd is implemented in Singular by Hashemi, Pfister, Schönemann and Steidel in the library `modstd.lib`. This implementation provides also the possibility to do computations in parallel. See below for more information on this.

(3) Not so long ago modStd was restricted to either homogeneous input or to a local order on $\mathcal{P}$. Recently Idrees, Pfister, and Steidel have proven in [103] the correctness and termination of modStd for the inhomogeneous case also for global orders.

(4) Note that if one does not test the computed set G to be a standard basis for $\langle G \rangle$ over the rationals, G must not be a standard basis for I w.r.t. $<$, but it is with a high probability. In some applications this probabilistic answer is sufficient, but a modular standard basis computation without tests at the end cannot guarantee that its result is the requested standard basis.

(5) If the inital problem is given in $\mathcal{P}$ equipped with a mixed order $<$ one could homogenize the ideal and compute the homogenized standard basis G^{h} in $U^{-1}\mathcal{K}[x_0, \ldots, x_n]$ w.r.t. $<_{\mathrm{h}}$ (see Section 1.5 for more information about the connection between $<$ and the homogenized order $<_{\mathrm{h}}$). Afterwards, a dehomogenization of G^{h} results in the requested standard basis G. But note that the computation of G^{h} can be much harder than the one in the inhomogeneous case.

Having understood the modular computations the idea of *parallelizing modStd* is quite easy.

The following steps of modStd can be parallelized easily:

(1) the modular standard basis computations, and

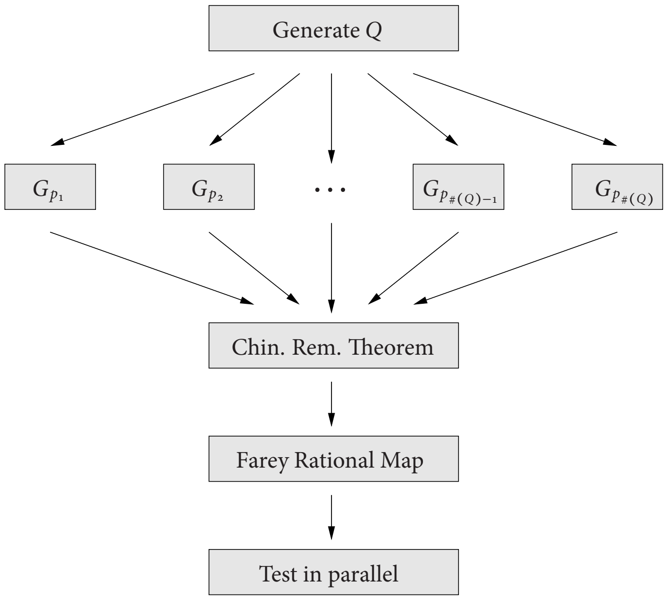

Figure 2.9.1: Parallelized modStd

(2) the tests:

a) Test whether $G_q := \{g + q\mathbb{Z} \mid g \in G\}$ is a standard basis for I_q for some prime $q \notin Q$: For this, show that

$$f_i + q\mathbb{Z} \in \langle G_q \rangle \text{ and } G_q \subseteq \textsc{Std}(I_p, \mathrm{NF}),$$

b) Test whether $I \subseteq \langle G \rangle$ or not:

$$I \subseteq \langle G \rangle \Longleftrightarrow f_i \in \langle G \rangle \text{ for all } f_i \text{ generating } I,$$

c) Test whether G is a standard basis for $\langle G \rangle$ or not: Check, if all s–polynomials, not detected by the Buchberger criteria (see Section 2.3), reduce to zero w.r.t. G.

Of course, this parallelization pattern is based on the fact that all parallelized computations are done in a similar timespan, e.g. the timings of the computations of G_{p_1} and G_{p_2} should not differ widely. The very same holds for the inclusion checks for the generators of I and the s–polynomials.

Remark 2.9.6.

(1) To keep Figure 2.9.1 readable, we abandon to illustrate the parallelization of the tests in detail.

(2) The process of parallelizing the test should be clear, but from the implementational point of view, a significant distinction has to be done: Whereas the modular standard basis computations of the G_p can be parallelized easily using one process per computation, one needs to use multiple threads doing the parallelized tests. Otherwise the overhead of sending and receiving data from one process to the other takes longer than the complete reduction itself.

(3) Let us clarify that the above presented attempt to parallelize MODSTD is just a first approach, but it is still too static and based on the ideas of sequential computations. The SINGULAR team is recently optimizing the implementation of MODSTD, including many ideas to make the distributed computations more dynamic, even on different computers in connected networks. For example, depending on the relation between the still–to–be–computed modular standard bases and the number of cpu cores resp. processors available the Chinese Remainder Theorem can be used to lift already computed modular bases meanwhile others are still computed concurrently. It is nearly not possible to utilize multicore resp. multiprocessor computers to the full by just parallelizing parts of known sequential algorithms. New ideas leading to new concepts of algorithms must be developed and implemented for this task.

In 1988, Traverso presented the so–called *Gröbner trace algorithm* ([153]). On the one hand, his algorithm is in some sense the origin of the already presented MODSTD, as there the idea of lucky prime numbers and modular attempts are noted first in a connection with Gröbner basis computations. On the other hand, the Gröbner trace algorithm is a much more aggressive realization of those ideas.

Whereas MODSTD uses only the idea of finding lucky primes p modulo whose the standard basis computations are done independently, the Gröbner trace algorithm enforces upon all modular computations the same setting, the *trace*, and worms them in a tight corset.

Definition 2.9.7. Let $I = \langle f_1, \ldots, f_r \rangle$ be an ideal in $\mathcal{P}$, $<$ a monomial order on $\mathcal{P}$. When we are computing a Gröbner basis $G = \{g_1, \ldots, g_s\}$ for I w.r.t. $<$ with $f_i = g_i$ for $i \in \{1, \ldots, r\}$, we define the *Gröbner trace* $T(t, \mathcal{S}, n, \lambda)$ where

(1) m is a finite sequence of the leading monomials of G: $m = (m_1, \ldots, m_s)$,

(2) $\mathcal{S}$ is a finite sequence of all generated critical pairs: $\mathcal{S} = (\mathcal{S}_{r+1}, \ldots, \mathcal{S}_s)$,

(3) n is a finite sequence of finite sequences of integers $n_{j,k}$: $n = (n_{r+1}, \ldots, n_s)$ such that $n_j = (n_{j,1}, \ldots, n_{j,k_j})$ where $n_{j,k} < j$ for all $j \in \{r+1, \ldots, s\}$, and

(4) λ is a finite sequence of finite sequences of terms $\lambda_{j,k}$: $\lambda = (\lambda_{r+1}, \ldots, \lambda_s)$ such that $\lambda_j = (\lambda_{j,1}, \ldots, \lambda_{j,k_j})$ for all $j \in \{r+1, \ldots, s\}$.

So what is the deal? During the computation of the Gröbner basis G we store all essential information in the Gröbner trace:

(1) We store for each element in G its leading monomial in m.

(2) Each computed s–polynomial is stored in $\mathcal{S}$.

(3) Every reduction step is uniquely determined by each entry in n and λ: $n_{j,k}$ is the index of the reducer in G for the k–th reduction of the the j–th element. $\lambda_{j,k}$ is the corresponding multiplier for this reduction step.

So in the end we know each reduction step of each s–polynomial.

With this we are ready to present the *Gröbner trace reconstruction algorithm*: Assume that we want to compute a Gröbner basis G for an ideal I w.r.t. $<$. Moreover, assume that we have already given a Gröbner trace T, for example by another Gröbner basis computation for I[12]. Then we can use Algorithm 23 for the computation of G:

Algorithm 23 Gröbner trace reconstruction algorithm (GBTrace)

Input: $I = \langle f_1, \ldots, f_r \rangle$ an ideal in $\mathcal{P}$, $T = (m, \mathcal{S}, n, \lambda)$ a Gröbner trace for I
Output: G a set of polynomials including $\{f_1, \ldots, f_r\}$, R, E integer values

1: $R \leftarrow 0, E \leftarrow 0$
2: $G \leftarrow \{f_1, \ldots, f_r\}$
3: **for** $(i = r+1, \ldots, s)$ **do**
4: $\quad h \leftarrow \mathcal{S}_i$
5: $\quad$ **for** $(j = 1, \ldots, j_i)$ **do**
6: $\quad\quad$ **if** $\left(\operatorname{lm}(f) = \lambda_{i,j} \operatorname{lm}(g_{n_{i,j}})\right)$ **then**
7: $\quad\quad\quad f \leftarrow \operatorname{lc}(g_{n_{i,j}}) f - \operatorname{lc}(f) \lambda_{i,j} g_{n_{i,j}}$
8: $\quad\quad$ **else if** $\left(\operatorname{lm}(f) < \lambda_{i,j} \operatorname{lm}(g_{n_{i,j}})\right)$ **then**
9: $\quad\quad\quad R \leftarrow 1$
10: $\quad\quad$ **else**
11: $\quad\quad\quad E \leftarrow 1$
12: $\quad\quad\quad$ **return** (G, R, E)
13: $\quad$ **if** $\left(\operatorname{lm}(f) = m_i\right)$ **then**
14: $\quad\quad g_i \leftarrow f$
15: $\quad\quad G \leftarrow G \cup \{g_i\}$
16: $\quad$ **else if** $\left(\operatorname{lm}(f) > m_i\right)$ **then**
17: $\quad\quad E \leftarrow 1$
18: $\quad$ **else**
19: $\quad\quad E \leftarrow 2$
20: $\quad\quad$ **return** (G, R, E)
21: **return** (G, R, E)

Let us have a closer look at the pseudo code:
A set G is computed using the Gröbner trace T. The important point is that we do not really compute anything besides some reduction steps, everything else is predefined by the Gröbner trace:

(1) In Line 4 we choose the s–polynomials from T.

(2) In Line 6 we choose the corresponding reducers already stored in T by $n_{i,j}$ and $\lambda_{i,j}$.

[12] For example, think of different modular computations as in modStd.

(3) In Line 15 we do not compute new critical pairs, since the whole set of s–polynomials to be investigated is already given by T in $\mathcal{S}$.

This has *several advantages* to a usual Gröbner basis computation: We do not need to search for any element, we do not need to generate for the multipliers of the reducers, and we do not need to check new critical pairs by criteria. One of the most important optimizations is that, besides some coefficient size and polynomial length differences, we know quite accurately the memory consumption of the computation.

Of course, Algorithm 23 pays dearly for this simplification with a static behaviour, which cannot react on changes resp. unforeseen steps as dynamically as STD can do. Thus we need to add two boolean variables R and E, which keep track of problems happening during the computations of GBTRACE:

(1) R is set to 1 if a *redundancy* has happened. This means that at some reduction step the leading term of f is lower than expected (Line 8). At this point we do not need to interrupt the computations, it is possible that $\mathrm{lt}(f)$ is equal to the leading term of the next reducer/multiplier pair stored in T. So even if a redundancy takes place, $\mathrm{lt}(g_i) = t_i$ can still be fulfilled in Line 13.

(2) E is set to 1 or 2 if an *error* has happened. This can happen at exactly two points of Algorithm 23:

   a) If $\mathrm{lm}(f) > \lambda_{i,j}\,\mathrm{lm}(g_{n_{i,j}})$, then the computation cannot go on from this point (Line 10). All following reducers, generated by the lists n_i and λ_i in the Gröbner trace T, have a leading monomial smaller than $\lambda_{i,j}\,\mathrm{lm}(g_{n_{i,j}})$, thus no further reduction for f takes place. At this point the algorithm returns the already computed set G and marks the error with $E = 1$.

   b) If in Line 16 $\mathrm{lm}(f) \neq t_i$, then the last reduction step went wrong. In this situation the algorithm must terminate with an error, too, as the following s–polynomials in $\mathcal{S} \in T$ would be no longer valid. Here we distinguish between two possible errors: If $\mathrm{lm}(f) > m_i$ then $E = 1$, otherwise $E = 2$. The reason why we need to distinguish these situations is explained in the discussion for TRACEMODSTD below.

Remark 2.9.8.

(1) How to handle errors in Algorithm 23 is not obvious. Thinking about using GBTRACE during a modular Gröbner basis computation the effects of one such error on the other computations must be handled with great care: Shall all computations stop? Shall we just kill this one computation and go on with the next prime number? Here we need again a good heuristic, but mostly one would perhaps decide to just kill the defective computation and to go on, awaiting not so many errors to follow.

(2) Be cautious that GBTRACE does not claim to return a Gröbner basis for I. If the algorithm terminates without an error it is highly probable that G is a Gröbner basis for I w.r.t. $<$, but it is not ensured. Thus, whenever using GBTRACE in a Gröbner basis computation we need to add tests at the end, similar to those in MODSTD.

Traverso gives different approaches of how to use GBTrace in Gröbner basis computations in [153], we restrict ourselves to the one most obvious, the *modular Gröbner trace computation*.

Algorithm 24 Modular Gröbner trace algorithm (TraceModStd)

Input: I an ideal in $\mathcal{P}$, $<$ an order on $\mathcal{P}$
Output: G a Gröbner basis for I w.r.t. $<$

1: $\mathcal{G} \leftarrow \emptyset$, $R \leftarrow 0$, $E \leftarrow 0$, $b \leftarrow 1$
2: $Q \leftarrow \{p$ prime numbers $\mid p$ chosen heuristically $\}$
3: Choose p_0 from Q.
4: $Q \leftarrow Q \setminus \{p_0\}$
5: $(G_{p_0}, T) \leftarrow \text{tStd}(I_{p_0}, \text{NF})$
6: $\mathcal{G} \leftarrow \mathcal{G} \cup \{G_{p_0}\}$
7: **while** $(b = 1)$ **do**
8:     **while** $(Q \neq \emptyset)$ **do**
9:         Choose p from Q.
10:         $Q \leftarrow Q \setminus \{p\}$
11:         $(G_p, R, E) \leftarrow \text{GBTrace}(I_p, T)$
12:         **if** $(E = 0)$ **then**
13:             $\mathcal{G} \leftarrow \mathcal{G} \cup \{G_p\}$
14:         **else if** $(E = 1)$ **then**
15:             $\mathcal{G} \leftarrow \emptyset$
16:             $(G_p, T) \leftarrow \text{tStd}(I_p, \text{NF})$
17:     RemoveNotLucky$(\mathcal{G})$
18:     $G \leftarrow \text{Lift}(\mathcal{G})$
19:     **if** $\big(\text{Test}(G, I, Q)\big)$ **then**
20:         **return** G
21:     $R \leftarrow \{p$ prime numbers $\mid p \notin Q$ and p chosen heuristically $\}$
22:     $Q \leftarrow Q \cup R$

The whole "modular wrapper" in Algorithm 24 should be clear, it is very similar to modStd. The main differences are:

(1) In Line 5 a first modular Gröbner basis is done modulo the prime number p_0. There, tStd denotes any standard basis algorithm equipped with the feature that it also stores all neccessary data for the corresponding Gröbner trace T.

(2) Next the other modular computations are performed (Line 11), but this time no standard basis computation is done. We use GBTrace to compute the corresponding sets G_p. This speeds–up the computations tremendously.

(3) If an error is reported from GBTrace we must have a closer look:

   a) If $E = 1$ (Line 14), then at some point the leading monomial of an element computed for G_p is greater than the corresponding one stored in T. At this point we must assume that p is lucky and all beforehand used primes were not lucky. Thus we delete all previously computed Gröbner basis from $\mathcal{G}$ (Line 15) and compute a new Gröbner trace using the (hopefully lucky prime number p).

b) If $E = 2$ we just discard the computed modular Gröbner basis G_p and go on with the computations. We can hope that the previous prime numbers are lucky and p is not lucky.

This is exactly the reason, why we have to distinguish the different types of errors in GBTrace.

Clearly, if no error is reported, we add G_p to $\mathcal{G}$ and go on with the next prime number.

Let us close this topic with two remarks on implementational aspects considering the Gröbner trace.

Remark 2.9.9.

(1) As already noted in Remark 2.9.5 the pseudo code of Algorithm 24 is given focussing on comprehension, not on efficiency. It is clear that one has to think about how to recover T in Line 16 possibly without a complete Gröbner basis computation. Moreover, Traverso gives other possible implementations using different computation–test–balances and error handling.

(2) An attempt of using the ideas of tracing together with the improved reduction process of F4 are given in [106]. There it is used for algebraic attacks on cryptosystems. In this setting one needs to compute Gröbner bases of polynomial systems having the same shape, differing only in coefficients which are either random or depend on a small number of parameters.

This finishes our discussion about modular standard basis algorithms. One should keep in mind that there is a lot of space optimizing the parallel attempt, explicitly the balance between computing and testing shall be investigated in more detail to receive a better performance.

2.10 Involutive bases

As a last, but quite different attempt to improve standard basis computations, we give a short overview of involutive methods. The main idea is to define an *involutive monomial division* and to show the correspondences to the usual division. Using this fact, *involutive normal forms* can be defined. With these so–called *involutive bases* can be computed, which fulfill the Buchberger Criterion (Theorem 1.8.3). Thus any such involutive basis is a standard basis, too.

Note that we introduce the notion of involutive bases only over the polynomial ring $\mathcal{P}$ equipped with a well–order $<$. This is, again, due to the fact that the involutive approach is not in the focus of this thesis and we want to keep notation as simple as possible.

This topic is discussed in depth in various publications, for example, [11,24,25,40,82,84].

Definition 2.10.1. Let $u, v, w \in \mathcal{P}$ be monomials. We define the *involutive (monomial) division* by the relation $|_I$, which has the following properties:

(1) $u \mid_I v \Rightarrow u \mid v$.

(2) $u \mid_I u$ for all monomials $u \in \mathcal{P}$.

(3) $u \mid_I uv$ and $u \mid_I uw \Leftrightarrow u \mid_I uvw$.

(4) If $u \mid_I w$ and $v \mid_I w$, then $u \mid_I v$ or $v \mid_I u$.

(5) If $u \mid_I v$ and $v \mid_I w$, then $u \mid_I w$.

Remark 2.10.2. Note that the usual monomial division satisfies Property (4) only in the univariate case, i.e. if $\mathcal{P} = \mathcal{K}[x]$. For example, assume $\mathcal{P} = \mathcal{K}[x, y]$. Then

$$x \mid xy \text{ and } y \mid xy \text{ but } x \nmid y \text{ and } y \nmid x.$$

Important examples of involutive divisions are the Janet division ([104]), the Pommaret division ([139]), and the Thomas division ([151]).

Definition 2.10.3.

(1) A subset M of the set of all monomials in $\mathcal{P}$ is called *involutive* if for any element $u \in M$ it holds that

$$\bigcup_{u \in M} \{um \mid m \text{ any monomial in } \mathcal{P}\} = \bigcup_{u \in M} \{v \in M \mid u \mid_I v\}.$$

(2) A subset M of the set of all monomials in $\mathcal{P}$ is called *involutively autoreduced* if for any two elements $u, v \in M$ it holds that

$$u \nmid_I v \text{ and } v \nmid_I u.$$

(3) A finite set F in $\mathcal{P}$ is called *involutively autoreduced* if $\operatorname{lt}(F)$, the set of the leading terms of elements in F, is involutively autoreduced and no $f = \sum_\alpha c_\alpha \boldsymbol{x}^\alpha \in F$ has a term $c_{\alpha_0} \boldsymbol{x}^{\alpha_0} = c_{\alpha_0} u \neq \operatorname{lt}(f)$ where

$$u \in \bigcup_{t \in \operatorname{lt}(F)} \{v \in \mathcal{P} \text{ monomial } \mid t \mid_I v\}.$$

Using the relation $|_I$ one can introduce the notion of an *involutive normal form* $\eta_I(p, G)$ which corresponds to the usual normal form (see Definition 1.7.12). Altogether, the main object to study in this area can be defined:

Definition 2.10.4. A finite set $G = \{g_1, \ldots, g_s\}$ in $\mathcal{P}$ is called an *involutive basis* for the ideal $\langle g_1, \ldots, g_r \rangle$, $r \leq s$, if

(1) G is an involutively autorreduced set, and

(2) for all $g_i \in G$ and all monomials $u \in \mathcal{P}$ it holds that $\eta_I(ug_i, G) = 0$.

Theorem 2.10.5. *If G is an involute set, then for all $p \in \mathcal{P}$ it holds that*

$$\eta(p, G) = \eta_I(p, G).$$

Using this equivalence we can easily follow:

Corollary 2.10.6. *Let G be a finite set in $\mathcal{P}$. If G is an involutive basis for $\langle G\rangle$, then G is a standard basis for $\langle G\rangle$.*

The proof of Theorem 2.10.5 and a more extensive introduction on this topic can be found in [24].

Over the last couple of years, the ideas of how to parallelize the computation of involutive bases are developed (see [84]), a variant of FGLM has been implemented (see [83]), and lots of other improvements in this area of computational algebra have been made.

2.11 Concluding remarks

In this chapter we have presented a wide range of improvements or variants of Std, using *classical methods*. Some of them use the Hilbert–Poincaré series, others different orders on the sets of critical pairs, still others transform the normal form computations to matrix operations. For most of these approaches we have seen lots of benefits, but often drawbacks, too. For example, one has to consider restrictions on the input and the efficiency of the methods is highly dependent of the behaviour of the data during the computations, which cannot be known beforehand. Thus there is not the one and only best way to compute standard bases. To get a standard basis in an efficient way, one needs to implement and combine most of the presented ideas, bind together by well–elaborated heuristics.

In the same way the *signature–based computations* we present in the following are not the utility knife of standard basis algorithms. On the one hand we see that in most situations they find more useless critical pairs than Buchberger's Criteria. In some situations, examples beforehand intractable, even with the improvements of this chapter, can be solved using the signature–based approach. On the other hand we get some restrictions on the reduction process and overhead is generated due to how aggressive the signature–based criteria are chosen. Thus it is not the question of getting the one best algorithm, but even more about how to combine the signature–based world with already highly efficient improvements of the classic world.

Clearly, ideas like the Gröbner walk, the FGLM transformation or modular approaches can be used straightforwardly with signature–based algorithms. Those improvements can be understood as *wrappers* around random standard basis algorithms, thus we only have to guarantee that the result is a standard basis with the requested properties, for example, being reduced. Other improvements are not as easy to apply to signature–based algorithms, some of them even cannot be combined or harm performance by interfering with signature–based criteria.

With these considerations in mind the motivation for our research on signature–based ideas is clear:

(1) Improve timings, memory usage, and performance of already efficient and improved computations.

(2) Combine new ideas with as many as possible improvements presented in this chapter.

(3) Try to merge ideas of both worlds to gain an even better insight into the underlying theory, which could lead to more improvements in the future.

3 Syzygy modules and standard bases

This chapter can be somewhat understood as *connecting link* between everything already stated in chapters 1 and 2, and the signature–based attempt, whose introduction follows in Chapter 4.

Facing all the discussions and problems understanding signature–based standard basis algorithms, in particular Faugère's F5 Algorithm, which started this field of research, this chapter can be also seen as a *missing link*. Starting the discussion of signature–based algorithms with this interlude makes it a lot easier for us to understand the way things work there. Moreover, some disparities to ideas presented in Chapter 2 appear here for the first time, e. g. the restriction of reducers in Algorithm 30, which uses so–called syzygies to improve the computation of standard bases.

So what are these *syzygies*? Loosely speaking they can be understood as relations between elements. Having given a finite set $F = \{f_1, \ldots, f_r\} \in \mathcal{P}^k$ the question arises if there are any dependencies and connections between the different f_is. Moreover, the nice fact is that these syzygies again build a module in some $\mathcal{P}^l$. A generalization of a syzygy module is the so–called *free resolution*. It stores a lot of data about the structure of F and is useful, and even essential, for lots of applications in algebraic geometry. Furthermore, syzygies are very useful in theory, for example one can give quite a nice proof of Buchberger's Criterion (Theorem 1.8.3) using them.

Clearly, computing these patterns is a lot more difficult than to compute a standard basis of F. Otherwise, computing a standard basis of F can help to compute the syzygy module of F. Even more astonishing is the fact that intermediately computed syzygies can improve the performance of standard basis algorithms deeply.

In Section 3.1 we introduce the notion of a *staggered linear basis*, which can be also used to improve standard basis computations. We see that this points directly to syzygies and their computations, which are covered in Section 3.2. In Section 3.3 we show how syzygies can be exploited to give new criteria for the detection of useless data in a standard basis computation. This is exactly the point at which the signature–based world in computer algebra starts.

3.1 Staggered linear bases

In 1986 Gebauer and Möller presented a new idea for detecting useless critical pairs in a Gröbner basis computation ([80]). For this they introduced a new kind of basis, the so–called *staggered linear basis*. Later on, due to some problems with the initial attempt, Mora has presented a revised version of their idea in [131]. In 2009, based on the previous work, Dellaca has outlined both attempts, has revised them, and has shown their respective advantages / problems ([52]).

We see in the following that the idea of staggered linear bases can be seen as an initial spark for the development of signature–based standard basis algorithms.

In this chapter we again restrict ourselves to the polynomial case, and always assume $\mathcal{P}$ to be equipped with a well–order $<$.

Definition 3.1.1. Let I be an ideal in $\mathcal{P}$. A *Gauss generating set B for I* is defined by the following properties:

(1) $B \subset I$,

(2) $B = \operatorname{span}_{\mathcal{K}}(I)$.

Moreover, if B also fulfills

(3) $\operatorname{lm}(f) = \operatorname{lm}(g) \Rightarrow f = g$ for all $f, g \in B$,

then B is called a *Gauss basis for I*.

In the following we characterize Gauss bases, and derive from this discussion easily the connection between Gauss bases and Gröbner bases.

Lemma 3.1.2. *Let I be an ideal in $\mathcal{P}$, and let $B = \{g_1, \ldots, g_s\} \subset I$ finite such that $B = \operatorname{span}_{\mathcal{K}}(I)$. Then B is a Gauss basis for I if and only if for each $f \in I$ there exists a representation*[1]

$$f = \sum_{i=1}^{s} c_i g_i, \text{ such that } c_i \in \mathcal{K}, \operatorname{lm}(f) \geq \operatorname{lm}(g_i) \text{ for all } i.$$

Proof. See Proposition 3.4 in [52]. □

Corollary 3.1.3 (Lemma 22.2.2 in [131])**.** *Let I be an ideal in $\mathcal{P}$, and let G be a finite subset of I. Then the following conditions are equivalent:*

(1) G is a Gröbner basis for I.

(2) $B := \{mg_i \mid g_i \in G, m \in \operatorname{Mon}(x_1, \ldots, x_n)\}$ is a Gauss basis for I.

Proof. This follows easily from the fact that $L(B) = L(G)$. □

As a consequence of Corollary 3.1.3 one can easily construct a Gauss basis, whenever a corresponding Gröbner basis is already given. The other way around is more interesting for us: When we have computed a Gauss basis for an ideal I, can we construct a Gröbner basis for I out of it? The answer to this question lies in the process of "staggering" the Gauss basis.

Definition 3.1.4. Let I be an ideal in $\mathcal{P}$. Then the set

$$S := \bigcup_{i=1}^{s} \{(g_i, M_i) \mid M_i \subset \operatorname{Mon}(x_1, \ldots, x_n)\}$$

is called a *staggered linear basis for I* if

$$B_S := \bigcup_{i=1}^{s} \{mg_i \mid m \in \operatorname{Mon}(x_1, \ldots, x_n) \backslash \langle M_i \rangle\}$$

is a Gauss basis for I.

The idea is to have a Gauss basis with different levels at which the monomials multiplied to the generating polynomials are restricted. These restrictions, represented by M_i, are local to every generator g_i.

With this definition we get a practical solution for constructing Gröbner bases out of staggered linear bases.

Theorem 3.1.5. *With the staggered linear basis S for I as in Definition 3.1.4 and the corresponding Gauss basis*

$$B_S := \bigcup_{i=1}^{s} \{mg_i \mid m \in \operatorname{Mon}(x_1, \ldots, x_n) \backslash \langle M_i \rangle\}$$

[1]Such a representation is sometimes called a Gauss representation.

the set $G \subset I$ defined by

$$G := \bigcup_{i=1}^{s} \{ g_i \mid \mathrm{lm}(g_j) \nmid \mathrm{lm}(g_i) \textit{ for all } j < i \}$$

is a Gröbner basis G for I.

Proof. See Lemma 25.4.4 of [131]. □

Let us give a small example showing how all this different kinds of bases are related to each other. Moreover, this example outlines the ideas behind the staggered linear basis algorithms presented in the following.

Example 3.1.6. Let $I = \langle f_1, f_2 \rangle \subset \mathcal{K}[x, y, z]$, using $<_{\mathrm{dp}}$, where $f_1 = x^2 y - z^2$, $f_2 = yz^4 - x^3$. We can easily define a Gauss generating set, namely

$$B := \{ m f_1 \mid m \in \mathrm{Mon}(x_1, \ldots, x_n) \} \cup \{ m f_2 \mid m \in \mathrm{Mon}(x_1, \ldots, x_n) \}.$$

Clearly, B is not a Gauss basis for not fulfilling the third property of Definition 3.1.1:

$$z^4 \,\mathrm{lm}(f_1) = x^2 \,\mathrm{lm}(f_2), \text{ but } z^4 f_1 \neq x^2 f_2.$$

Next one could think about including the idea of staggered linear bases, i.e. restricting the possible multiples of $\mathrm{lm}(f_2)$. Thus we would set $M_2 = \{x^2\}$ since this is the first multiplier for which the multiplied leading monomials of f_1 and f_2 interfere. Hence we could construct the set

$$B' := \{ m f_1 \mid m \in \mathrm{Mon}(x_1, \ldots, x_n) \} \cup \{ m f_2 \mid m \in \mathrm{Mon}(x_1, \ldots, x_n) \backslash \langle M_2 \rangle \}.$$

The problem with this approach is that B' is no longer a Gauss generating set since $B \neq \mathrm{span}_{\mathcal{K}}(I)$: For example, there is no representation for $x^2 f_2$ with elements in B'. Accordingly we need to add some element to B'. Since $y^4 \,\mathrm{lm}(f_1)$ is already represented by $y^4 f_1$, the element f_3 we need to add must fulfill $\mathrm{lm}(f_3) < \mathrm{lm}(f_1)$. In particular, f_3 must fulfill the following equation:

$$\begin{aligned} x^2 f_2 &= y^4 f_1 + m_1 f_1 + m_2 f_2 + f_3 \\ x^2 f_2 - y^4 f_1 - \sum_{i=1}^{2} m_i f_i &= f_3 \\ \mathcal{S}(f_2, f_1) - \sum_{i=1}^{2} m_i f_i &= f_3. \end{aligned}$$

In other words, we need to compute the s–polynomial of f_1 and f_2 and compute its normal form w.r.t. $\{f_1, f_2\}$. It follows that $\mathcal{S}(f_2, f_1)$ does not reduce to zero, but results in a new polynomial f_3 which must be added to B'. Performing the computations we end up with $f_3 := z^6 - x^5$. Adding f_3 to B' we receive a Gauss generating set

$$\begin{aligned} B' := &\{ m f_1 \mid m \in \mathrm{Mon}(x_1, \ldots, x_n) \} \cup \{ m f_2 \mid m \in \mathrm{Mon}(x_1, \ldots, x_n) \backslash \langle M_2 \rangle \} \\ &\cup \{ m f_3 \mid m \in \mathrm{Mon}(x_1, \ldots, x_n) \}. \end{aligned}$$

We are still not finished, since one sees easily that

$$z^6 \operatorname{lm}(f_1) = x^2 y \operatorname{lm}(f_3) \text{ as well as } z^6 \operatorname{lm}(f_2) = yz^4 \operatorname{lm}(f_3).$$

To get a Gauss basis B'' for I we need to restrict some of these polynomial multiples to ensure uniqueness of leading monomial and corresponding polynomial in B''. Here we have a choice: On the one hand we can restrict M_1 and M_2, and on the other hand we can restrict M_3. We decide us to go the easier way, namely restricting only the multiples of f_3:

$$\begin{aligned}
M_3 &:= \left\{ \operatorname{lm}(f_1), \operatorname{lm}(f_2) \right\}, \\
B'' &:= \left\{ m f_1 \mid m \in \operatorname{Mon}(x_1, \ldots, x_n) \right\} \cup \left\{ m f_2 \mid m \in \operatorname{Mon}(x_1, \ldots, x_n) \backslash \langle M_2 \rangle \right\} \\
&\quad \cup \left\{ m f_3 \mid m \in \operatorname{Mon}(x_1, \ldots, x_n) \backslash \langle M_3 \rangle \right\}.
\end{aligned}$$

With this B'' is a staggered linear basis for I, since one clearly sees that $G = \{f_1, f_2, f_3\}$ is a Gröbner basis for I (from the above discussion it follows that $\mathcal{S}(f_3, f_1)$ as well as $\mathcal{S}(f_3, f_2)$ are detected by Buchberger's 1st Criterion).

Accordingly, we need an algorithm to compute a staggered linear basis in the vein of the construction presented in Example 3.1.6. This algorithm was given initially by Gebauer and Möller in [80].

Let us discuss the corresponding pseudo code given in Algorithm 25 in more detail:

The first main difference to already known standard basis algorithms, for example use GM for comparisons, can be found in the lines 1 – 3. There the initial values of the sets M_i for each f_i of the input F are computed:

$$\begin{aligned}
M_1 &= \varnothing, \\
M_2 &= \left\{ \operatorname{lm}(f_1) \right\}, \\
&\ \ \vdots \\
M_r &= \left\{ \operatorname{lm}(f_1), \ldots, \operatorname{lm}(f_{r-1}) \right\}.
\end{aligned}$$

After the reduction has taken place, the set M_i is extended by $\frac{\tau(f_i, f_j)}{\operatorname{lm}(f_i)}$ (Line 24) and the set M_{l+1} for the newly generated element f_{l+1} is constructed (Lines 19–20) by

$$\langle M_{l+1} \rangle = \langle M_i \rangle : \left\langle \frac{\tau(f_i, f_j)}{\operatorname{lm}(f_i)} \right\rangle + \langle f_1, \ldots, f_l \rangle.$$

Based on the above constructions of the M_i we see a completely new criterion for the detection of useless critical pairs in Line 12: A pair (f_i, f_j) is deleted whenever

$$m_i \Big| \frac{\tau(f_i, f_j)}{\operatorname{lm}(f_i)} \text{ for some } m_i \in M_i.$$

Let us discuss this:

Algorithm 25 Initial staggered linear basis algorithm (StagGB1)

Input: $F = \{f_1, \ldots, f_r\}$ a set of polynomials in $\mathcal{P}$
Output: G a Gröbner basis for $\langle F \rangle$, B a Gauss basis for $\langle F \rangle$

$P \leftarrow \varnothing, M_1 \leftarrow \varnothing$
for $\left(i = 2, \ldots, r\right)$ **do**
  $M_i \leftarrow M_{i-1} \cup \left\{ \operatorname{lm}(f_{i-1}) \right\}$
$G \leftarrow f_1$
for $(i = 2, \ldots, r)$ **do**
  $P \leftarrow P \cup \left\{ (f_i, f_j) \mid f_j \in G, j < i \right\}$
$l \leftarrow r$
while $(P \neq \varnothing)$ **do**
  $P' \leftarrow$ Select(P)
  $P \leftarrow P \backslash P'$
  while $(P' \neq \varnothing)$ **do**
    $(f_i, f_j) \leftarrow$ First element of P'
    $P' \leftarrow P' \backslash \{(f_i, f_j)\}$
    if $\left(\frac{\tau(f_i, f_j)}{\operatorname{lm}(f_i)} \notin \langle M_i \rangle \right)$ **then**
      $h \leftarrow \mathcal{S}(f_i, f_j)$
      $h \leftarrow \operatorname{NF}(h, G)$
      if $(h \neq 0)$ **then**
        $f_{l+1} \leftarrow h$
        $\langle Q \rangle \leftarrow \langle M_i \rangle : \left\langle \frac{\tau(f_i, f_j)}{\operatorname{lm}(f_i)} \right\rangle$
        $M_{l+1} \leftarrow \left\{ \operatorname{lm}(f_1), \ldots, \operatorname{lm}(f_r) \right\} \cup Q$
        $P \leftarrow P \cup \left\{ (f_{l+1}, f_j) \mid f_j \in G, j < l + 1 \right\}$
        $G \leftarrow G \cup \{f_{l+1}\}$
        $l \leftarrow l + 1$
        $M_i \leftarrow M_i \cup \left\{ \frac{\tau(f_i, f_j)}{\operatorname{lm}(f_i)} \right\}$
$B \leftarrow \bigcup_{i=1}^{r} \left\{ m f_i \in G \mid \operatorname{lm}(f_j) \mid m \operatorname{lm}(f_i) \Rightarrow j \geq i, m \in \operatorname{Mon}(x_1, \ldots, x_n) \right\}$
return (G, B)

(1) The initial construction of the M_i adds the leading monomials of all previous elements, i.e. elements of index $< i$. Assume we build an s–polynomial $\mathcal{S}(f_i, f_j)$ such that $\tau(f_i, f_j) = \operatorname{lm}(f_i) \operatorname{lm}(f_j)$. Then we know by Buchberger's 1st Criterion that we can discard this s–polynomial. Clearly, in this situation $\frac{\tau(f_i, f_j)}{\operatorname{lm}(f_i)} \in \langle M_i \rangle$ as $\operatorname{lm}(f_j) \in M_i$.

(2) If $\mathcal{S}(f_i, f_j)$ is rejected by Buchberger's 2nd Criterion, then there exists some $\mathcal{S}(f_i, f_k)$ such that $\tau(f_i, f_k) \mid \tau(f_i, f_j)$, which is computed before $\mathcal{S}(f_i, f_j)$ in Algorithm 25. But then $\frac{\tau(f_i, f_k)}{\operatorname{lm}(f_i)} \in M_i$, and thus $\mathcal{S}(f_i, f_j)$ can be rejected here, too.

The main idea of StagGB1 is to enhance the criteria checks of critical pairs, i.e. to find more useless critical pairs than GM. As seen in the short discussion above, whenever a critical pair is detected by one of Buchberger's criteria, StagGB1 also detects it.

Two main problems occur investigating Algorithm 25 a bit more closely:

Remark 3.1.7. As a matter of fact, StagGB1 has two drawbacks:

(1) The checks for useless criteria are not efficient enough. With an easy optimization we can fix this (as shown in Algorithm 26).

(2) The algorithm does not always return a Gröbner basis G for its input. Dellaca has shown in [52] that using the `Noon-3` example from [93] as input, StagGB1 returns the set[2] $G = \{g_1, \ldots, g_{10}\}$, which is not a Gröbner basis. This is due to the fact that $\mathcal{S}(g_4, g_9)$ does not reduce to zero w.r.t. G. This error is inherited from the fact that during the computations of the algorithm the critical pair (g_4, g_9) is detected being useless, and thus the corresponding s–polynomial, which would not reduce to zero, is not added to G.

Clearly, part (2) of Remark 3.1.7 must be solved, otherwise the idea of computing staggered linear bases to receive corresponding Gröbner bases is obsolete.

Mora resp. Dellaca gave a revised version of the algorithm including minor changes on the detection of useless critical pairs, which improves the first drawback mentioned in Remark 3.1.7. Moreover, a small restriction for the reducers computing the normal forms is introduced, which is the key point to ensure correctness of Algorithm 26.

In Line 14 we see the easy improvement of checking both multipliers of $\mathcal{S}(f_i, f_j)$, $\frac{\tau(f_i,f_j)}{\operatorname{lm}(f_i)}$ w.r.t. M_i as well as $\frac{\tau(f_i,f_j)}{\operatorname{lm}(f_i)}$ w.r.t. M_j.

The change having a larger impact on the computations is the new algorithm called StagNF used in Line 16. Let us have a closer look at the corresponding pseudo code given in Algorithm 27:

In Line 3 we see the difference to other normal form algorithms: Not all possible reducers $g_k \in G$ are allowed to reduce h, but only those for which the multiple $m \notin \langle M_k \rangle$.

Using this revised version of the algorithm one can prove the following:

Theorem 3.1.8. *If Algorithm 26 terminates, then the result is correct.*

Proof. See Theorem 3.9 in [52]. □

Sadly, the proof for termination of StagGB2 cannot be given, as it is not ensured anymore due to the restriction of the possible reducers in StagNF. A non–terminating example can be found in Chapter 3 of [52].

Remark 3.1.9.

(1) One should also be very careful on how to choose the next pair to be computed. In [80] the normal selection strategy is used. Mora, on the contrary, sorts the critical pairs by increasing degrees of the least common multiples of the pairs. In the following the idea behind this attempt, influenced by Faugère, becomes clearer.

[2] We use the corresponding notation from [52].

Algorithm 26 Revised staggered linear basis algorithm (StagGB2)

Input: $F = \{f_1, \ldots, f_r\}$ a set of polynomials in $\mathcal{P}$
Output: G a Gröbner basis for $\langle F \rangle$, B a staggered linear basis for $\langle F \rangle$

$P \leftarrow \varnothing, M_1 \leftarrow \varnothing$
for $\left(i = 2, \ldots, r\right)$ **do**
  $M_i \leftarrow M_{i-1} \cup \left\{ \mathrm{lm}(f_{i-1}) \right\}$
$G \leftarrow f_1$
for $(i = 2, \ldots, r)$ **do**
  $P \leftarrow P \cup \left\{ (f_i, f_j) \mid f_j \in G, j < i \right\}$
$l \leftarrow r$
while $(P \neq \varnothing)$ **do**
  $P' \leftarrow \textsc{Select}(P)$
  $P \leftarrow P \backslash P'$
  while $(P' \neq \varnothing)$ **do**
    $(f_i, f_j) \leftarrow$ First element of P'
    $P' \leftarrow P' \backslash \{(f_i, f_j)\}$
    if $\left(\frac{\tau(f_i, f_j)}{\mathrm{lm}(f_j)} \notin \langle M_j \rangle \text{ and } \frac{\tau(f_i, f_j)}{\mathrm{lm}(f_i)} \notin \langle M_i \rangle \right)$ **then**
      $h \leftarrow \mathcal{S}(f_i, f_j)$
      $h \leftarrow \textsc{StagNF}(h, G)$
      if $(h \neq 0)$ **then**
        $f_{l+1} \leftarrow h$
        $\langle Q \rangle \leftarrow \langle M_i \rangle : \left\langle \frac{\tau(f_i, f_j)}{\mathrm{lm}(f_i)} \right\rangle$
        $M_{l+1} \leftarrow \left\{ \mathrm{lm}(f_1), \ldots, \mathrm{lm}(f_r) \right\} \cup Q$
        $P \leftarrow P \cup \left\{ (f_{l+1}, f_j) \mid f_j \in G, j < l + 1 \right\}$
        $G \leftarrow G \cup \{f_{l+1}\}$
        $l \leftarrow l + 1$
        $M_i \leftarrow M_i \cup \left\{ \frac{\tau(f_i, f_j)}{\mathrm{lm}(f_i)} \right\}$
$B \leftarrow \bigcup_{i=1}^{r} \left\{ m f_i \in G \mid \mathrm{lm}(f_j) \mid m\, \mathrm{lm}(f_i) \Rightarrow j \geq i, m \in \mathrm{Mon}(x_1, \ldots, x_n) \right\}$
return (G, B)

(2) An in–depth discussion on the problem of the dependencies of termination and correctness of the algorithms StagGB1 and StagGB2 is given in [57]. There the problem of termination of Faugère's F5 Algorithm is brought to light and different attempts to ensure termination are presented. See Section 6.5 for more details on this topic.

(3) Note that in [51] a quite similar attempt of computing a Gröbner basis is given. There it is proven that, if we have just added a new element f_i to G, it is enough to consider only those s–polynomials $\mathcal{S}(f_i, f_j)$, $j < i$, such that $\frac{\tau(f_i, f_j)}{\mathrm{lm}(f_i)}$ involves some element of $M_i := \left\langle \mathrm{lm}(f_1), \ldots, \mathrm{lm}(f_{i-1}) \right\rangle : \mathrm{lm}(f_i)$. All other s–polynomials are useless and need not be computed.

All in all, the following roundup can be done according to staggered linear bases and

Algorithm 27 Normal form computation for staggered linear bases (StagNF)

Input: f a polynomial in $\mathcal{P}$, $G = \{g_1, \ldots, g_s\}$ a set of polynomials in $\mathcal{P}$
Output: h the staggered normal form of f w.r.t. G

```
h ← f
while (h ≠ 0) do
    if (D_h := {(m, g_k) ∈ Mon(x_1, ..., x_n)\⟨M_k⟩ × G | m lm(g_k) = lm(h)} ≠ ∅) then
        Choose any (m, g) ∈ D_h.
        h ← h − (lc(h)/lc(g)) m g
    else
        return h
return h
```

how to compute them:

(1) A new criterion to detect useless critical pairs is introduced.

(2) Correctness of the computation using this new criterion is based on a restriction of possible reducers.

(3) This restriction as well as the sorting of the critical pairs influence if the algorithm terminates or not.

It is quite amazing that all these facts, which are the hard parts understanding Faugère's F5 Algorithm and related signature–bases algorithms, pop up already at this point. Clearly, staggered linear bases have kicked off the ideas behind the algorithms in focus of this thesis.

Note that we do not try to give any proof of the above presented facts. This is based on the problem that understanding the ideas in full can be done much nicer with another structure we need to introduce, namely *syzygies*. Using syzygies our view on the ideas presented in this section changes dramatically and enables us to get a deeper insight.

3.2 Syzygies and free resolutions

A *syzygy* is a very important structure in commutative algebra storing the relationships between elements. Having some elements $f_1, \ldots, f_r \in \mathcal{M}$ the question *how independent they are from each other* arises quite naturally. The *syzygy module* stores exactly these information. This procedure can be repeated by searching for relations between the generators of the syzygy module, which leads to a so–called *free resolution*, another very important concept in commutative algebra.

In this section we introduce the notions of syzygies and free resolutions. We see that whenever we want to compute the generators of the syzygy module of some module M, the standard basis for M is used for this attempt.

As we have already noted in Section 3.1 we are interested in the other way around: *How can we use information about the (partially computed) syzygy module to improve standard basis computations?*

Definition 3.2.1. A *complex $\mathcal{C}$ of $\mathcal{P}$–modules M_i* is a (in–)finite sequence

$$\ldots \longrightarrow M_{k+1} \xrightarrow{\phi_{k+1}} M_k \xrightarrow{\phi_k} M_{k-1} \longrightarrow \ldots$$

where $\phi_k \circ \phi_{k+1} = 0$ for all k.

(1) $\mathcal{C}$ is called *exact at M_k* if $ker(\phi_k)/\operatorname{im}(\phi_{k+1}) = 0$.

(2) $\mathcal{C}$ is called an *exact sequence* if it is exact at every M_k.

Let us see how we get a free presentation of an arbitrary $\mathcal{P}$–module M. We choose generators $\{f_i\}$ of M, as well as generators $\{e_i\}$ of a free $\mathcal{P}$–module F_0. Next we consider the module homomorphism π defined by

$$\begin{aligned} \pi : \quad F_0 &\longrightarrow M \\ e_i &\longmapsto f_i. \end{aligned}$$

We can reproduce this step again exchanging M by $\ker(\pi)$ and F_0 by another free $\mathcal{P}$–module F_1. Rewriting

$$F_1 \longrightarrow \ker(\pi) \longrightarrow F_0$$

by σ we get an exact sequence

$$F_1 \xrightarrow{\sigma} F_0 \xrightarrow{\pi} M \longrightarrow 0.$$

Repeating this process, one receives a possibly infinite exact sequence

$$\ldots \longrightarrow F_{k+1} \xrightarrow{\phi_{k+1}} F_k \xrightarrow{\phi_k} F_{k-1} \ldots \longrightarrow F_1 \xrightarrow{\phi_1} F_0 \xrightarrow{\pi} M \longrightarrow 0. \tag{3.2.1}$$

Definition 3.2.2. With the notations as above we define the following:

(1) Every such exact sequence as in 3.2.1 is called a *free resolution of M*.

(2) If there exists an integer l such that $F_i = 0$ for all $i > l$, then the free resolution is called to be *finite of length l*.

(3) The images $\operatorname{im}(\phi_k)$ are called the *k–th syzygy module of M*. We denote them by $\operatorname{Syz}_k(M) := \operatorname{Syz}_k(f_1, \ldots, f_r)$.

(4) If $M = \langle f_1, \ldots, f_r \rangle$ is an ideal in $\mathcal{P}$, then any element $f_j e_i - f_i e_j$, where $j < i$, is called a *principal syzygy of M*.

Convention. We are mostly interested in the first syzygy module of some module M. Thus let us agree on the shorthand notation $\operatorname{Syz}(M) = \operatorname{Syz}_1(M)$ for the rest of this thesis. Moreover, "syzygy module" always means "first syzygy module".

Remark 3.2.3. Besides this a bit abstract definition one can think of syzygies in a rather vivid way:

(1) One can think of a syzygy of M as a relation $(g_1, \ldots, g_k) \in \mathcal{P}^k$ of elements $f_1, \ldots, f_k$ such that
$$\sum_{i=1}^{k} g_i f_i = 0.$$
Thus the first syzygy module defined above can be understood as $\text{Syz}(M) = \ker(\pi)$.

(2) Note that, defining $\text{Syz}_0(M) = M$, we can also define the k–th syzygy module of M recursively by
$$\text{Syz}_k(M) := \text{Syz}\left(\text{Syz}_{k-1}(M)\right), k \geq 0.$$

In the following, let us sum up some of the main properties of syzygies resp. free resolutions of a finitely generated module M over $\mathcal{P}$. You can find proofs of these statements in any introductory book on commutative algebra or computer algebra, e.g. [51, 97, 112].

One remarkable theorem proved by Hilbert in 1890 is stated in the following. It can be seen as the initial point of *homological algebra*.

Theorem 3.2.4 (Hilbert's Syzygy Theorem)**.** *Every finitely generated $\mathcal{P}$–module M has a finite free resolution of finitely generated, free $\mathcal{P}$–modules, which has length at most n.*

Example 3.2.5. Let $\mathcal{P} = \mathcal{K}[x, y, z]$ with $<_{\text{dp}}$. Consider the polynomials
$$\begin{aligned} f_1 &= xy^2 - xz, \\ f_2 &= 3x^2 - yz, \\ f_3 &= 2z^4 - xy^2. \end{aligned}$$

On the one hand it is easy to give some syzygies of $F = \{f_1, f_2, f_3\}$. For example, one could only consider two out of the three elements by multiplying the third by 0:
$$g_1 f_1 + g_2 f_2 + 0 f_3 \stackrel{!}{=} 0.$$

One solution for the above equation is to set $g_1 = f_2$ and $g_2 = -f_1$, as $f_2 f_1 - f_2 f_1 = 0$. This process can be done for all $\binom{3}{2}$ combinations:
$$\begin{aligned} f_2 f_1 - f_1 f_2 - 0 f_3 &= 0, \\ f_3 f_1 - 0 f_2 - f_1 f_3 &= 0, \\ 0 f_1 - f_3 f_2 - f_2 f_3 &= 0. \end{aligned}$$

Even if it is easy to state the principal syzygies of $\{f_1, f_2, f_3\}$, the real problem lies in finding the generators of a basis for $\text{Syz}(f_1, f_2, f_3)$.

We already know that the kernel of a module homomorphism is again a module. Thus searching a set of generators of $\text{Syz}(M)$ is nothing else but the question about the computation of a standard basis for $\text{Syz}(M)$. For this, different algorithms are known. Let us assume in the following that $F = \{f_1, \ldots, f_r\} \subset \mathcal{P}^k$, $M = \langle F \rangle$.

(1) One method is presented in Algorithm 28: First a standard basis G' for the module $\langle F' \rangle$ is computed (Line 3), where

$$F' := \{f_1 + e_{r+1}, \ldots, f_k + e_{r+k}\} \subset \mathcal{P}^{r+k}.$$

The elements $e_1, \ldots, e_{r+k}$ are the canonical generators of $\mathcal{P}^{r+k}$. From this computation we can extract three different, but related elements:

a) The syzygy module $\mathrm{Syz}(M)$ is constructed by taking all elements g' from G' for which the first k entries are zero and extract the last r entries (Line 4):

$$g' = (\underbrace{0, \ldots, 0}_{k\text{-times}}, \underbrace{h_1, \ldots, h_r}_{r\text{-times}}) \in \mathcal{P}^{r+k}$$
$$\Rightarrow h = (h_1, \ldots, h_r) \in \mathrm{Syz}(M).$$

b) We also get the standard basis $G = \{g_1, \ldots, g_s\}$ of M w.r.t. $<$ (Line 5): Here we take all elements $g' \in G'$ such that the first k entries are not zero.

$$\begin{aligned} g' &= (g_1, \ldots, g_k, h_1, \ldots, h_r) \in \mathcal{P}^{r+k} \\ \Rightarrow \quad g &= (g_1, \ldots, g_k) \in G. \end{aligned}$$

c) Moreover, we also get a third important structure, the transformation matrix T, which fulfills the equation

$$(g_1, \ldots, g_s)^t = T \cdot (f_1, \ldots, f_r)^t.$$

We do not give a proof of this concept, you can find a discussion on this algorithm and its theoretical basics in Section 2.5 of [97].

Algorithm 28 Standard basis algorithm for the first module of syzygies (Syz1)

Input: $F = \{f_1, \ldots, f_r\}$ a finite subset of $\mathcal{P}^k$, $\{e_1, \ldots, e_{r+k}\}$ a set of canonical generators of $\mathcal{P}^{r+k}$, $<$ a module order on $\mathcal{P}^k$

Output: G a standard basis for F w.r.t. $<$, S a finite subset of $\mathcal{P}^k$ such that $\langle S \rangle = \mathrm{Syz}(F)$ w.r.t. $<$, T a matrix in $\mathcal{P}^{s \times r}$ where $s = \#(G)$

1: $F' \leftarrow \{f_1 + e_{r+1}, \ldots, f_r + e_{r+k}\}$
2: $<' \leftarrow$ generalization of $<$ to $\mathcal{P}^{r+k}$
3: $G' \leftarrow \mathrm{STD}(F', \eta, <')$
4: $S \leftarrow \{h \mid h \in \mathcal{P}^r, (0, h) \in G'\}$
5: $G \leftarrow \{g \mid g \in \mathcal{P}^k, (g, h) \in G', g \neq 0\}$
6: $s \leftarrow \#(G)$
7: $T \leftarrow 1_{s \times r}$
8: **for** $(i = 1, \ldots, s)$ **do**
9: row T_i of $T \leftarrow h_i$ where $(g_i, h_i) \in G'$
10: **return** (G, S, T)

(2) The second approach we want to mention in this thesis is a bit more straightforward and can be found in several publications (e.g. [112, 135, 155]).

The main idea of Syz2, presented in Algorithm 29, is to compute the standard basis for F and the standard basis for $\mathrm{Syz}(F)$ at the same time. It computes the the standard basis for F directly and not with some detour over a higher dimensional computation as presented in Algorithm 28. For an easier notation, we restrict ourselves to the case $F = \{f_1, \ldots, f_r\} \subset \mathcal{P}$ in the following. Whereas the standard basis for F is stored in G, the standard basis for $\mathrm{Syz}(F)$ is stored in S. So, in some sense we just use the Gebauer–Möller implementation GM and add some overhead to it: Whenever we compute the normal form of an s–vector $\mathcal{S}(f_i, f_j)$ we need to do some bookkeeping and store all reducers resp. corresponding multipliers λ_k needed to get from $\mathcal{S}(f_i, f_j)$ to h (see Line 18). In the end two situations are possible:

a) If $h \neq 0$, then we need to add h to G in order to compute a standard basis for F.

b) If $h = 0$, then we do not need to add anything to G, but we need to add the syzygy corresponding to this zero reduction to S. For this we need to store the bookkept data from the normal form computation in some elemente e_{m+1} (Line 19) and add it to S (Line 27). Note that we have to rewrite any element e_k corresponding to a reducer f_k in e_{m+1} where $k > r$ by its corresponding relation such that in the end all $k' \leq r$.

It was firstly proved by Wall in [155] that one can use the Buchberger criteria, which detect useless critical pairs, also for computing syzygies. In particular one can implement the algorithm in exactly the same way we have presented Algorithm 29: A Gebauer–Möller implementation with some overhead for the storage of the syzygies. Note that whereas it is shown in [155] that the 2nd Buchberger Criterion does not influence the computation of $\mathrm{Syz}(F)$, the 1st Buchberger Criterion does. For this matter of fact we need to add all syzygies $f_j e_i - f_i e_j$ for $j < i$ in lines 5 – 6. Every s–vector that fulfills Buchberger's 1st Criterion (and thus reduces to zero) corresponds to a multiple of such a syzygy. It follows that we can add these elements to S.

Remark 3.2.6. Let us have a closer look at the computational aspects of the different, above presented algorithms constructing a standard basis for $\mathrm{Syz}(M)$.

(1) Syz1 goes the indirect way, i.e. it does not compute a standard basis of M or $\mathrm{Syz}(M)$, but extracts them from a related computation: On the one hand, we need to compute a standard basis in $\mathcal{P}^{r+k}$ which can be much harder than a corresponding computation in $\mathcal{P}^k$. On the other hand, this is the only "real" computation that needs to be done. All other results can be obtained from G' by extracting special elements.

(2) Syz2 has the advantage to not compute a standard basis for a module of higher rank, but it has the drawback of storing and remembering all the different reducers and their multiples during a normal form computation of some s–vector. This overhead can be the determining part of running time and memory consumption; it can even render the computation of a basis for the syzygy module of an ideal, for which the standard basis computation is straightforward, impossible.

(3) In [10] Ars and Hashemi give an attempt of using the matrix version of the F5 Algorithm to compute a basis for the module of syzygies of F in the vein of Syz2. This

matrix version does not use the full strengths of F5. We see in Section 7.6 that one can do even better, using some improved variant of F5 to compute $\mathrm{Syz}(M)$.

Algorithm 29 Standard basis algorithm for the first module of syzygies (Syz2)

Input: $F = \{f_1, \ldots, f_r\}$ a finite subset of $\mathcal{P}$, $<$ a module order over $\mathcal{P}$
Output: G a standard basis for F w.r.t. $<$, S a finite subset of $\mathcal{P}^s$ such that $\langle S \rangle = \mathrm{Syz}(G)$

$S \leftarrow \varnothing$
$P \leftarrow \varnothing$
for $(i = 2, \ldots, r)$ **do**
  $P \leftarrow \textsc{Update}(P, G, f_i)$
  for $j = 1, \ldots, i-1$ **do**
    $S \leftarrow S \cup \{f_j e_i - f_i e_j\}$
  $G \leftarrow G \cup \{f_i\}$
$l \leftarrow r$
$m \leftarrow \#(S)$
while $(P \neq \varnothing)$ **do**
  $P' \leftarrow \textsc{Select}(P)$
  $P \leftarrow P \backslash P'$
  while $(P' \neq \varnothing)$ **do**
    $(f_i, f_j) \leftarrow$ First element of P'
    $P' \leftarrow P' \backslash \{(f_i, f_j)\}$
    $h \leftarrow \mathrm{lc}(f_j) \frac{\tau(f_i,f_j)}{\mathrm{lm}(f_i)} f_i - \mathrm{lc}(f_i) \frac{\tau(f_i,f_j)}{\mathrm{lm}(f_j)} f_j$
    $e_{m+1} \leftarrow \mathrm{lc}(f_j) \frac{\tau(f_i,f_j)}{\mathrm{lm}(f_i)} e_i - \mathrm{lc}(f_i) \frac{\tau(f_i,f_j)}{\mathrm{lm}(f_j)} e_j$
    $h \leftarrow \mathrm{NF}(h, G)$ such that $h = \mathcal{S}(f_i, f_j) - \sum_{k=1}^{l} \lambda_k f_k$
    $e_{m+1} \leftarrow e_{m+1} - \sum_{k'=1}^{r} \lambda_{k'} e_{k'}$
    if $(h \neq 0)$ **then**
      $f_{l+1} \leftarrow h$
      $P \leftarrow \textsc{Update}(P, G, f_{l+1})$
      $G \leftarrow G \cup \{f_{l+1}\}$
      $l \leftarrow l + 1$
      $m \leftarrow m + 1$
    else
      $S \leftarrow S \cup \{e_{m+1}\}$
      $m \leftarrow m + 1$
return (G, S)

Example 3.2.7 (Example 3.2.5 revisited)**.** Let us reconsider the previous example, let us assume $<_{\mathrm{i}}$ on $\mathcal{P}^3$. We have already given some syzygies of $F = \{f_1, f_2, f_3\}$, namely the principal ones. We try to compute a basis for $\mathrm{Syz}(F)$ using Syz2 in the following. Clearly, $\mathcal{S}(f_3, f_1)$ as well as $\mathcal{S}(f_3, f_2)$ reduce to zero due to Buchberger's 1st Criterion. Let us reduce

$\mathcal{S}(f_2, f_1)$:

$$f_4 := -y^3z + yz^2 = \underbrace{3x^2y^2 - y^3z - 3x^2y^2 + 3x^2z}_{\mathcal{S}(f_2,f_1)} - \underbrace{(3x^2z - yz^2)}_{zf_2},$$

$$e_4 := (y^2 - z)e_2 - 3xe_1.$$

Next we need to compute $\mathcal{S}(f_4, f_1)$ and $\mathcal{S}(f_4, f_3)$:

$$0 = \underbrace{-xy^3z + xyz^2 + xy^3z - xyz^2}_{\mathcal{S}(f_4,f_1)},$$

$$\begin{aligned} e_5 &:= xe_4 + yze_1 \\ &= (xy^2 - xz)e_2 - (3x^2 - yz)e_1. \end{aligned}$$

$$0 = \underbrace{-2z^3z^4 + 2yz^5 + 2y^3z^4 - xy^5}_{\mathcal{S}(f_4,f_3)} + \underbrace{xy^5 - xy^3z}_{y^3f_1} - \underbrace{(2yz^5 - xy^3z)}_{yzf_3},$$

$$\begin{aligned} e_6 &:= 2z^3e_4 + (y^3 - yz)e_3 + y^3e_1 \\ &= (y^3 - yz)e_3 + (2y^2z^3 - 2z^4)e_2 + (-6xz^3 - y^3)e_1. \end{aligned}$$

At this point the standard basis computation stops and we have $G = \{f_1, f_2, f_3, f_4\}$. Moreover, we can state a basis S for $\operatorname{Syz}(F)$: This is done in the easiest way using a so–called *syzygy matrix*, in which the ith row can be understood as the ith computed syzygy and the jth column represents the jth canonical module generator e_j. With this the following notation is quite clear:

$$\begin{pmatrix} -3x^2 + yz & xy^2 - xz & 0 \\ -2z^4 + xy^2 & 0 & xy^2 - xz \\ 0 & -2z^4 + xy^2 & 3x^2 + yz \\ -6xz^3 + y^3 & 2y^2z^3 - 2z^4 & y^3 - yz \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Note that e_5 computed above coincides with the principal syzygy $f_1e_2 - f_2e_1$, thus we only have 4 generators of the basis S of $\operatorname{Syz}(F)$.

Remark 3.2.8.

(1) In the following we see that the principal syzygies correspond exactly to one of the criteria used in signature–based standard basis algorithms. From this it follows that standard bases for ideals having only these relations of their generators can be computed without any zero reduction.

(2) Using the above methods recursively to compute syzygies of syzygies one can compute free resolutions of a free module $M \in \mathcal{M}$ to a given length l.

This finishes our introduction to syzygies and their computations. We have seen that, given a module M, the computation of a standard basis for M is used to improve the calculations for a standard basis for $\operatorname{Syz}(M)$.

The other way around is just the starting point for signature–based standard basis attempts: *How can we use simultaneously computed syzygies of M (or parts of them) to improve the computation of a standard basis for M?*

3.3 Computing standard bases using syzygies

Until now we have a one–way connection between standard bases and syzygies: Use the standard basis computation to obtain a basis for the syzygy module from it. This attempt is presented in Section 3.2 and pretty well known over the last couple of years.

In this section we try to go the other way around: How to use information from syzygies to make standard basis computations more efficient?

Again we restrict the discussion to the polynomial situation, i.e. $F = \{f_1, \ldots, f_r\}$ is a finite subset of polynomials in $\mathcal{P}$.

Algorithm 30 Standard basis algorithm using syzygies to improve computations (SyzStd)

Input: $F = \{f_1, \ldots, f_r\}$ a finite subset of $\mathcal{P}$, $<$ a module order over $\mathcal{P}$

Output: G a standard basis for F w.r.t. $<$

$G \leftarrow \emptyset, P \leftarrow \emptyset, S \leftarrow \emptyset$
for $i = 1, \ldots, r$ **do**
  $G \leftarrow G \cup \{f_i\}$
  $P \leftarrow P \cup \left\{ue_t \mid \exists k < i \text{ and } u = \min\left\{u \in \mathcal{P} \mid u\,\mathrm{lm}(f_i) = \tau(f_i, f_k)\right\}\right\}$
$t \leftarrow r$
while $(P \neq \emptyset)$ **do**
  $P' \leftarrow$ Select(P)
  $P \leftarrow P \backslash P'$
  while $(P' \neq \emptyset)$ **do**
    $me_j \leftarrow$ First element of P'
    Reduce(S)
    $P' \leftarrow P' \backslash \{me_j\}$
    if $\big((\nexists s \in S$ such that $\mathrm{lm}(s) \mid me_j)$ or (some other criterion for $me_j)\big)$ **then**
      $h \leftarrow mf_j$
      $e_{t+1} \leftarrow me_j$
      $(h, e_{t+1}) \leftarrow$ SyzNF(h, e_t, G)
      if $(h \neq 0)$ **then**
        $t \leftarrow t + 1$
        $f_t \leftarrow h$
        $P \leftarrow P \cup \left\{ue_t \mid \exists k < t \text{ and } u = \min\left\{u \in \mathcal{P} \mid u\,\mathrm{lm}(f_t) = \tau(f_t, f_k)\right\}\right\}$
        $G \leftarrow G \cup \{f_t\}$
        Add to S further known syzygies if possible.
      else
        $s \leftarrow e_{t+1}$
        $S \leftarrow S \cup \{s\}$
return (G, S)

The initial idea was given in 1992 by Möller, Mora, and Traverso in [126]. Based on the discussions of Section 3.1 currently computed syzygies are used to detect useless critical pairs in the standard basis computation.

We present the algorithm given in [126] in a slightly different notation, which fits better to the further discussions on signature–based algorithms. The pseudo code of it is divided into two parts: The main part is Algorithm 30, which represents the overall computations. In there SyzNF is called (Algorithm 31), which is a special version of already presented normal form algorithms (see Sections 1.7 and 2.6).

Remark 3.3.1.

(1) Note that in [126] different versions of SyzStd are given. We concentrate on the standard version, as we discuss the other variants described there in the signature–based setting later on. Moreover, we focus on the explanation why the algorithm works and what are the crucial improvements that can be done.

(2) The algorithm presented here also differs from the presentation in [126]. In our discussion we focus on the connection to signature–based standard basis algorithms. Thus we have adjusted some small pieces to fit better in that picture. Nevertheless, the changes are minor and do not influence the behaviour of the algorithm fundamentally.

Algorithm 31 Normal form w.r.t. G of SyzStd (SyzNF)

Input: f_i a polynomial in $\mathcal{P}$, e_l a module element in $\mathcal{P}^m$, $G \subset \mathcal{P}$ a finite sequence
Output: h the normal form of f_i w.r.t. SyzStd, e in $\mathcal{P}^m$

1: $e \leftarrow e_l$
2: $h \leftarrow f_i$
3: **while** $\left(h \neq 0 \text{ and } D_h \leftarrow \left\{f_j \in G \mid \operatorname{lm}(f_j) \mid \operatorname{lm}(h) \text{ and } (j \neq i \text{ or } e \neq e_l)\right\} \neq \emptyset\right)$ **do**
4: Choose any $f_j \in D_h$.
5: $h \leftarrow h - \frac{\operatorname{lt}(h)}{\operatorname{lt}(f_j)} f_j$
6: $e \leftarrow e - \frac{\operatorname{lt}(h)}{\operatorname{lt}(f_j)} e_j$
7: **return** (h, e)

Algorithm 30 looks quite similar to Algorithm 29, but differs in some crucial points:

(1) First of all, SyzStd does not handle s–vectors, but only multiples of elements mf_i in G. Those can be understood to be one of the generators of an s–vector, whereas the 2nd generator is dynamically chosen by SyzNF as the first allowed reducer (Line 4).

(2) SyzStd does not use algorithm Update (introduced in the Gebauer–Möller implementation in Section 2.4), which detects useless critical pairs and generates useful ones in Syz2. Instead, new criteria are used, which are in some sense quite similar to the ideas presented in Section 3.1: We see in Line 13 that only those elements mf_i are considered, if there exists no syzygy $s \in S$ already such that $\operatorname{lm}(s) \mid me_i$. Moreover, other possible variants of this syzygy criterion can be used, for more details on this see [126]. We consider such more sophisticated criteria in the follwing chapters, when we are discussing signature–based attempts.

We see in lines 4 and 20 that only those new multiples ue_j are added to P and are then further reduced, which are minimal in the sense that there exists an element $f_k \in G$, $k < j$ such that

$$u \operatorname{lm}(f_j) = \tau(f_j, f_k) \text{ where } u \text{ is minimal with this property.}$$

This is essentially the same idea as given in the staggered linear basis setting. All other possible multiples of mf_j are useless and would be rejected by the criterion in Line 13 either way.

(3) In Line 11 a procedure Reduce is called with S as a parameter. We do not define Reduce, but we just explain a way how this can be implemented. The main idea is the following: Assume that there are two syzygies s_1 and s_2 in S, added during previous computations, which leading monomials are multiples of each other, that means that there exists $m \in \operatorname{Mon}(x_1, \ldots, x_n)$ such that $m \operatorname{lm}(s_1) = \operatorname{lm}(s_2)$. In SyzStd those syzygies are only used to detect useless elements in Line 13. This detection is only based on the leading monomial of the syzygies, thus it is useful to compute

$$s_2 := s_2 - ms_1.$$

After this computation, $\operatorname{lm}(s_2) < m \operatorname{lm}(s_1)$ and thus possibly more useless elements are detected. In other words, Reduce could be implemented as an interreduction of S such that as a result all elements remaining in S have different leading monomials.

(4) In a very similar manner Line 22 can be understood: Sometimes one has some more data about the module by some background information which can lead to new syzygies that can be added during the computations. Of course, more syzygies in S cause more possible detections of useless elements in Line 13.

(5) In Line 16 the new normal form SyzNF is called. Its pseudo code is described in Algorithm 31: The main difference between a usual normal form computation and this one is that in SyzNF only some elements of G are allowed to reduce. Other elements are not allowed because they do not only come from the same e_i, but also have the very same me_i as leading monomial. If such reductions would be allowed one could for example reduce mf_i by mf_i, which is something we clearly do not want. Thus for the allowed reducer of an element $h = m \cdot \pi(e_i)$ the following hold:

   a) Either the reducer r comes from another e_j than h,
   b) or the reducer r comes from the same e_i, but $m_r e_i \neq me_i$ for $m_r \operatorname{lm}(r) = \operatorname{lm}(h)$.

With the above discussion, termination and correctness of SyzStd follow easily, thus the following theorem holds:

Theorem 3.3.2. *Let $F \subset \mathcal{P}$ be the input of SyzStd. Then SyzStd is an algorithm computing a standard basis G for $\langle F \rangle$ w.r.t. $<$.*

Proof. See the above discussion and [126]. □

Remark 3.3.3.

(1) See Section 5 of [126] for more information on possible implementations of further criteria indicated in Line 13 of SyzStd. Moreover, more variants of the algorithm are given there, too.

(2) Note that the selection of which new elements have to be added to P in Line 20 of Algorithm 30 is also known from other settings than the staggered linear basis one, for example see [51].

(3) Also the choice of the order of elements in P' must be done with lots of care: One no longer orders the elements mf_i by increasing leading monomials, but the corresponding elements me_i. Since the syzygy module is the kernel of the map $\pi : \mathcal{P}^t \to \mathcal{P}^k$, the module order on $\mathcal{P}^t$ is decisive. Clearly, the me_i are just the leading monomials of the possibly later on added syzygies $s \in S$. Thus it is senseless to compute an element ue_i before an element ve_j with $ue_i > ve_j$, since ue_i cannot add some new information for a possible rejection of ve_j in the following, whereas this can happen the other way around.

(4) In [126] the authors suggest quite a lot of optimizations, for example different implementations of the more vaguely described subalgorithms like Reduce or other criteria to be used. Other ideas cover computations of the syzygies modulo a prime p or even storing only parts of the syzygy. This speeds up the computations quite a lot.

(5) Note that Reduce(S) in Line 11 could not only add new criteria for rejection of useless elements when computing $s_2 := s_2 - s_1$ for two elements $s_1, s_2 \in S$ with the same leading monomial, but it also makes the criteria check more efficient: Thinking about having multiple elements with the same leading monomial ve_j in S and an element ue_i to be checked, but which is not detected to be useless, one would do these checks several times, whereas we know already after the first check that it is not detected by this leading monomial. Thus Reduce(S) does also make the criteria checks more efficient, even if $s_2 - s_1$ leads to no new leading monomial in S (and reduces to zero in further iterations of the reduction step).

As we have already noted in Remark 3.3.3 (3), one must be careful with the choice of the module order on $\mathcal{P}^t$, in which P, P', and S live. Also we have already given some module orders in Example 1.4.5, namely $<_{\mathrm{m}}$ and $<_{\mathrm{i}}$, those lack a connection between the module and the ring world. Considering syzygies, we have a connection via the map $\pi : \mathcal{P}^t \to \mathcal{P}^k$ such that $\pi(e_i) = f_i$ for all elements $f_i \in G$. This is achieved in the following way: In the beginning, assuming $F = \{f_1, \ldots, f_r\}$ it holds for all elements $f_i \in F$ where $\pi : \mathcal{P}^r \to \mathcal{P}^k$. Adding a new element f_{r+1} to G, we generalize π to $\pi' : \mathcal{P}^{r+1} \to \mathcal{P}^k$ by

$$\pi'(e_i) := \begin{cases} \pi(e_i) & \text{if } 1 \leq i \leq r, \\ f_{r+1} & \text{else.} \end{cases}$$

Doing this iteratively we get a connection between the e_is and the f_is for all elements in G. Thus it makes sense to define a new module order, which incorporates these relations:

Example 3.3.4 (Schreyer order)**.** Let $G = \{f_1, \ldots, f_s\}$ be a subset of $\mathcal{P}$. Moreover, let $\pi : \mathcal{P}^s \to \mathcal{P}$ be a map between finitely generated, free $\mathcal{P}$–modules such that $\{e_1, \ldots, e_s\}$ is a basis of $\mathcal{P}^s$, and $\pi(e_i) = f_i$ for all $f_i \in G$. Then we can define a module order $\prec_{\mathrm{lm}}$ on $\mathcal{P}^s$ by

$$\begin{aligned} m_i e_i \prec_{\mathrm{lm}} m_j e_j :\Longleftrightarrow\ & m_i \operatorname{lm}(f_i) < m_j \operatorname{lm}(f_j) \text{ or,} \\ & m_i \operatorname{lm}(f_i) = m_j \operatorname{lm}(f_j) \text{ and } i < j. \end{aligned}$$

This order prefers the leading monomial of the image under π to the index of the element. Clearly, as we want to use this order in SyzStd for a standard basis computation in $\mathcal{P}$ such a privilege makes absolutely sense.

The main improvement of SyzStd is that it finds more useless elements during the standard basis computation than GM does. This is achieved by using the criterion in Line 13 of Algorithm 30 instead of Buchberger's criteria. Let us illustrate this in an example:

Example 3.3.5. Let us give an example computation of a Gröbner basis for some ideal. Assume that $\mathcal{P} = \mathcal{K}[x, y, z]$ is equipped with $<_{\mathrm{dp}}$, and let $F = \{f_1, f_2, f_3\}$ where

$$\begin{aligned} f_1 &= x^2 y - z^2, \\ f_2 &= xz^2 - y^2, \\ f_3 &= yz^3 - x^2. \end{aligned}$$

Moreover, let us use the module order $\prec_{\mathrm{lm}}$ defined in Example 3.3.4 on $\mathcal{P}^3$ (and on each intermediate $\mathcal{P}^s$ during the Gröbner basis computation, defined as explained in Example 3.3.4). For the selection strategy in P we always choose the smallest element w.r.t. $\prec_{\mathrm{lm}}$. Furthermore note that we compute a normal form as reduced as possible with the restrictions given in SyzNF.

In the beginning we start with $P = \{e_1, e_2, e_3\}$, $S = \emptyset$. Clearly, $e_1 \prec_{\mathrm{lm}} e_2 \prec_{\mathrm{lm}} e_3$, thus f_1 is the first element added to G. As there are no other elements in G to generate critical pairs with at this moment, we go on with adding f_2 to G, but at this point we add xye_2 to P. Next f_3 is added to G and the elements xe_3 and $z^2 e_3$ are possible new elements for P. We see in Line 20 that only the smaller one of these two, xe_3 needs to be added to P. Note that at this point we already have some elements in S, namely the principal syzygies

$$\begin{aligned} s_{2,1} &= f_1 e_2 - f_2 e_1, \\ s_{3,1} &= f_1 e_3 - f_3 e_1, \\ s_{3,2} &= f_2 e_3 - f_3 e_2. \end{aligned}$$

Right now $P = \{xe_3, xye_2\}$ and we go on with xf_3. It reduces to an element

$$\begin{aligned} f_4 &= y^3 z - x^3, \\ e_4 &= xe_3 - yze_2. \end{aligned}$$

Using $s_{3,2}$ and e_4 together we can get another syzygy representation for S:

$$\begin{aligned} & s_{3,2} = xz^2 e_3 - yz^3 e_2 - y^2 e_3 + x^2 e_2 \\ \Rightarrow\ & s_4 = z^2 e_4 - y^2 e_3 + x^2 e_2. \end{aligned}$$

So f_4 is added to G and $P = \{xye_2, z^2e_4, xze_4, x^2e_4\}$ after constructing new elements. Moreover, clearly we have $s_{4,1}$, $s_{4,2}$, and $s_{4,3}$ in S, too. xyf_2 reduces to an element

$$\begin{aligned} f_5 &= -xy^3 + z^4, \\ e_5 &= xye_2 - z^2e_1. \end{aligned}$$

Similarly to above we add a syzygy $s_5 = xe_5 + y^2e_1 - z^2e_2$ (computed out of $s_{2,1}$) and the principal syzygies $s_{5,1}, \dots, s_{5,4}$ to S. Two new elements are added to P,

$$P = \{ze_5, xe_5, z^2e_4, xze_4, x^2e_4\}.$$

Reducing zf_5 we receive a sixth element for G and new representation for $s_{5,4}$ (plus the principal syzygies) for S:

$$\begin{aligned} f_6 &= z^5 - x^4, \\ e_6 &= ze_5 - xe_4, \\ s_6 &= ye_6 - z^2e_3 + x^2e_1. \end{aligned}$$

Now $P = \{xe_5, ye_6, xe_6, z^2e_4, xze_4, x^2e_4\}$ and when we check xe_5 by the criteria we see that $\mathrm{lm}(s_5) \mid xe_5$ and because of this, we can reject this element and go on with ye_6. This element is also divisible by $\mathrm{lm}(s_6)$ and deleted. Fortunately, xe_6 gives rise to a new element:

$$\begin{aligned} f_7 &= -x^5 + z^2, \\ e_7 &= xe_6 - z^3e_2 - ye_3 - e_1, \\ s_7 &= z^2e_7 + x^4e_2 + x^2ye_1 + z^2e_1 (\text{from } s_{6,2}). \end{aligned}$$

Two new elements are added to $P = \{z^2e_4, xze_4, x^2e_4, ye_6, z^2e_6\}$, z^2e_4 is detected to be useless by $\mathrm{lm}(s_4) = z^2e_4$. So the computations for xzf_4 follow:

$$\begin{aligned} f_8 &= y^5 + x^4z, \\ e_8 &= xze_4 - y^3e_2, \\ s_8 &= ze_8 - y^2e_4 + x^3e_2 (\text{from } s_{4,2}). \end{aligned}$$

This adds the elements ze_8 and xe_8 to P, but the next element in the row is x_2e_4. For detecting its uselessness some further known syzygy can be used: Combining s_5 and the the syzygy $e_6 - ze_5 - xe_4$ we get

$$\begin{aligned} s' &= x(e_6 - ze_5 + xe_4) - z\underbrace{(xe_5 + y^2e_1 - z^2e_2)}_{s_5} \\ &= x^2e_4 + y^2ze_1 + e_7 + ye_3 + e_1. \end{aligned}$$

Whereas ze_8 is also detected by s_8, xe_8 is not detected at all and reduces to zero. The remaining two elements in P, ye_7 and z^2e_7 are discarded by syzygies in S and thus the computations stops with the Gröbner basis

$$G = \{x^2y - z^2, xz^2 - y^2, yz^3 - x^2, y^3z - x^3, xy^3 - z^4, z^5 - x^4, x^5 - z^2, y^5 - x^4z\}.$$

During this computation only 1 zero reduction has happened (xf_8). As noted in [126], a Gebauer–Möller implementation would compute 7 zero reductions for this example.

Remark 3.3.6. Let us comment shortly on the fact why Example 3.3.5 is exactly the same as the one given in [126] and [62]: First of all the example is well–suited for showing the main ideas, both for the syzygy and for the signature–based attempt. Moreover, Faugère used this example in [62] to show the different behaviour of the F5 Algorithm in comparison to SyzStd. In an even more important situation, namely the discussion of termination of the F5 Algorithm, we can use this example to show the differences between several signature–based standard basis algorithms. Thus deciding to state exactly this example again is justified and reasonable.

At this point we are ready to enter the world of signature–based algorithms. It will turn out that the main step from the syzygy–based attempt to the signature–based one has been already mentioned in Remark 3.3.3 (4): On the one hand, try to keep the overhead caused by the lengthy computations of the syzygies as small as possible, and, on the other hand, keep the range of the criteria to detect useless elements as big as possible.

4 An introduction to signature–based standard basis algorithms

Although the starting point of signature–based standard basis algorithms can be found in [126], as already mentioned in Chapter 3, the first "real" algorithm based thoroughly on signatures is Faugère's F5 Algorithm presented in [62].

Also this is the source of nearly everything presented in the following of this thesis, it is not the best point to start with: The F5 Algorithm is a rather "aggressive" implementation of the main ideas behind signature–based computations.

On the other hand, the G2V Algorithm by Gao, Guan, and Volny, see [76], has a straightforward implementation, but it lacks performance. Note that there are some rumours about G2V being multiple times faster than F5. We have an in–depth discussion on this topic, in which we do not only compare both algorithms with each other, but also show comparable implementations of both.

The main new idea behind the usage of signatures is to introduce new criteria to detect useless critical pairs during a standard basis computation. Instead of using Buchberger's criteria by checking the leading monomials of critical pairs, we take the signature of an element into account. In some sense, defined in detail in Section 4.1, one can ask for the *minimal* signature of an element. Keeping only those elements, whose corresponding signatures are minimal, leads to a high–performance standard basis algorithm, which in some specialized, but still rather usual setting does not compute any zero reduction at all.

Note that until now the signature–based world of standard basis computations is limited to the computation of bases for ideals in $\mathcal{P}$. This is due to the fact that one does not have essential structures like *principal syzygies* in the world of modules (since $\mathcal{P}$ is an $\mathcal{K}$–algebra, whereas $\mathcal{P}^s$, for $s > 1$, is not). Thus in the following when speaking about computations and algorithms, we always work in the polynomial setting.

This chapter has to be understood as an introduction to the topic, presenting the foundations for more efficient implementations discussed in detail in the following chapters. It is structured in the following way:

(1) In Section 4.1 we give the definition of signatures. Instead of copying the already known, but also sometimes differing definitions, we give a more general one, which give us more flexibility. It turns out that the usual signature, as defined in [62], is just a special case of our definition.

(2) Having some knowledge about signatures we give a generic signature–based standard basis algorithm in Section 4.2. This algorithm is the counterpart to Std given in Section 1.8, as it gives just the general structure, but does not deliver an efficient algorithm. Using it to explain the basic ideas of the signature–based criteria to reject useless critical pairs builds a ground for understanding highly optimized implementations as F5 or G2V in the following.

(3) We finish this chapter with an in–depth discussion on various restrictions of the reduction process in signature–based algorithms. Based on this an example computation of the generic algorithm is given.

With the ideas of this chapter in mind signature–based criteria to detect useless critical pairs can be understood much easier.

4.1 Basic ideas behind signatures and labeled polynomials

In this section we give the definition of a signature of a polynomial $g \in I = \langle f_1, \ldots, f_k \rangle$, an ideal in $\mathcal{P}$. Doing this we point out the connections to syzygies (see Chapter 3). It turns out that a signature is nothing else but a part of a module element $h \in \mathcal{P}^k$ corresponding to g.

Besides defining signatures for the initial generators $f_1, \ldots, f_k$, we also show how s–vectors, generated during the standard basis computation, are equipped with signatures.

Note that this is the first time signatures are defined in such a general way. The benefits and drawbacks of using different variants of signatures are explained in more detail in the following.

Definition 4.1.1. Let $F = \{f_1, \ldots, f_k\}$ be a finite subset in $\mathcal{P}$, $I = \langle F \rangle$ be a finitiely generated ideal in $\mathcal{P}$, and let $e_1, \ldots, e_k$ be the canonical generators of $\mathcal{P}^k$ such that

$$\begin{array}{rccl} \pi: & \mathcal{P}^k & \longrightarrow & I \\ & e_i & \longmapsto & f_i \text{ for all } 1 \le i \le k \end{array}$$

is a surjective module homomorphism. Let $<$ be a well–order on $\mathcal{P}^k$, and let $g \in I$, $h \in \mathcal{P}^k$. We define the set of all *labels of g* by

$$\text{labels}(g) = \left\{ h \in \mathcal{P}^k \mid \pi(h) = g \right\}.$$

It is clear that, by construction, for any element $g \in I = \langle f_1, \ldots, f_k \rangle$ there exists an element $h \in \mathcal{P}^k$ such that $\pi(h) = g$. The crucial point is that there exist infinitely many such elements h.

Example 4.1.2. Let $I = \langle f_1, f_2 \rangle$ be an ideal in $\mathcal{K}[x, y, z]$ with $<_{\text{dp}}$ where

$$\begin{aligned} f_1 &= xy + x, \\ f_2 &= y^2 - 1. \end{aligned}$$

Moreover, let e_1 and e_2 be the canonical generators of $\mathcal{P}^2$ equipped with $<_{\text{i}}$. In this setting we easily construct the labels:e_1 for f_1 and e_2 for f_2. Besides them we can construct infinitely many different labels, for example by adding the principal syzygy $f_1 e_2 - f_2 e_1$ to the corresponding labels:

$$\begin{aligned} p(f_1 e_2 - f_2 e_1) + e_1 &\in \text{labels}(f_1), \\ q(f_1 e_2 - f_2 e_1) + e_2 &\in \text{labels}(f_2). \end{aligned}$$

where $p, q \in \mathcal{P}$ are some polynomials. Even other, not so obvious, labels can be easily constructed:

$$x e_2 - y e_1 \in \text{labels}(f_1) \text{ as } f_1 = y f_1 - x f_2.$$

We conclude that for any element $f \in I$ there exist infinitely many different $h \in \mathcal{P}^2$ such that $\pi(h) = f$.

Most of the time, we are only interested in some special part of the labels, namely the so–called signatures:

Definition 4.1.3. Let the setting be the same as in Definition 4.1.1. Furthermore, let $<$ be a well–order on $\mathcal{P}^k$, and let $g \in I$, $h \in \mathcal{P}^k$.

(1) The *signature of h* is defined by $\text{sig}(h) = \text{lt}_<(h)$.

(2) The *set of all signatures of g* is given by

$$\operatorname{signatures}(g) := \left\{ \operatorname{sig}(h) \mid h \in \operatorname{labels}(g) \right\}.$$

(3) The *(minimal) signature of g* is denoted

$$\operatorname{sig}(g) := \operatorname{sig}\left(\min_{<} \operatorname{labels}(g) \right).$$

Definition 4.1.4. We call an element $r = (l, g) \in \mathcal{P}^k \times \mathcal{P}$ a *labeled polynomial of g* iff $\operatorname{sig}(l) \in \operatorname{signatures}(g)$. Moreover, we define

(1) the *polynomial part of r*, $\operatorname{poly}(r) = g$,

(2) the *label of r*, $\operatorname{label}(r) = l$, and

(3) the *signature of r*, $\operatorname{sig}(r) = \operatorname{sig}(l)$.

We define the relation between two labeled polynomials f and g in $\mathcal{P}^k \times \mathcal{P}$ by

$$f = g :\Longleftrightarrow \operatorname{label}(f) = \operatorname{label}(g) \text{ and } \operatorname{poly}(f) = \operatorname{poly}(g).$$

Remark 4.1.5.

(1) Our definition of a labeled polynomial has the key advantage to previously given definitions: We only assume $\operatorname{sig}(l) \in \operatorname{signatures}(g)$. Thus l can be anything between a complete label h of g, i.e. from $\pi(l) = g$, to $l = \operatorname{sig}(h)$. This allows much more flexibility and is pretty useful when distinguishing between theoretical and practical aspects as well as generalizing ideas. See Section 4.3 for more information on this topic.

(2) Note that whereas we have seen in Example 4.1.2 that there exist infinitely many different elements $h \in \mathcal{P}^k$ such that $\pi(h) = g$ for $g \in I$, $\operatorname{sig}(r)$ is uniquely defined for a labeled polynomial r. Assume $h_1 \neq h_2 \in \mathcal{P}^k$ such that

$$\pi(h_1) = g = \pi(h_2).$$

This defines *two different* labeled polynomials:

$$r_1 = \left(h_1, g\right), \text{ and}$$
$$r_2 = \left(h_2, g\right).$$

From the point of view of labeled polynomials it is clear that if $\operatorname{lt}(h_1) \neq \operatorname{lt}(h_2)$, then $\operatorname{sig}(r_i) = \operatorname{lt}(h_j)$ if and only if $i = j$ for $1 \leq i, j \leq 2$.

(3) The reader should be careful with our definition of a labeled polynomial, which is also a generalization of other definitions in the literature. For a labeled polynomial $r = (l, p)$ we only assume that

$$\operatorname{sig}(l) \in \operatorname{signatures}(p),$$

whereas a labeled polynomial as defined in [57–59, 144] fulfills the inclusion

$$l \in \mathrm{signatures}(p).$$

In some situations it is quite useful to have more data stored in l besides the signature of r in theoretical considerations. But also in practice this can have a positive effect on the efficiency of the implementation. We give a deeper insight in this topic in Section 7.3.

(4) Our definition of a signature is more general than the ones given in any publication about signature–based algorithms, e.g. [62, 76]:

a) In Section 7.3 we generalize our definition of signatures, giving them different lengths. The signatures presented in Definitions 4.1.3 and 4.1.4 are a special case, namely a signature of length 1. No longer consisting of a module leading term only, one can try to reduce two signatures for a length $j > 1$. This idea is first mentioned in [126], see also Remark 3.3.3 (4).

b) Secondly, we define $\mathrm{sig}(r)$ to be a leading term of an element in $\mathcal{P}^k$, i.e. we allow coefficients in our signatures. This is different to the signatures introduced in [62], but fits to the corresponding definition in [76]. So the leading monomial of $\mathrm{sig}(r)$ corresponds to Faugère's signature. We see in the following, when comparing G2V and F5, that this distinction need not been made when using only signatures of level 1. Using more general signatures of length $j > 1$ we need to take care of the coefficients.

From Remark 4.1.5 one could conclude that Definitions 4.1.1 and 4.1.4 are quite useless and that the notion of a labeled polynomial is more or less a tautological expression. The real idea behind a labeled polynomial $r = (l, p)$ gets clear with Definition 4.1.3: We have infinitely many different labeled polynomials r of a polynomial p, but we are searching for those with $\mathrm{sig}(l) = \mathrm{sig}(p)$.

Example 4.1.6. Reconsidering Example 4.1.2 we can easily construct the labeled polynomials $r_1 = (e_1, f_1)$ and $r_2 = (e_2, f_2)$. Besides them we can construct infinitely many different labeled polynomials, for example by adding the principal syzygy $f_1 e_2 - f_2 e_1$ to the corresponding labels:

$$\begin{aligned} r_{1,p} &= \big(p(f_1 e_2 - f_2 e_1) + e_1, f_1\big), \\ r_{2,q} &= \big(q(f_1 e_2 - f_2 e_1) + e_2, f_2\big). \end{aligned}$$

where $p, q \in \mathcal{P}$ are some polynomials. Defining

$$r_1' = (xe_2 - ye_1, f_1) \text{ as } f_1 = yf_1 - xf_2,$$

as already seen above, r_1' is again a different labeled polynomial of f_1

Reviewing Example 4.1.2 we see that $\mathrm{sig}(f_i) = \mathrm{sig}(r_i) = e_i$ for $1 \leq i \leq 2$. Having a closer look at $\mathrm{sig}(f_1) = e_1$, on the one hand, we get that $\mathrm{sig}(r_1) = \mathrm{sig}(r_{1,p})$ if and only if $p = 0$. Whenever $p \neq 0$ $\mathrm{sig}(r_{1,p}) \succ \mathrm{sig}(r_1)$. On the other hand, comparing r_1 and r_1' it clearly holds that $\mathrm{sig}(r_1) \prec \mathrm{sig}(r_1')$. Certainly, all of these signatures are elements of $\mathrm{signatures}(f_1)$

Remark 4.1.7.

(1) By ensuring $<$ to be a well–order on $\mathcal{P}^k$ the minimal signature of a polynomial p is uniquely defined. Thus we can find a labeled polynomial $r = (l, p)$ with $\operatorname{sig}(l) = \operatorname{sig}(p)$.

(2) Let us give a short outlook on why we are doing this: In the following we show that whenever a signature–based algorithm wants to find out if a critical pair is useful or not, it just checks the corresponding signatures. In the algorithm we always consider labeled polynomials, not the polynomials itself. Thus the question if the label of r is also the label of $\operatorname{poly}(r)$ arises. If this is not the case, then such an element need not be computed at all.

(3) The labels of those labeled polynomials we construct in our algorithms are strongly related to the reduction process during a standard basis computation. Moreover, the initial labels are predefined by the input of our algorithms. This prevents an ambiguity of different labels with equal leading terms for one polynomial to appear and it enables a strong criterion for detecting useless critical pairs.

As we want to consider labeled polynomials in our algorithms we need to define some notions, important when computing standard bases, also for labeled polynomials.

First of all let us make notation easier with the following:

Definition 4.1.8. Let $I = \langle f_1, \ldots, f_k \rangle$ be an ideal in $\mathcal{P}$, $p \in I$, let $f = (l, p)$ be a labeled polynomial in $\mathcal{P}^k \times I$, and let π as defined before.

(1) We extend the following operators:

   a) $\operatorname{lc}(f) = \operatorname{lc}(p)$,

   b) $\operatorname{lm}(f) = \operatorname{lm}(p)$,

   c) $\operatorname{lt}(f) = \operatorname{lt}(p)$, and

   d) $\deg(f) = \deg(p)$.

   Moreover, assuming a second labeled polynomial $g = (t, q)$ we define

   e) $\tau(f, g) = \tau(p, q)$.

(2) Some special parts of the label of f are of interest for us:

   a) The *index of f* is denoted $\operatorname{index}(f) = \operatorname{index}\left(\operatorname{lt}(l)\right)$.

   Moreover, assuming $\operatorname{lt}(l) = a\lambda e_i$, $a \in \mathcal{K}$, λ a monomial in $\mathcal{P}$ we can define

   b) the *leading monomial of the label of f*, $\operatorname{siglm}(f) = \lambda e_i$,

   c) the *monomial part of the signature of f*, $\operatorname{slm}(f) = \lambda$,

   d) the *coefficient of the signature of f*, $\operatorname{slc}(f) = a$,

   e) the *term of the signature of f*, $\operatorname{slt}(f) = a\lambda$, and

   f) the *degree of the signature of f* $\operatorname{sig-deg}(f) = \deg(\lambda) + \deg\left(\pi(e_i)\right)$.

(3) Last we define multiplications with labeled polynomials. Let $m \in \mathrm{Mon}(x_1, \ldots, x_n)$, $b \in \mathcal{K}$, then

a) $br = (bl, bp)$,

b) $mr = (ml, mp)$.

In Definition 1.7.12 (3) we have defined the standard representation of a polynomial f w.r.t. some finite set of polynomials G. This definition is crucial for the notion of a normal form, which is the main tool computing standard bases. As a matter of fact, we need to introduce such a representation also for labeled polynomials.

Definition 4.1.9. Let $I = \langle f_1, \ldots, f_k \rangle$ be an ideal in $\mathcal{P}$ equipped with $<$, let $r, r_1, \ldots, r_l \in \mathcal{P}^k \times I$ be labeled polynomials, and let $G = \{r_1, \ldots, r_l\}$. Moreover, let $\prec$ be a well–order on $\mathcal{P}^k$. We say that r has a *standard representation w.r.t.* G if there exist polynomials $p_1, \ldots, p_l \in \mathcal{K}[x_1, \ldots, x_n]$ and a unit $u \in \mathcal{P}$ such that

$$u\,\mathrm{poly}(r) = \sum_{i=1}^{l} p_i\, \mathrm{poly}(r_i),$$

where

$$\begin{aligned} \max_{<}\left\{\mathrm{lm}(p_i)\,\mathrm{lm}(r_i)\right\} &\leq \mathrm{lm}(r), \qquad \text{and} \\ \max_{\prec}\left\{\mathrm{lm}(p_i)\,\mathrm{siglm}(r_i)\right\} &\preceq \mathrm{siglm}(r). \end{aligned}$$

Remark 4.1.10. The standard representation of a labeled polynomial r has two properties:

(1) $\mathrm{poly}(r)$ has a standard representation w.r.t. $\{\mathrm{poly}(r_1), \ldots, \mathrm{poly}(r_l)\}$.

(2) The signatures of the multiples of the r_i are not greater than the signature of r.

This second property makes the standard representation of a labeled polynomial more restrictive than that of a polynomial.

For the rest of this section we always assume the following setting: Let $I = \langle f_1, \ldots, f_k \rangle$ be an ideal in $\mathcal{P}$, and let $p, q \in I$ be two polynomials such that $a\lambda e_i \in \mathrm{signatures}(p)$ and $b\sigma e_j \in \mathrm{signatures}(q)$.

The following properties of signatures are straightforward by their definition.

Proposition 4.1.11. *Let $c \in \mathcal{K}$, and let $m \in \mathrm{Mon}(x_1, \ldots, x_n)$. Then the following hold:*

(1) $a\lambda e_i \in \mathrm{signatures}(p \pm q)$, *if* $\lambda e_i \succ \sigma e_j$.

(2) $(a \pm b)\lambda e_i \in \mathrm{signatures}(p \pm q)$, *if* $\lambda e_i = \sigma e_j$ *and* $a \pm b \neq 0$.

(3) $cam\lambda e_i \in \mathrm{signatures}(cmp)$.

These properties we want to use for defining a reduction process for labeled polynomials similar to the one for usual polynomials defined in Section 1.7.

Corollary 4.1.12. *Suppose that there exist $c \in \mathcal{K}$ and $m \in \operatorname{Mon}(x_1, \ldots, x_n)$ such that* $\operatorname{lt}(p) = cm \operatorname{lt}(q)$. *Then the following hold:*

$$\text{If } m\sigma e_j < \lambda e_i, \text{ then } a\lambda e_i \in \operatorname{signatures}(p - cmq). \tag{4.1.1}$$

$$\text{If } m\sigma e_j = \lambda e_i \text{ and } a \neq bc, \text{ then } (a - bc)\lambda e_i \in \operatorname{signatures}(p - cmq). \tag{4.1.2}$$

This in mind one can define a reduction process for labeled polynomials in a quite natural manner:

Definition 4.1.13. Let $f = (s, p)$ and $g = (t, q)$ be two labeled polynomials such that $\operatorname{sig}(f) = a\lambda e_i$ resp. $\operatorname{sig}(g) = b\sigma e_j$. Suppose that there exist $c \in \mathcal{K}$ and $m \in \operatorname{Mon}(x_1, \ldots, x_n)$ such that $cm \operatorname{lt}(f) = \operatorname{lt}(g)$. Then the following hold:

(1) We say that $p - cmq$ is a *sig–safe reduction of p w.r.t. q* iff $(a\lambda e_i, p)$ and $(b\sigma e_j, q)$ satisfy either (4.1.1) or (4.1.2). Otherwise, the reduction $p - cmq$ is called *sig–unsafe*.

(2) Let $G = \{r_1, \ldots, r_l\}$ be a finite set of labeled polynomials in $\mathcal{P}^k \times I$, and let r be another labeled polynomial. We say that the *reduction of r w.r.t. G is sig–safe* iff for each $j \in \{1, \ldots, l\}$ the reduction of r (possibly already by other elements of G sig–safe reduced) by r_j is sig–safe.

(3) A sig–safe reduction is called *complete* if reductions satisfying (4.1.1) and (4.1.2) are allowed.

(4) A sig–safe reduction is called *semi–complete* if only reductions satisfying (4.1.1), but not those satisfying (4.1.2) are allowed.

Note that in Definition 4.1.13 we have made use of our more general definition of labeled polynomials. We have defined two labeled polynomials, whose labels are just the signatures.

The intention of defining such a sig–safe reduction is to keep the main values of the signature of a labeled polynomial r, namely the index $\operatorname{index}(r)$ and the monomial part $\operatorname{slm}(r)$ constant. This means that we allow no reduction with an element r' of a "really" higher signature, i.e. where $\operatorname{siglm}(r') > \operatorname{siglm}(r)$. This sounds strange thinking about termination and correctness for a standard basis algorithm, but in Section 4.2 we see that a signature–based algorithm has some fallback functionality to cope with these not allowed reduction steps.

As a last step in preparation of a first generic framework of a signature–based standard basis algorithm we need to define critical pairs resp. s–vectors of labeled polynomials.

Definition 4.1.14. Let f and g in $\mathcal{P}^k \times \mathcal{P}$ be two labeled polynomials. Let

$$u = \operatorname{lc}(g)\frac{\tau(f,g)}{\operatorname{lm}(f)}, v = \operatorname{lc}(f)\frac{\tau(f,g)}{\operatorname{lm}(g)}.$$

(1) Assume that $\operatorname{lm}\left(u \operatorname{label}(f)\right) \neq \operatorname{lm}\left(v \operatorname{label}(g)\right)$.

   a) We define the *s–vector of f and g by*

$$\mathcal{S}(f,g) = \left(u \operatorname{label}(f) - v \operatorname{label}(g), u \operatorname{poly}(f) - v \operatorname{poly}(g)\right).$$

b) We call the tuple (uf, vg) a *critical pair of the labeled polynomials f and g*. The *degree of the critical pair* (uf, vg) is defined to be $\deg(\tau(f,g))$.

(2) Conversely, if $\operatorname{lm}\left(u \operatorname{label}(f)\right) = \operatorname{lm}\left(v \operatorname{label}(g)\right)$ we say that (uf, vg) *is sig–equivalent*.

(3) In more general, for any two terms $\lambda, \sigma \in \mathcal{P}$ we define the notation

$$\lambda f - \sigma g := \left(\lambda \operatorname{label}(f) - \sigma \operatorname{label}(g), \lambda \operatorname{poly}(f) - \sigma \operatorname{poly}(g)\right).$$

Remark 4.1.15.

(1) We use the shorthand notation of Part (3) of Definition 4.1.14 to generalize our definition of a sig–(un–)safe reduction, speaking now of $f - cmg$ instead of $p - cmq$ where $\operatorname{poly}(f) = p$ and $\operatorname{poly}(g) = q$.

(2) The definition of a critical pair of two labeled polynomials differs slightly from those of usual polynomials given in Definition 1.8.6. Here we explicitly store the multipliers for the s–vector computation, too. This is due to the fact that those multipliers are also used to get the signature of the corresponding s–vector. Moreover, signature–based algorithms are depending on a special order, in which the critical pairs have to be handled, namely by increasing signatures (see Algorithm 33, Line 12). This selection is done before the s–vector itself is constructed, thus the data of the multipliers are important to be stored in the critical pair.

In the signature–based world we work with labeled polynomials, but in the end our single interest is the standard basis G of some polynomial ideal. Thus it makes sense, in terms of presenting pseudo codes, for example in Algorithms 32 , 33, and 34, to define a shorthand notation for the polynomial part of a set of labeled polynomials.

Definition 4.1.16. Let $G = \{r_1, \ldots, r_l\}$ be a set of labeled polynomials. We denote the *polynomial ideal of G* by

$$\operatorname{poly}(G) := \left\{ \operatorname{poly}(r_1), \ldots, \operatorname{poly}(r_l)\right\}.$$

We can easily adopt Buchberger's Criterion from the classical, polynomial situation to the labeled polynomial one here.

Proposition 4.1.17. *Let $<$ be a monomial order on $\mathcal{P}$, and let $G = \{g_1, \ldots, g_s\}$ be a set of labeled polynomials. Moreover, let $I = \langle f_1, \ldots, f_k\rangle$ be an ideal in $\mathcal{P}$ such that $\{f_1, \ldots, f_k\} \subset$* $\operatorname{poly}(G)$. *If for each pair $(g_i, g_j) \in G \times G$ with $i > j$ $\mathcal{S}(g_i, g_j)$ has a standard representation w.r.t. G, then* $\operatorname{poly}(G)$ *is a standard basis of I.*

Proof. For each $\mathcal{S}(g_i, g_j)$ having a standard representaion w.r.t. G $\mathcal{S}\left(\operatorname{poly}(g_i), \operatorname{poly}(g_j)\right)$ has a standard representation w.r.t. $\operatorname{poly}(G)$. As $\operatorname{poly}(G) \subset I$ the statement follows by Theorem 1.8.3. □

In first place this does not make sense at all: Why do we require a stronger variant of standard representation on the s–vectors than we even need to? Quite similar to our discussions in Chapter 2 we show that based on the statement of Proposition 4.1.17 we find criteria to narrow down the number of labeled s–vectors really that need to be verified having a standard representation.

Corollary 4.1.18. *Let $<$ be a monomial order on $\mathcal{P}$, and let $G = \{g_1, \ldots, g_s\}$ be a set of labeled polynomials. Moreover, let $I = \langle f_1, \ldots, f_k \rangle$ be an ideal in $\mathcal{P}$ such that $\{f_1, \ldots, f_k\} \subset \operatorname{poly}(G)$. For each pair $(g_i, g_j) \in G \times G$ with $i > j$ let*

$$t_k = \operatorname{lc}(g_l)\frac{\tau(g_i, g_j)}{\operatorname{lm}(g_k)}$$

for $k \in \{i, j\}, l \in \{i, j\} \backslash \{k\}$. If for each such pair (g_i, g_j) either

(1) $\operatorname{lm}(t_i)\operatorname{siglm}(g_i) = \operatorname{lm}(t_j)\operatorname{siglm}(g_j)$, *or*

(2) $l = t_i \operatorname{label}(g_i) - t_j \operatorname{label}(g_j)$ *and* $\big(l, t_i \operatorname{poly}(g_i) - t_j \operatorname{poly}(g_j)\big)$ *has a standard representation w.r.t. G,*

then $\operatorname{poly}(G)$ *is a standard basis of I.*

Proof. By Definition 4.1.9 all pairs (g_i, g_j) fulfilling (2) of the above statement, generate labeled polynomials $r = \big(l, t_i \operatorname{poly}(g_i) - t_j \operatorname{poly}(g_j)\big)$ such that $\operatorname{poly}(r)$ has a standard representation w.r.t. $\operatorname{poly}(G)$. Thus it remains to show that all pairs (g_i, g_j) that meet (1) lead to s–vectors, which have a standard represenation, at least of their polynomial part w.r.t. $\operatorname{poly}(G)$.

For this, let P be the set of all pairs (g_i, g_j) and delete all those fulfilling (2). Now order the remaining pairs by increasing $\operatorname{lm}\big(t_i \operatorname{label}(g_i)\big) = \operatorname{lm}\big(t_j \operatorname{label}(g_j)\big)$. Let (f, g) be the pair minimal by this order, let u, v be the corresponding terms such that $u \operatorname{lt}(f) = v \operatorname{lt}(g)$, and let $a = \operatorname{lc}\big(\operatorname{label}(f)\big), b = \operatorname{lc}\big(\operatorname{label}(g)\big)$. Assume the labeled polynomial

$$r = \left(u \operatorname{label}(f) - \frac{a}{b} v \operatorname{label}(g), u \operatorname{poly}(f) - \frac{a}{b} v \operatorname{poly}(g)\right).$$

By construction it holds that $\operatorname{siglm}(r) \prec \operatorname{lm}\big(u \operatorname{label}(f)\big)$, thus r has a standard representation w.r.t. G. If $a = b$ then r is just the s–vector of f and g and we are done. Otherwise we have a closer look at the standard representation of r:

$$\operatorname{poly}(r) = \sum_{k=1}^{s} p_k \operatorname{poly}(g_k), \quad p_k \in \mathcal{P}$$

where $\operatorname{lm}\big(p_k \operatorname{label}(g_k)\big) \prec \operatorname{lm}\big(u \operatorname{label}(f)\big)$ for all k. From this we can conclude that any two summands of the same leading term correspond to an s–vector which has a standard representation. Exchanging all those matching leading terms by the corresponding standard representation w.r.t. G we find some element $h \in G$ and some term $w \in \mathcal{P}$ such that

$$u \operatorname{lt}\big(\operatorname{poly}(f)\big) = v \operatorname{lt}\big(\operatorname{poly}(g)\big) = w \operatorname{lt}\big(\operatorname{poly}(h)\big).$$

This means that we can find $\lambda_1, \lambda_2 \in \mathcal{P}$ such that

$$u \operatorname{poly}(f) - v \operatorname{poly}(g) = \lambda_1 \mathcal{S}\big(\operatorname{poly}(f), \operatorname{poly}(h)\big) - \lambda_2 \mathcal{S}\big(\operatorname{poly}(h), \operatorname{poly}(g)\big).$$

By construction, $\mathcal{S}(f, h)$ as well as $\mathcal{S}(h, g)$ have a standard representation w.r.t. G. From this the statement follows. $\square$

Remark 4.1.19.

(1) Whereas during the reduction of a labeled polynomial we do not allow sig–unsafe reductions, the construction of the s–vector itself seems to contradict this idea looking at Definition 4.1.14: Assume that $\mathrm{sig}(r_1) > \mathrm{sig}(r_0)$. It is possible that $t_1 \,\mathrm{sig}(r_1) < t_0 \,\mathrm{sig}(r_0)$. Whereas a signature–based algorithm can handle the suppressed reductions quite nicely, there is no real alternative in the s–vector situation: We have to build $\mathcal{S}(r_1, r_0)$ to ensure the correctness of the standard basis in the end. One could cope with this situation by discarding $\mathcal{S}(r_1, r_0)$ and generating the new s–vector $\mathcal{S}(r_0, r_1)$. Clearly, $\mathcal{S}(r_0, r_1)$ gets the correct signature, but those two labeled s–vectors differ only by a multiplication with the unit −1. Thus, in an implementation a benefit from keeping even the s–vector generation sig–safe is missing, and much less problematic compared to the computational overhead of discarding one s–vector and generating another one.

(2) Note that we have not defined an s–vector of two labeled polynomials g, h if the multiplied signatures coincide. To understand this, we need to think in a more general setting: Generating the s–vector $u \,\mathrm{poly}(g) - v \,\mathrm{poly}(h)$ on the polynomial part, this would also mean to subtract the corresponding multiplied labels of the two polynomials. From this point of view, at least when $u \,\mathrm{sig}(g) = v \,\mathrm{sig}(h)$ it is clear that the computation on the labels would cancel out the leading terms $u \,\mathrm{sig}(g)$ resp. $v \,\mathrm{sig}(h)$, and end up with a completely new leading term. Clearly, this cannot be handled on the level of labeled polynomials, having only the signatures stored in such an s–vector. If $\mathrm{lm}(u) \,\mathrm{siglm}(g) = \mathrm{lm}(v) \,\mathrm{siglm}(h)$ on the one hand, but $u \,\mathrm{sig}(g) \neq v \,\mathrm{sig}(h)$ on the other hand, this means that the multiplied signatures differ only by some constant. In this situation the signature of the corresponding s–vector can be defined. Certainly, we see in Section 4.2 that those sig–equivalent critical pairs are not needed if we want to compute standard bases.

    Moreover, this problem of vanishing signatures when building s–vectors is one of the main tasks we want to handle a bit more dynamically generalizing signatures in Section 7.3.

4.2 A generic signature–based standard basis algorithm

This section can be understood as the signature–based counterpart to Section 1.8. Here we present a generic standard basis algorithm based on signatures to detect useless critical pairs. The key points are:

(1) Use labeled polynomials instead of usual ones.

(2) Use the signatures of these elements to reject useless computations.

(3) Use a sig–safe reduction process to keep the signatures and regardlessly manage to retain the correctness of the algorithm computing a standard basis.

The idea is to give an easy introduction to the behaviour of signature–based algorithms with a framework, which does not focus on efficiency, but comprehensibility. Moreover, the structure of the algorithm is kept generic enough such that all later on presented efficient implementations, e.g. F5 or G2V, can be easily derived from it.

Nearly all of the ideas presented in this section are already published in [59], which is a collaboration with John Perry.

We need some notion for a situation occurring in signature–based algorithms, which seems to be very strange in the first place, as they cannot occur in the usual polynomial setting. It can happen that some labeled s–vector reduces to a labeled polynomial r such that $\operatorname{poly}(r) \neq 0$, but r is useless.

Definition 4.2.1. Let $I = \langle f_1, \ldots, f_k \rangle$ be an ideal in $\mathcal{P}$, let r be a labeled polynomial, and let G be a finite set of labeled polynomials in $\mathcal{P}^k \times I$. If there exists some $g \in G$ such that

$$\operatorname{siglm}(g) \mid \operatorname{siglm}(r) \quad \text{and} \quad \operatorname{lm}(g) \mid \operatorname{lm}(r),$$

then r is called to be *sig–redundant (w.r.t. G)*.

Algorithm 32 Generic signature–based standard basis computation w.r.t. $<$ (SigStd)

Input: $F = \{f_1, \ldots, f_r\}$ a finite subset of $\mathcal{P}$
Output: G a standard basis for $\langle F \rangle$ w.r.t. $<$

$G_1 \leftarrow \{f_1\}$
for $(i = 2, \ldots, r)$ **do**
  $f_i \leftarrow \text{Reduce}(f_i, G_{i-1})$
  if $(f_i \neq 0)$ **then**
    $G_i \leftarrow \text{IncSig}(f_i, G_{i-1})$
  else
    $G_i \leftarrow G_{i-1}$
$G \leftarrow G_r$
return G

As one can see we have split up the generic signature–based standard basis algorithm into 3 parts:

(1) Algorithm 32 is nothing else but the main loop, which goes over all elements f_i in the generating set F of the ideal I. The only computation done is the reduction of f_i w.r.t. already known standard basis G_{i-1} of $\langle f_1, \ldots, f_{i-1} \rangle$. Clearly, if the reduced $f_i = 0$, then we go on with the next element.

(2) SigStd calls Algorithm 33, in which the computation of a standard basis G_i for the ideal generated by $f_1, \ldots, f_i$ is done.

(3) The reduction process itself is out–sourced in Algorithm 34. For every previously constructed s–vector it computes a semi–complete sig–safe reduced labeled polynomial.

Looking at this description, even without thinking about labeled polynomials, a rather obvious difference to Std can be seen: The standard basis G is computed *incrementally*. Let us give a short explanation of this:

Instead of computing a standard basis G for $\langle f_1, \ldots, f_r \rangle$ at once, it is computed piecewise: We know that $G_1 = \{f_1\}$ is a standard basis for $\langle f_1 \rangle$. With this information stored we can compute the standard basis G_i for $\langle f_1, \ldots, f_i \rangle$ for $2 \le i \le r$ recursively:

- ▷ We know that $\langle f_1, \ldots, f_i \rangle = \langle f_i, g \mid g \in G_{i-1} \rangle$.
- ▷ We can start with $G_i = G_{i-1} \cup \{f_i\}$, building only those critical pairs of f_i with the other $g \in G_i$; any critical pair $(t_i g_i, t_j g_j)$ leads to an s–vector that reduces to zero w.r.t. G_i due to the fact that G_{i-1} is already a standard basis for itself.
- ▷ Doing the usual computation steps known from Std we end up with G_i being the standard basis for $\langle f_1, \ldots, f_i \rangle$.

At the end of the above described process, we compute G_r, which is the standard basis for $I = \langle f_1, \ldots, f_r \rangle$.

Remark 4.2.2. It is important to note that signature–based algorithms do not necessarily need an incremental framework as given by Algorithm 32. The reasons why we present our first, introductory signature–based algorithm in this fashion are:

(1) The two most famous and widest spread implementations of algorithms using signatures, namely F5 and G2V, are based on an incremental frame. As the focus of this thesis is to explain and to compare those two algorithms in detail, it is, in an educational manner, best to explain both as optimizations of the generic algorithm presented in this section. This makes it easier for the reader to understand their peculiarities.

(2) The reason why F5 and G2V are given in an incremental fashion is that both are based on the order $<_{\mathrm{i}}$ on the set of signatures. This means that the index has a higher priority than the monomial. From this point of view it makes sense to compute G incrementally, as in Line 12 we see that the order in which the critical pairs are handled is by increasing signature. Thinking about any critical pair being generated by elements out of $\langle f_1, f_2 \rangle$ the index of corresponding signature is 2. Any pair built with f_3 must have a signature of at least index 3, and is thus always stored, but never processed further until all critical pairs of index 2 have been computed.

Clearly, thinking geometrically about ideals of intersections being not complete, i.e. ideals generated by more polynomials than the number of variables in the corresponding polynomial ring, the behaviour of an incremental standard basis algorithm can be quite bad and preformance can suffer a lot from computing step by step and not using all data at once. In the signature–based world this means that other monomial orders on the signatures must be taken into account. First steps in this direction are already taken, see [9, 77, 148, 149]. We give a more general attempt on this topic in Section 7.4.

Algorithm 33 Incremental signature–based standard basis computation w.r.t. $<$ (IncSig)

Input: f_i a polynomial, $G_{i-1} = \{p_1, \ldots, p_{s-1}\}$ a standard basis for $\langle f_1, \ldots, f_{i-1} \rangle$
Output: B a standard basis for $\langle f_1, \ldots, f_i \rangle$ w.r.t. $<$

1: $B \leftarrow \emptyset, G \leftarrow \emptyset, P \leftarrow \emptyset$
2: $p_s \leftarrow f_i$
3: $t \leftarrow s$
4: **for** $(k = 1, \ldots, s)$ **do**
5:     $g_k \leftarrow (e_k, p_k)$
6: $G \leftarrow \{g_1, \ldots, g_s\}$
7: **for** $(k = 1, \ldots, s-1)$ **do**
8:     $u \leftarrow \operatorname{lc}(g_k) \frac{\tau(g_s, g_k)}{\operatorname{lm}(g_s)}$
9:     $v \leftarrow \operatorname{lc}(g_s) \frac{\tau(g_s, g_k)}{\operatorname{lm}(g_k)}$
10:     $P \leftarrow P \cup \{(ug_s, vg_k)\}$
11: **while** $(P \neq \emptyset)$ **do**
12:     Choose (uf, vg) from P with $\max_{<} \{u \operatorname{sig}(f), v \operatorname{sig}(g)\}$ minimal w.r.t. $<$.
13:     $P \leftarrow P \backslash \{(uf, vg)\}$
14:     $l \leftarrow u \operatorname{label}(f) - v \operatorname{label}(g)$
15:     $r \leftarrow \big(l, u \operatorname{poly}(f) - v \operatorname{poly}(g)\big)$
16:     $r \leftarrow$ SigRed(r, G)
17:     **if** $\big($ poly$(r) \neq 0$ and r not sig–redundant w.r.t. $G\big)$ **then**
18:         **for** $(k = 1, \ldots, t)$ **do**
19:             $u \leftarrow \operatorname{lc}(g_k) \frac{\tau(r, g_k)}{\operatorname{lm}(r)}$
20:             $v \leftarrow \operatorname{lc}(r) \frac{\tau(r, g_k)}{\operatorname{lm}(g_k)}$
21:             **if** $\big(\operatorname{lm}(u) \operatorname{siglm}(r) \neq \operatorname{lm}(v) \operatorname{siglm}(g_k)\big)$ **then**
22:                 $P \leftarrow P \cup \{(ur, vg_k)\}$
23:         $t \leftarrow t + 1$
24:         $g_t \leftarrow r$
25:         $G \leftarrow G \cup \{g_t\}$
26: $B \leftarrow \operatorname{poly}(G)$
27: **return** B

Let us have a closer look at Algorithm 33: IncSig computes the standard basis B for $\langle f_1, \ldots, f_i \rangle$. The starting point is the idea to use the previously computed standard basis $G_{i-1} = \{p_1, \ldots, p_{s-1}\} \subset \mathcal{P}$ for $\langle f_1, \ldots, f_{i-1} \rangle$ to compute a standard basis B for the ideal $\langle p_1, \ldots, p_{s-1}, f_i \rangle$ which is equal to $\langle f_1, \ldots, f_i \rangle$.

(1) As a first point we start with the construction of our initial set of labeled polynomials, taking the element f_i, previously reduced w.r.t. G_{i-1}, and store it in p_s (Line 2). Note that we assume $p_1, \ldots, p_s$ to be our initial generators of the ideal we want to compute a standard basis for, thus our labeled polynomials are elements of $\mathcal{P}^s \times \mathcal{P}$, where $\pi : \mathcal{P}^s \to \langle p_1, \ldots, p_s \rangle$ maps e_k to p_k for $1 \leq k \leq s$.

(2) In the `for`–loop we build the first batch of critical pairs. Here we note that we do

Algorithm 34 Semi–complete sig–safe reduction algorithm (SigRed)

Input: f a labeled polynomial, $G = \{g_1, \ldots, g_t\}$ a finite set of labeled polynomials
Output: h a labeled polynomial sig–safe reduced w.r.t. G

```
s ← siglm(f)
l ← label(f)
p ← poly(f)
while (p ≠ 0 and D_p ← {g ∈ G | lm(poly(g)) | lm(p)} ≠ ∅) do
    Choose any g ∈ D_p.
    u ← lt(p) / lt(poly(g))
    if (lm(u) siglm(g) ≺ s) then
        p ← p − u poly(g)
        l ← l − u label(g)
h ← (l, p)
return h
```

not need to consider any critical pair generated by g_k, g_l such that $k, l < s$ as G_{i-1} is already a standard basis for itself.

(3) The next point is very important as correctness as well as termination of SigStd are based on it: The order in which critical pairs are handled: Having a pair (uf, vg) the corresponding s–vector gets the signature

$$\max_{\prec}\left\{u \operatorname{sig}(f), v \operatorname{sig}(g)\right\}.$$

The choice we make is to get exactly those pair, whose maximum of its two signatures is minimal for all pairs in P (Line 12). In more detail, we choose the pair (uf, vg) from P such that

$$\max_{\prec}\left\{u \operatorname{sig}(f), v \operatorname{sig}(g)\right\} = \min_{\prec}\left\{\max_{\prec}\left\{u' \operatorname{sig}(f'), v' \operatorname{sig}(g')\right\} \mid (u'f', v'g') \in P\right\}.$$

If there are several critical pairs of the same signature we take the one, which was added to P first.

(4) After the computation of the s–vector r we handle its reduction w.r.t. G in Algorithm 34. The crucial point is that SigRed fulfills only semi–complete reductions with r. This has some impact on the algorithm:

▷ The signature of r remains unchanged during the reduction steps.

▷ $\operatorname{lm}(r)$ can still be $\lambda \operatorname{lm}(g_j)$ for some $\lambda \in \operatorname{Mon}(x_1, \ldots, x_n)$, $g_j \in G$. This happens if $\lambda \operatorname{siglm}(g_j) \geq \operatorname{siglm}(r)$. This can lead to sig–redundant labeled polynomials, which explains why one needs to test this in Line 17.

(5) If r is not sig–redundant and $\operatorname{poly}(r) \neq 0$, then we go on with the following steps:

▷ Generate new critical pairs with r and elements of G as long as the pair is not sig–equivalent. We show in Lemma 4.2.4 that those elements are not needed to be investigated by the algorithm.

▷ Add r to G.

(6) Then we go on with the next element in P, again choosing the one of minimal signature. When P is empty we are done and take the polynomial part of G, which is a standard basis for $\langle f_1, \ldots, f_i \rangle$.

Obviously, we need to clarify some points and decisions described above:

▷ Why is it enough to do semi–complete reductions and not complete ones?

▷ Why do we not need to care about sig–equivalent critical pairs? Are they not important for the correctness of the algorithm?

We prove correctness and termination of SigStd in several steps, giving answers to the above questions.

Lemma 4.2.3. *Let (uf, vg) be a critical pair generated in IncSig, and let $h = \mathcal{S}(f, g)$. Then it holds that* $\operatorname{sig}(h) \in \operatorname{signatures}\big(\operatorname{poly}(h)\big)$.

Proof. This is clear by the fact that the signatures of the initial elements $g_1, \ldots, g_s$ of G are correct: For all g_i with $i < s$ this is clear by definition. For g_s this is clear as we have reduced f_i beforehand by G_{i-1} in Line 3 of Algorithm 32. Thus e_s is the minimal signature for p_s. □

Lemma 4.2.4. *Suppose $f = (l, p)$ is a sig–redundant labeled polynomial in IncSig as return value of SigRed (Line 16) with* $\operatorname{poly}(f) \neq 0$. *Then there exists some $g \in G$ such that any s–vector being generated by f has a standard representation w.r.t. G, when all s–vectors generated by g have been considered.*

Proof. By Definition 4.2.1 there exists $g = (t, q) \in G$ such that $\operatorname{lm}(t) \mid \operatorname{lm}(l)$ and $\operatorname{lm}(q) \mid \operatorname{lm}(p)$. Let $v \in \operatorname{Mon}(x_1, \ldots, x_n)$ such that $v \operatorname{lm}(q) = \operatorname{lm}(p)$. Looking at the signatures two cases can happen:

(1) If $v \operatorname{lm}(t) \prec \operatorname{lm}(l)$, then reducing f by g is sig–safe and semi–complete. This is a contradiction to our assumption that f is the return value of SigRed, since then this reduction must have already taken place.

(2) If $v \operatorname{lm}(t) \succeq \operatorname{lm}(l)$, then there exists $w \in \operatorname{Mon}(x_1, \ldots, x_n)$ such that $w < v$, $w \operatorname{lm}(t) = \operatorname{lm}(l)$, and $w \operatorname{lm}(q) < \operatorname{lm}(p)$. Let $c \in \mathcal{K}$ such that $\operatorname{lt}(l) = cwt$. As we compute new s–vectors in IncSig by increasing signature, $h := (u, r)$ with $r := p - cwq$ has a standard representation w.r.t. G, as $\operatorname{lm}(u) \prec \operatorname{lm}(t)$. Any s–vector $\mathcal{S}(f, g')$ for some $g' = (t', q') \in G$ can be rewritten by h and a corresponding multiple of $\mathcal{S}(g, g')$, i.e. there exist terms $\lambda, \sigma \in \mathcal{P}$ such that

$$\lambda p - \sigma q' = \lambda(r + cwq) - \sigma q'.$$

Whenever $\mathcal{S}(g, g')$ has been considered by IncSig, $\mathcal{S}(f, g)$ has a standard representation w.r.t. G.

□

Lemma 4.2.5. *In* SigRed *there cannot be a complete sig–safe reduction without a semi–complete sig–safe reduction.*

Proof. Assume two labeled polynomials f, g, and let $c \in \mathcal{K}$ and $\lambda \in \mathrm{Mon}(x_1, \ldots, x_n)$ such that

$$\begin{aligned} \mathrm{lm}(f) &= \lambda\,\mathrm{lm}(g), \\ \mathrm{lc}(f) &= c\,\mathrm{lc}(g), \\ \mathrm{siglm}(f) &= \lambda\,\mathrm{siglm}(g), \text{ but} \\ \mathrm{sig}(f) &\neq c\lambda\,\mathrm{sig}(g). \end{aligned}$$

Clearly, there exists $c \neq d \in \mathcal{K}$ such that $\mathrm{lc}\big(\mathrm{sig}(f)\big) = d\,\mathrm{lc}\big(\mathrm{sig}(g)\big)$. As $\mathrm{siglm}(f - d\lambda g) < \mathrm{siglm}(f)$, and since IncSig proceeds by increasing signatures, $f - d\lambda g$ has a standard representation w.r.t. G. Thus there exist some $h \in G$ and some monomial $\sigma \in \mathcal{P}$ fulfilling $\sigma\,\mathrm{siglm}(h) < \mathrm{siglm}(f)$ and $\sigma\,\mathrm{lm}(h) = \mathrm{lm}(f)$. But then there exists a semi–complete, sig–safe reduction of f by h. □

Remark 4.2.6. The question arises why we have considered complete sig–safe reductions at all in Section 4.1? The point is that G2V is defined by using complete sig–safe reductions in [76]. It is first shown in [59] that it is enough to consider semi–complete reductions. So it is important to mention this fact when talking about signature–based standard basis algorithms.

On the other hand, current research of signature–based algorithms tries to generalize the signatures to include more than the leading term of the label for detecting useless critical pairs (see Section 4.3 resp. 7.3). There it is essential to include the coefficients in the computations as otherwise the data is corrupted.

Now we are ready to prove correctness and termination of SigStd.

Theorem 4.2.7. *Let $F \subset \mathcal{P}$ be the input of* SigStd*. Then* SigStd *is an algorithm computing a standard basis G for $\langle F \rangle$ w.r.t. the underlying monomial order $<$ on $\mathcal{P}$.*

Proof. We need to prove correctness and termination of SigStd.

(1) The proof of correctness of SigStd is based on Corollary 4.1.18. All s–vectors are considered besides

- ▷ those generated by sig–redundant elements (Line 17 of Algorithm 33), and
- ▷ those, whose corresponding critical pair is sig–equivalent. (Line 21 of Algorithm 33).

By Lemma 4.2.4 the s–vectors generated by sig–redundant elements can be assumed to have standard representations.

Moreover, we need to investigate the fact that only sig–safe reductions are taken into account. On the one hand, a sig–unsafe reduction of an element r by some element f in G, which is possibly needed for the correctness of the SigStd, is not computed in place. On the other hand, if this sig–unsafe reduction is necessary for poly(G) being a standard basis in the end, it will be considered in the following way: In Line 22 we generate new critical pairs with r. Here clearly the critical pair of r and f is considered, if this pair is needed at all. Thus the sig–unsafe reduction of r by f rejected beforehand leads to a new s–vector of higher signature (those of the multiple of the signature of f), which is reduced in the following to ensure the correctness of poly(G) (see Remark 4.3.3 for more details).

With this SigStd fulfills the hypothesis of Corollary 4.1.18.

(2) The monoid $\mathrm{Mon}(x_1,\ldots,x_{2n})$ can be considered, similar to $\mathrm{Mon}(x_1,\ldots,x_n)$, as an Noetherian $\mathrm{Mon}(x_1,\ldots,x_{2n})$–monomodule. Moreover, assume the initial setting of IncSig, and let $f = (a\lambda e_j, p) \in \mathcal{P}^s \times \langle p_1,\ldots,p_s\rangle$ be a labeled polynomial. From this we can extract the following, monomial data: λ and $\mathrm{lm}(p)$. Now, consider the map

$$\begin{array}{rccc} \psi: & \mathrm{Mon}(x_1,\ldots,x_n)\times\mathrm{Mon}(x_1,\ldots,x_n) & \longrightarrow & \mathrm{Mon}(x_1,\ldots,x_{2n}) \\ & \left(\lambda,\mathrm{lm}(p)\right) = \left(\prod_{i=1}^n x_i^{\alpha_i}, \prod_{i=1}^n x_i^{\beta_i}\right) & \longmapsto & \prod_{i=1}^n x_i^{\alpha_i} x_{n+i}^{\beta_i}. \end{array}$$

Let N be the $\mathrm{Mon}(x_1,\ldots,x_{2n})$–submodule generated by the labeled elements in G during the computations of IncSig.

Assume that IncSig adds an element $(b\sigma e_j, q)$ to G and N does not expand. This implies that there exists some $(c\tau e_j, r) \in G$ such that $\tau \mid \sigma$ and $\mathrm{lm}(r) \mid \mathrm{lm}(q)$. As it follows then that $(b\sigma e_j, q)$ is sig–redundant, IncSig does not add $(b\sigma e_j, q)$ to G. Thus we have a contradiction. So it follows that whenever IncSig adds a new element to G N expands. By Lemma 1.1.15, N, as a submodule of a Noetherian monomodule, can expand only finitely many times, so IncSig can only compute finitely many critical pairs. As the input of SigStd is finite, the number of iteration steps, i.e. calls of IncSig, is finite, too. All in all, SigStd terminates.

□

Lemma 4.2.8. *Let f be a recently reduced labeled polynomial such that* $\mathrm{poly}(f) \neq 0$ *in Line 17 of Algorithm 33. If there exist $\lambda \in \mathrm{Mon}(x_1,\ldots,x_n)$, $g \in G$ such that*

$$\lambda\,\mathrm{siglm}(g) = \mathrm{siglm}(f) \quad \textit{and} \quad \lambda\,\mathrm{lm}(g) = \mathrm{lm}(f),$$

then f is sig–redundant w.r.t. G.

Proof. Assume there exists $g \in G$ such that $\mathrm{siglm}(g) \mid \mathrm{siglm}(f)$ and $\mathrm{lm}(g) \mid \mathrm{lm}(f)$, and w.l.o.g. $\mathrm{lc}(f) = \mathrm{lc}(g) = 1$. Let $\lambda, \sigma \in \mathrm{Mon}(x_1,\ldots,x_n)$ be two monomials, and let $c \in \mathcal{K}$ such that

$$c\lambda\,\mathrm{sig}(g) = \mathrm{sig}(f) \quad \text{and} \quad \sigma\,\mathrm{lm}(g) = \mathrm{lm}(f).$$

We need to consider the following situations:

(1) If $\sigma < \lambda$, then the sig–safe reduction of f with g would not have taken place in SigRed. This is a contradiction.

(2) If $\sigma > \lambda$, then $\mathrm{lm}(f) = \sigma\,\mathrm{lm}(g) > \lambda\,\mathrm{lm}(g)$. By assumption, $\mathrm{sig}(f - c\lambda g) < \mathrm{sig}(f)$, but $\mathrm{lm}(f - c\lambda g) = \mathrm{lm}(f)$. Due to the smaller signature, $f - c\lambda g$ already has a standard representation w.r.t. G. By definition, there exist $h \in G$, $\gamma \in \mathrm{Mon}(x_1, \ldots, x_n)$ such that $\gamma\,\mathrm{lm}(h) = \mathrm{lm}(f) = \mathrm{lm}(f - c\lambda g)$ and $\mathrm{sig}(h) \leq \mathrm{sig}(f - c\lambda g) < \mathrm{sig}(f)$. But then SigRed should have computed the reduction of f by h, as it is semi–complete. This is, again, a contradiction.

□

We finish our introduction to signature–based algorithms with some essential remarks on the sig–safeness in the next section. Due to this, we postpone an example computation of SigStd to Section 4.3.

4.3 Some remarks on sig–safeness

Having presented a basic framework for signature–based standard basis computations in Section 4.2 based on the introduction of labeled polynomials in Section 4.1, we should draw the reader's attention to some apparent peculiarities:

At this point it is not really clear, why we force reductions in SigRed to be sig–safe. Reducing a labeled polynomial f by another onei, g, the information of a sig–unsafe reduction still seems to be valid, since we compute

$$\mathrm{label}(f) = \mathrm{label}(f) - u\,\mathrm{label}(g) \text{ for } u = \frac{\mathrm{lt}(f)}{\mathrm{lt}(g)}$$

in Line 9 of SigRed. Thus assuming $\mathrm{label}(f)$ and $\mathrm{label}(g)$ being complete labels for $\mathrm{poly}(f)$ resp. $\mathrm{poly}(g)$ in any possible configuration $\mathrm{sig}(f)$ and $u\,\mathrm{sig}(g)$ are related to each other, no data corruption could happen. Reconsidering Möller, Mora, and Traverso's attempt using syzygies to compute Gröbner bases presented in Section 3.3 their key problem reappears: Storing the whole label for a labeled polynomial is too much data. Keeping track and adjusting the label in each sig–safe reduction step in SigRed generates an undeniable overhead in the computations of SigStd.

The most efficient signature–based standard basis algorithms right now thus limit the data stored in the labeled polynomial: Instead of storing the whole label they only keep the signature of the labeled polynomial. Note that we only enforce $\mathrm{sig}(l) \in \mathrm{signatures}(p)$ for any labeled polynomial $r = (l, p)$ in Definition 4.1.4. Because of this we can align the data stored in l from a whole label of p to only a signature of p. As this last variant of labeled polynomials is essential in the following discussion, let us define some notation for it.

Definition 4.3.1. Let $r = (l, p)$ be a labeled polynomial. r is called *slim* iff $l \in \mathrm{signatures}(p)$.

Moreover, when constructing s–vectors of two slim labeled polynomials $f = (s, p)$ and $g = (t, q)$ we get

$$\mathcal{S}(f, g) = (w, up - vq),$$

where

(1) u, v are the corresponding multipliers for the s–vector $\mathcal{S}(p, q)$, and

(2) $w = \max_{<}\{us, vt\}$.

Clearly, the above definition coincides with Definition 4.1.14 w.r.t. the restriction on the labels of slim labeled polynomials.

Using this slim version of a labeled polynomial, Definition 4.1.13 becomes clearer: Having only the signature to be stored in our labeled polynomial, a sig–safe reduction can be done much faster than one that is not sig–safe. This is due to the fact that no computation on the label of the labeled polynomial must be done at all. In fact, we can completely delete Line 9 in SigRed. This has several advantages:

▷ One stores less data in memory,

▷ does less computations in SigRed, but

▷ still has information about the signature due to sig–safe reductions.

Clearly, on the one hand, having more data stored in the label of the labeled polynomial enables us to be less restrictive on the reduction process. On the other hand, the computational costs we inherit by doing this could decrease performance. Thus a more in–depth discussion on this topic can be found in Chapter 7.

In the meantime we concentrate on available implementations of signature–based algorithms and tempt to understand the main ideas behind them. For this, the following convention is quite useful.

Convention. Whenever we are looking at the theory behind signature–based algorithms we assume labeled polynomials with the complete corresponding label, i.e. $\pi\big(\operatorname{label}(f)\big) = \operatorname{poly}(f)$. In terms of implementation the reader can always assume slim labeled polynomials besides other noted.

Remark 4.3.2. Let us give some evident reason, why it is useful to consider the non–slim variant of labeled polynomials in theoretical considerations: Assume two labeled polynomials f and g such that $\operatorname{sig}(f) = t \operatorname{sig}(g)$ for some term $t \in \mathcal{P}$. Whereas from the implementational point of view it is enough for us to know the relation $\operatorname{sig}(f - tg) < \operatorname{sig}(f), t \operatorname{sig}(g)$, it is useful in theory to know the exact value of $\operatorname{sig}(f - tg)$. Note that it is only *useful, but not required* to prove statements in the following. It just shortens notation and makes results clearer.

Moreover, it simplifies the transition to the more general case presented in Section 7.3.

In this sense the stated pseudo code is given for slim labeled polynomials, which can differ slightly from the one for arbitrary ones, for example see the following "slim version" of SigRed given in Algorithm 35. Note that due to our above convention we also call this algorithm SigRed.

One clearly sees that the computation of the label reduction is absent. The reason is that in Line 8 the returned labeled polynomial still has the same label resp. signature as in the beginning of the computation.

Algorithm 35 Slim semi–complete sig–safe reduction algorithm (SigRed)

Input: f a labeled polynomial, $G = \{g_1, \dots, g_t\}$ a finite set of labeled polynomials
Output: h a labeled polynomial sig–safe reduced w.r.t. G

$s \leftarrow \operatorname{siglm}(f)$
$p \leftarrow \operatorname{poly}(f)$
while $\left(p \neq 0 \text{ and } D_p \leftarrow \left\{g \in G \mid \operatorname{lm}(\operatorname{poly}(g)) \mid \operatorname{lm}(p)\right\} \neq \varnothing\right)$ **do**
    Choose any $g \in D_p$.
    $u \leftarrow \frac{\operatorname{lt}(p)}{\operatorname{lt}(\operatorname{poly}(g))}$
    if $\left(\operatorname{lm}(u) \operatorname{siglm}(g) \prec s\right)$ **then**
        $p \leftarrow p - u \operatorname{poly}(g)$
$h \leftarrow \left(\operatorname{sig}(f), p\right)$
return h

In the same fashion Line 14 of Algorithm 33 changes from

$$l = u \operatorname{label}(f) - v \operatorname{label}(g)$$

to

$$l = \max_{\prec} \left\{u \operatorname{label}(f), v \operatorname{label}(g)\right\}.$$

As we restate IncSig in the following chapters several times due to the addition of criteria to detect useless critical pairs, we abandon a restatement of a slim version of IncSig at this point and incorporate the above mentioned change in the optimized versions presented later on.

Besides these evident reasons to choose a sig–safe reduction from an implementational point of view using slim labeled polynomials, we still need to discuss some curiosity in its behaviour during the computations of SigStd as already mentioned in the proof of Theorem 4.2.7.

Remark 4.3.3. The fact to allow only sig–safe reductions in SigStd clearly generates some computational overhead.

(1) First of all each new possible reducer in SigRed must be checked not only for divisibility of leading monomials, but also for a smaller signature.

(2) The second, even stranger point, is the way how sig–unsafe reductions are handled: Assume that one could reduce f by $c\lambda g$ in SigRed, since $\operatorname{lm}(f) = \lambda \operatorname{lm}(g)$, $c = \frac{\operatorname{lc}(f)}{\operatorname{lc}(g)}$. The reduction $f - c\lambda g$ does not take place if $\lambda \operatorname{lm}\left(\operatorname{sig}(g)\right) \succ \operatorname{lm}\left(\operatorname{sig}(f)\right)$. Assuming furthermore that there is no other reducer of f left and f being not sig–redundant, this means that f generates new critical pairs with elements already in G in IncSig (Line 22) and later on added to G (Line 25). The main fact ensuring correctness of $\operatorname{poly}(G)$ in the end is the generation of the critical pair $\left(c\lambda g, f\right)$. This critical pair is just the sig–unsafe reduction, we have rejected beforehand. So two things have happened:

a) The element f has been added to G whereas its leading monomial is not needed at all to ensure Buchberger's criterion.

b) The *real* reduction has been postponed, but still takes place.

The question arises why to do such a complicated and overhead–producing reduction process at all? The answer to this question is quite easy: the signatures.

▷ We need to ensure that the signature of an element does not increase during the reduction process, since our main idea is to equip a polynomial with its minimal possible signature.

▷ The element f is clearly useless for $\operatorname{poly}(G)$ *in the end*, but it is possibly crucial for its computation at all. Thinking about upcoming reductions in IncSig one can now reduce multiples of $\operatorname{lm}(f)$ either by σf or by $\sigma\lambda g$. As the signature of λg is greater than those of f it can happen that whereas a reduction with σf is allowed (sig–safe), a corresponding reduction by $\sigma\lambda g$ is rejected (sig–unsafe).

Thus neither f nor the reduction $f - c\lambda g$ can be left out to ensure correctness of the standard basis computations.

Let us close our discussion on this first, generic framework for signature–based algorithms, illustrating its behaviour by a small example. For this we choose the slim representation of labeled polynomials

Example 4.3.4. Let us give a small example illustrating computations done by SigStd. Assume $\mathcal{P}$ equipped with $<_{\text{dp}}$. As a well–order on the signatures we use $<_{\text{i}}$. Consider

$$F = \{y^2 - xz, x^2 - yz, xyz - y^2z\}$$

as generating set for $I = \langle F \rangle$. In this example we use the slim variant of labeled polynomials, which results in reductions only on the polynomial part, but not on the labels themselves.

We start with $G_1 = \{y^2 - xz\}$ which is clearly a Gröbner basis for $\langle y^2 - xz\rangle$. In the next iteration we enter IncSig for the computation of G_2, a Gröbner basis of $\langle y^2 - xz, x^2 - yz\rangle$. We start with initializing the set of labeled polynomials

$$G := \Big\{ \underbrace{(e_1, y^2 - xz)}_{g_1}, \underbrace{(e_2, x^2 - yz)}_{g_2} \Big\}.$$

There is only one critical pair to be considered in P, namely $(y^2 g_2, x^2 g_1)$. Generating the corresponding s–polynomial $r = (y^2 e_2, x^3 z - y^3 z)$ we compute a sig–safe reduction in SigRed via

$$\begin{array}{lllllll} r & := & (y^2 e_2, x^3 z - y^3 z) & - & xz(e_2, x^2 - yz) & = & (y^2 e_2, -y^3 z + xyz^2), \\ r & := & (y^2 e_2, -y^3 z + xyz^2) & + & yz(e_1, y^2 - xz) & = & (y^2 e_2, 0). \end{array}$$

Note that both reductions are semi–complete sig–safe reductions since $xze_2 < y^2 e_2$ and $yze_1 < y^2 e_2$. Thus $G_2 = \operatorname{poly}(G)$ is a Gröbner basis of $\langle y^2 - xz, x^2 - yz\rangle$. So we go on with the last iteration step, adding $xyz - y^2 z$ to our initial data:

$$G := \Big\{ \underbrace{(e_1, y^2 - xz)}_{g_1}, \underbrace{(e_2, x^2 - yz)}_{g_2}, \underbrace{(e_3, xyz - y^2 z)}_{g_3} \Big\}.$$

Generating the first critical pairs we get the pair set, already ordered by ascending signatures:

$$P := \{(yg_3, xzg_1), (xg_3, yzg_2)\}.$$

We start with generating the s–polynomial r corresponding to (yg_3, xzg_1) and reduce it sig–safe in SigRed:

$$\begin{array}{rcllcl}
r & := & (ye_3, x^2z^2 - y^3z) & - & z^2(e_2, x^2 - yz) & = & (ye_3, -y^3z + yz^3), \\
r & := & (ye_3, -y^3z + yz^3) & + & yz(e_1, y^2 - xz) & = & (ye_3, -xyz^2 + yz^3), \\
r & := & (ye_3, -xyz^2 + yz^3) & + & z(e_3, xyz - y^2z) & = & (ye_3, -y^1z^2 + yz^3), \\
r & := & (ye_3, -y^2z^2 + yz^3) & + & z^2(e_1, y^2 - xz) & = & (ye_3, -xz^3 + yz^3).
\end{array}$$

At this point no further reductions are possible and r is returned to IncSig in Line 16. We see that $\operatorname{poly}(r) \neq 0$ and r is not sig–redundant w.r.t. G, thus we add new critical pairs to P generated by $g_4 := r$:

$$P := \left\{(xg_3, yzg_2), (yg_4, -z^2g_3), (xg_4, -z^3g_2), (y^2g_4, -xz^3g_1)\right\}.$$

Note that

$$\underbrace{xe_3}_{x\,\operatorname{sig}(g_3)} < \underbrace{y^2e_3}_{y\,\operatorname{sig}(g_4)} < \underbrace{xye_3}_{x\,\operatorname{sig}(g_4)} < \underbrace{y^3e_3}_{y^2\,\operatorname{sig}(g_4)},$$

so it follows that the set P is already ordered by increasing signatures. We add g_4 to G:

$$G = \{g_1, g_2, g_3, g_4\}.$$

Computing the semi–complete sig–safe reductions of all the corresponding s–polynomials r we get $\operatorname{poly}(r) = 0$ for each. Thus the computation stops, and we have found a Gröbner basis for I:

$$\operatorname{poly}(G) = \{y^2 - xz, x^2 - yz, xyz - y^2z, -xz^3 + yz^3\}.$$

As one can plainly see, SigStd computes lots of zero reductions due to the fact that besides rejecting sig–equivalent critical pairs and discarding sig–redundant labeled polynomials in IncSig no real criterion is used to detect useless critical pairs. So right now we have only shown *how to add signatures* to polynomials, but not *how to use them* efficiently. This is the topic of the following chapters.

5 Signature–based criteria to detect useless critical pairs

After we have given a first, rather generic framework for signature–based standard basis computations we need to achieve more efficient implementations. Similar to the ideas of Section 2.3 we have to find criteria to detect and to reject useless critical pairs of labeled polynomials in SigStd. This is the crucial point still missing.

In this chapter we present first attempts in this direction, including some well–known implementations like G2V. We lay the groundwork for more aggressive implementations, like F5, with the presented, rather generic criteria. The reader should interpret this chapter as a collection of rather efficient criteria, which can also quite easily be integrated in SigStd.

All known signature–based algorithms up to now are based on 2 main criteria: The first one can be understood as a check for the minimality of the signature for the corresponding polynomial. We denote it as (NM). The second one, denoted (RW), is more or less a test for rewritings, i.e. is there another polynomial with the very same signature we should prefer? In spite of our approach in Section 2.3, where the main question was how to implement the criteria (see Section 2.4), we need to answer two questions for signature–based algortihms:

▷ Where to place the criteria in SigStd?

▷ How aggressive should the two criteria be implemented?

This second question seems a bit strange, but we see that the main differences between known signature–based algorithms lays just in this area. Of course, the answer to this question is not given by the formula *the more aggressive, the more efficient*, but it is rather complex to interpret the different behaviours.

In Section 5.1 we state, similar to our approach in Section 4.2, generic versions of our criteria. Due to their genericity they are not really efficient, and in some sense they lack a concrete implementation, but they illustrate the general concept quite clear.

As a first step, we show how to avoid as much as possible computational overhead, which emerges from the constraint that reductions must be sig–safe. This attempt leads to a variant called SigStdRed, which uses reduced intermediate standard bases.

A first optimization of the criteria used by SigStdRed is given in Section 5.3. Algorithm AP, first stated in [7] as a variant of Faugère's F5 Algorithm, can be understood as a more efficient variant of SigStdRed detecting more useless critical pairs due to extending (NM) and (RW). The main idea of AP is to be more specific on the choices which can be made in an implementation of (RW). However, Algorithm MM presented in Section 5.4, is just a variant of AP differing only slightly in the above noted choices.

Last we present Algorithm G2V of [76]. Besides explaining the algorithm, we show that it is also just a variant of SigStdRed, whose peculiarities can be adopted quite easily.

We close this chapter with a section comparing all presented variants of SigStd resp. SigStdRed, giving not only timings but also other, very important data needed to decide about the performance of a standard basis algorithm.

All implementations we present in the upcoming sections are based on the same setting:

Convention. $<$ denotes a well–order on $\mathcal{P}$. As inherited module order on the signatures resp. labels of the labeled polynomials we use $<_i$.

For generalizations of these algorithms we refer the reader to Chapter 7.

5.1 Generic criteria based on signatures

In the signature–based world two main criteria to detect useless critical pairs are known and can be described rather easily assuming two different labeled polynomials f, g:

(1) If $\operatorname{siglm}(f) > \operatorname{siglm}\big(\operatorname{poly}(f)\big)$, then discard f.

(2) If $\operatorname{siglm}(f) = \operatorname{siglm}(g)$, then compute only f or g, but not both.

In this section we do not only prove why the above criteria are correct, we also give ideas of how to implement them. Those implementations are of a rather general fashion such that they can be easily put in SigStd. Clearly, the advantage of an easy and generic implementation has drawbacks in terms of efficiency. In the following sections we cope with this problem, giving more concrete and more aggressive implementations of the two criteria.

The first criterion we present is based on a search for the minimal signature. It somewhat answers the following question for $f = (l, p)$: *Is* $\operatorname{siglm}(l)$ *equal to* $\operatorname{lm}\big(\operatorname{sig}(p)\big)$? If this is not the case, then we know by the relation $l \geq \operatorname{sig}(p)$ that l is greater than $\operatorname{sig}(p)$. Thus l is not the minimal signature of p.

Lemma 5.1.1 (Non–minimal signature criterion)**.** *Let (uf, vg) be a critical pair generated in IncSig, and let $h = \mathcal{S}(f, g)$. If* $\operatorname{siglm}(h) \neq \operatorname{lm}\big(\operatorname{sig}\big(\operatorname{poly}(h)\big)\big)$*, then h has a standard representation w.r.t. G at the moment IncSig generates h in Line 15.*

Proof. Assume that $\operatorname{siglm}(h) \neq \operatorname{lm}\big(\operatorname{sig}\big(\operatorname{poly}(h)\big)\big)$. Let $t = \operatorname{sig}\big(\operatorname{poly}(h)\big)$. Clearly, there exists a representation $r = \sum_{i=1}^{s} h_i e_i \in \mathcal{P}^s$ such that $\pi(r) = \sum_{i=1}^{s} h_i g_i = \operatorname{poly}(h)$ and $\operatorname{lm}(r) = t$. As IncSig proceeds by ascending signatures all cancellations of leading terms in $\sum_{i=1}^{s} h_i g_i$ correspond to s–vectors of lower signature than $\operatorname{sig}(h)$. Thus we can rewrite all those by their corresponding standard representation at the moment h is generated in IncSig. From this we conclude with a standard representation of h. □

The second criterion is based on the fact that whenever two elements f, g have the same signature during the computations of SigStd than at least one of those elements is useless and needs not be considered in the algorithm at all. Its name is based on the fact that one can rewrite the information which a computation of g would generate using f and other elements already stored in the intermediately computed set of labeled polynomials G.

Lemma 5.1.2 (Rewritable signature criterion)**.** *Assume the critical pair (ug, vh) in IncSig, w.l.o.g. let* $\operatorname{siglm}(\mathcal{S}(g,h)) = \operatorname{lm}(u)\operatorname{siglm}(g)$. *$\mathcal{S}(g, h)$ has a standard representation, if one of the following statements hold for any $f \in R = \{g' \in G \mid \operatorname{siglm}(g') \mid \operatorname{lm}(u)\operatorname{siglm}(g)\}$:*

(1) *$f \neq g$.*

(2) *$\mathcal{S}(f, f')$ is computed, where $f = g$, $f' \neq h$, and* $\operatorname{siglm}\big(\mathcal{S}(f, f')\big) = \operatorname{lm}(u)\operatorname{siglm}(g)$.

Please note again that due to SigRed only performing sig–safe reductions the second condition of Lemma 5.1.2 can appear in SigStd.

Proof. Let $t = \#(G)$, let $\mathcal{S}(g, h) = ug - vh$, and let $f \in R$. Then we know that

$$\operatorname{index}(f) = \operatorname{index}(g) = s \quad \text{and} \quad \operatorname{slm}(f) \mid \operatorname{lm}(u)\operatorname{slm}(g).$$

There exists a monomial $m \in \operatorname{Mon}(x_1, \ldots, x_n)$ such that $m \operatorname{slm}(f) = \operatorname{lm}(u)\operatorname{slm}(g) = \lambda$. Furthermore, adjusting the coefficient $c = \frac{\operatorname{lc}(u)\operatorname{slc}(g)}{\operatorname{slc}(f)}$, we know that $ug - cwf$ has a signature smaller than λe_s. As we proceed in IncSig by increasing signatures we know that $ug - cwf$ has already a standard representation w.r.t. G, i.e. there exist $p_k \in \mathcal{P}$, $g_k \in G$ such that

$$\begin{array}{rrcl} & ug - cwf & = & \sum_{k=1}^{t} p_k g_k, \\ \Rightarrow & ug & = & cwf + \sum_{k=1}^{t} p_k g_k. \end{array}$$

All top–cancellations of this last representation of ug have a signature which is at most equal to λe_s. From this it follows that

$$\begin{aligned}\mathcal{S}(g,h) &= ug - vh \\ &= cwf + \sum_{k=1}^{t} p_k g_k - vh\end{aligned}$$

has a standard representation once IncSig either chooses a new critical pair in Line 12, which has a signature greater than λe_s, or terminates. □

Convention. Lemmata 5.1.1 and 5.1.2 are the basic versions of all signature–based criteria we present in this thesis. As we refer to them lots of time, let us agree on the notations (NM) for the non–minimal signature criterion, and (RW) for the rewritable signature criterion.

Whereas (RW), Lemma 5.1.2, is already given in a way an implementation in IncSig can be easily done, Lemma 5.1.1 lacks this practical formulation. Because of this we need to find a realizable approach for (NM):

Lemma 5.1.3. *In IncSig, let $G_{i-1} = \{p_1, \ldots, p_{s-1}\}$ be the previously computed standard basis of $\langle f_1, \ldots, f_{r-1}\rangle$. Let $S = \{\mathrm{lt}(p_1), \ldots, \mathrm{lt}(p_{s-1})\}$, and let (uf, vg) be a critical pair in P. If there exists an $1 \le j \le s-1$ such that $\mathrm{lt}(p_j) \mid u\,\mathrm{slt}(f)$, then $\mathcal{S}(f,g)$ has a standard representation.*

Proof. Let $\mathrm{lt}(p_j) \in S$ such that $\mathrm{lt}(p_j) \mid u\,\mathrm{slt}(f)$. Then there exists a term $v \in \mathcal{P}$ such that $v\,\mathrm{lt}(p_j) = u\,\mathrm{slt}(f)$. It follows that there exists a principal syzygy

$$\omega = p_j e_s - p_s e_j \in \mathcal{P}^s$$

such that $\mathrm{lt}(v\omega) = u\,\mathrm{slt}(f)$. Clearly, $\pi(\omega) = 0$, thus we can easily generate

$$l = u\,\mathrm{sig}(f) - v\omega$$

fulfilling $\pi(l) = u\,\mathrm{poly}(f)$ and $\mathrm{lm}(l) < u\,\mathrm{siglm}(f)$. By Lemma 5.1.1 $\mathcal{S}(f,g)$ has a standard presentation. □

Lemma 5.1.3 is the first practical attempt of (NM) so far. With an easy corollary we can improve (NM)'s implementation even more.

Corollary 5.1.4. *In IncSig, let $G_{i-1} = \{p_1, \ldots, p_{s-1}\}$ be the previously computed standard basis of $\langle f_1, \ldots, f_{s-1}\rangle$, and let $S = \{\mathrm{lt}(p_1), \ldots, \mathrm{lt}(p_{s-1})\}$. Whenever SigRed returns a labeled polynomial h such that* $\mathrm{poly}(h) = 0$ *we can add* $\mathrm{slt}(h)$ *to S.*

Proof. Let f be the input value of SigRed, let h be the corresponding return value such that $\mathrm{poly}(h) = 0$, and let $G = \{g_1, \ldots, g_t\}$. We know that for $j \in \{1, \ldots, t\}$ there exist $h_j \in \mathcal{P}$ such that

$$\mathrm{poly}(f) = \sum_{j=1}^{t} h_j\,\mathrm{poly}(g_j).$$

As SigRed performs only sig–safe reductions for all those j it holds that

$$\mathrm{sig}(f) \succ \mathrm{lt}(h_j)\,\mathrm{sig}(g_j).$$

Thus we can construct $\omega = \mathrm{label}(f) - \sum_{j=1}^{t} h_j\,\mathrm{label}(g_j) \in \mathcal{P}^s$ such that

$$\begin{aligned}\pi(\omega) &= \operatorname{poly}(f) - \textstyle\sum_{j=1}^{t} h_j \operatorname{poly}(g_j) = 0,\\ \operatorname{lt}(\omega) &= \operatorname{sig}(f).\end{aligned}$$

As already shown in the proof of Lemma 5.1.3 we know now that any critical pair (ug, vh) with $\operatorname{lt}(\omega) \mid u \operatorname{slt}(g)$ has a standard representation, as we can rewrite its signature with a lower leading term subtracting a corresponding multiple of ω. Thus we can add $\operatorname{lt}(\omega) = \operatorname{slt}(f) = \operatorname{slt}(h)$ to S. □

Remark 5.1.5. Also (RW) can be implemented straightforwardly, one needs to decide which labeled polynomial resp. which critical pair to keep whenever two of them have the same signature. We see that this is one of the main differences between the later on presented, optimized implementations of signature–based algorithms like F5 or G2V.

The generic (RW), as stated in Lemma 5.1.2, keeps the first element resp. critical pair entering G resp. P, whereas all others of the same signature are discarded.

With this in mind we can update our implementation of SigStd, more precisely IncSig. SigStd and SigRed remain unchanged for the time being, (NM) and (RW) affect only IncSig. Remember that we use, as explained in detail in Section 4.3, slim labeled polynomials in the following.

Looking at Algorithm 36[1] one notices two major differences to Algorithm 33:

(1) IncSigCrit keeps and updates a set S of leading terms of elements in $\mathcal{P}$.

(2) IncSigCrit uses two subalgorithms called NonMin? and Rewrite? to decide whether a critical pair should be kept or not.

The set S is just the set of terms in $\mathcal{P}$ we need to check (NM) via implementation of Lemma 5.1.3 and Corollary 5.1.4: In Line 7 we initially fill S with the leading terms of the polynomials $p_j \in G_{i-1}$: Every critical pair which signature's leading term is a multiple of some element of S can be discarded by Lemma 5.1.3. Incorporation of Corollary 5.1.4 is done in Line 23: Whenever SigRed returns a sig–safe reduced labeled polynomial r such that $\operatorname{poly}(r)$ is zero, we can add the term of the leading part of $\operatorname{sig}(r)$ to S.

Let us have a closer look at how Algorithms 37 and 38 implements (NM) and (RW): Both return boolean values, "true" if they have detected a critical pair to be useless, "false" otherwise.

(1) NonMin? is quite self–explanatory: In Line 1 we store the term of the maximum of $\{\operatorname{sig}(uf), \operatorname{sig}(vg)\}$ and check if it is a multiple of some element of S in Line 6.

(2) In contrast, Rewrite? needs to do a lot more computations to check for the uselessness of (uf, vg) compared to NonMin?. Besides computing the maximum of $\operatorname{sig}(uf)$ and $\operatorname{sig}(vg)$ one also needs to store the generators of the pair separately for further checks. In Line 7 condition (1) of Lemma 5.1.2 is checked. If no such element $g_j \in G$ is found Rewrite? goes on and searches in P for other critical pairs having the same signature (Lines 2–7). This implements condition (2) of Lemma 5.1.2.

[1] Note that we use "!" to negate boolean values in this thesis.

Algorithm 36 IncSig including implementations of (NM) and (RW) (IncSigCrit)

Input: f_i a polynomial, $G_{i-1} = \{p_1, \ldots, p_{s-1}\}$ a standard basis for $\langle f_1, \ldots, f_{i-1}\rangle$
Output: B a standard basis for $\langle f_1, \ldots, f_i\rangle$ w.r.t. $<$

$B \leftarrow \varnothing, G \leftarrow \varnothing, P \leftarrow \varnothing$
$S \leftarrow \varnothing$
$p_s \leftarrow f_i$
$t \leftarrow s$
for $(k = 1, \ldots, s-1)$ **do**
  $g_k \leftarrow (e_k, p_k)$
  $S \leftarrow S \cup \{\mathrm{lt}(p_k)\}$
$g_s \leftarrow (e_s, p_s)$
$G \leftarrow \{g_1, \ldots, g_s\}$
for $(k = 1, \ldots, s-1)$ **do**
  $u \leftarrow \mathrm{lc}(g_k)\frac{\tau(g_s, g_k)}{\mathrm{lm}(g_s)}$
  $v \leftarrow \mathrm{lc}(g_s)\frac{\tau(g_s, g_k)}{\mathrm{lm}(g_k)}$
  if $\big(!$NonMin?(ug_s, vg_k, S) and $!$Rewrite?$(ug_s, vg_k, G, P)\big)$ **then**
    $P \leftarrow P \cup \{(ug_s, vg_k)\}$
while $(P \neq \varnothing)$ **do**
  Choose (uf, vg) from P with $\max_{\prec}\{u\,\mathrm{sig}(f), v\,\mathrm{sig}(g)\}$ minimal w.r.t. $\prec$.
  if $\big(!$NonMin?(uf, vg, S) and $!$Rewrite?$(uf, vg, G, P)\big)$ **then**
    $P \leftarrow P \setminus \{(uf, vg)\}$
    $l \leftarrow \max_{\prec}\{u\,\mathrm{sig}(f), v\,\mathrm{sig}(g)\}$
    $r \leftarrow \big(l, u\,\mathrm{poly}(f) - v\,\mathrm{poly}(g)\big)$
    $r \leftarrow$ SigRed(r, G)
    if $(\mathrm{poly}(r) = 0)$ **then**
      $S \leftarrow S \cup \{\mathrm{slt}(r)\}$
    else if $\big(\mathrm{poly}(r) \neq 0$ and r not sig-redundant w.r.t. $G\big)$ **then**
      for $(k = 1, \ldots, t)$ **do**
        $u \leftarrow \mathrm{lc}(g_k)\frac{\tau(r, g_k)}{\mathrm{lm}(r)}$
        $v \leftarrow \mathrm{lc}(r)\frac{\tau(r, g_k)}{\mathrm{lm}(g_k)}$
        if $\big(\mathrm{lm}(u)\,\mathrm{siglm}(r) \neq \mathrm{lm}(v)\,\mathrm{siglm}(g_k)\big)$ **then**
          if $\big(!$NonMin?(ur, vg_k, S) and $!$Rewrite?$(ur, vg_k, G, P)\big)$ **then**
            $P \leftarrow P \cup \{(ur, vg_k)\}$
      $t \leftarrow t + 1$
      $g_t \leftarrow r$
      $G \leftarrow G \cup \{g_t\}$
$B \leftarrow \mathrm{poly}(G)$
return B

Remark 5.1.6.

(1) Note that we always consider the maximum of $\mathrm{sig}(uf)$ and $\mathrm{sig}(vg)$ of a critical pair (uf, vg) in (NM) resp. NonMin? or (RW) resp. Rewrite?. From this it follows

Algorithm 37 Generic implementation of (NM) (NonMin?)

Input: uf a labeled polynomial multiplied by a term, vg a labeled polynomial multiplied by a term, $S = \{t_1, \ldots, t_k\}$ a finite set of terms in $\mathcal{P}$

Output: TRUE if $\max_< \left\{ \operatorname{sig}(uf), \operatorname{sig}(vg) \right\}$ is not minimal, FALSE otherwise

$t \leftarrow \operatorname{slt}\left(\max_< \left\{ \operatorname{sig}(uf), \operatorname{sig}(vg) \right\}\right)$
for $\left(i = 1, \ldots, k\right)$ **do**
    if $\left(t_i \mid t\right)$ **then**
        return TRUE
return FALSE

that $\operatorname{index}(m) = s$ where $m = \max_< \left\{ \operatorname{sig}(uf), \operatorname{sig}(vg) \right\}$. Thus we always check signatures resp. labeled polynomials of the current index in IncSigCrit.

(2) It seems a bit extraordinary to check (NM) and (RW) 3 times in IncSigCrit (Lines 13, 17, and 29). Clearly, the check in Line 17 is the latest possible one, where "latest" means that after this line the s–vector computation and sig–safe reduction of the critical pair starts. So, from the point of view of fewer lines of code, this check is enough as this is the biggest data set (S, G, and P) we can consider for finding reasons to reject the corresponding critical pair. From the point of efficient implementations checking in Line 13 and 29 makes sense, too: The earlier we can throw away useless critical pairs, the better. Storing useless pairs in P costs time and memory and should be avoided as much as possible.

(3) Reconsidering (2) the reader should be aware that the sets S, G, and P could change dramatically between a check of (uf, vg) in Line 13 resp. Line 29 and a check in Line 17. Thus none of the three checks should be left out. Clearly, one can think of optimizations by considering in Line 17 only elements in S, G, and P which are added after the corresponding check in Line 13 resp. Line 17.

Clearly, correctness and termination of SigStd using IncSigCrit with NonMin? and Rewrite? follows straightforward from Lemma 5.1.1 and Lemma 5.1.2 combined with Theorem 4.2.7.

Corollary 5.1.7. *Let $F \subset \mathcal{P}$ be the input of SigStd. Then SigStd calling IncSigCrit is an algorithm computing a standard basis G for $\langle F\rangle$ w.r.t. the underlying monomial order $<$ on $\mathcal{P}$.*

Example 5.1.8 (Example 4.3.4 revisited)**.** Let us reconsider the example computation of SigStd using IncSig given in Section 4.3. This time we use SigStd with IncSigCrit.

Again we start with $G_1 = \{y^2 - xz\}$. Next we enter IncSigCrit with $p_2 = x^2 - yz$, generating

$$G := \Big\{ \underbrace{(e_1, y^2 - xz)}_{g_1}, \underbrace{(e_2, x^2 - yz)}_{g_2} \Big\}.$$

Algorithm 38 Generic implementation of (RW) (Rewrite?)

Input: uf a labeled polynomial multiplied by a term, vg a labeled polynomial multiplied by a term, $G = \{g_1, \ldots, g_t\}$ a finite set of labeled polynomials, $P = \{p_1, \ldots, p_k\}$ a finite set of critical pairs of labeled polynomials

Output: TRUE if $h \in \{uf, vg\}$ such that $\operatorname{sig}(h) = \max_{\prec}\left\{\operatorname{sig}(uf), \operatorname{sig}(vg)\right\}$ is detected by (RW), FALSE otherwise

if $\left(\operatorname{sig}(uf) \succ \operatorname{sig}(vg)\right)$ **then**
  $h \leftarrow f, w \leftarrow u$
  $\tilde{h} \leftarrow g, \tilde{w} \leftarrow v$
else
  $h \leftarrow g, w \leftarrow v$
  $\tilde{h} \leftarrow f, \tilde{w} \leftarrow u$
for $\left(j = 1, \ldots, t\right)$ **do**
  if $\left(g_j \neq h \text{ and } \operatorname{sig}(g_j) \mid \operatorname{sig}(wh)\right)$ **then**
    return TRUE
for $\left(j = 1, \ldots, k\right)$ **do**
  $(u'f', v'g') \leftarrow p_j$
  if $\left(\operatorname{sig}(u'f') \succ \operatorname{sig}(v'g')\right)$ **then**
    $h' \leftarrow f', w' \leftarrow u'$
    $\tilde{h}' \leftarrow g', \tilde{w}' \leftarrow v'$
  else
    $h' \leftarrow g', w' \leftarrow v'$
    $\tilde{h}' \leftarrow f', \tilde{w}' \leftarrow u'$
  if $\left(h' = h \text{ and } w' = w \text{ and } \tilde{h}' \neq \tilde{h}\right)$ **then**
    return TRUE
return FALSE

Moreover, $S = \{y^2\}$. The single critical pair $(y^2 g_2, x^2 g_1)$ is not computed, but discarded, since $y^2 \in S$ is equal to the multiplier y^2 of g_2, which represents

$$y^2 e_2 = \max_{\prec}\left\{\operatorname{sig}(y^2 g_2), \operatorname{sig}(x^2 g_1)\right\}.$$

Thus no s–vector is considered at all in this round of IncSigCrit, and we add $p_3 = xyz - y^2 z$ to our data set. Having $S = \{y^2, x^2\}$. As in Example 4.3.4 we consider the pairs (yg_3, xzg_1) and (xg_3, yzg_2). The first one generates $g_4 = (ye_3, -xz^3 + yz^3)$. At this point the pair set consists of 4 critical pairs:

$$P := \left\{(xg_3, yzg_2), (yg_4, -z^2 g_3), (xg_4, -z^3 g_2), (y^2 g_4, -xz^3)\right\}.$$

Instead of computing all those pairs and ending up with a zero reduction in each case as it is done in IncSig, IncSigCrit actively uses S to detect zero reductions in advance:

▷ (xg_3, yzg_2) is computed and ends with a zero reduction. It follows that x from $xe_3 = \max_{\prec}\left\{\operatorname{sig}(xg_3), \operatorname{sig}(yzg_2)\right\}$ is added to S.

▷ $y^2 e_3 = \max_{<}\{\operatorname{sig}(yg_4), \operatorname{sig}(-z^2 g_3)\}$ is detected by $y^2 \in S$, thus $(yg_4, -z^2 g_3)$ is discarded.

▷ $xye_3 = \max_{<}\{\operatorname{sig}(xg_4), \operatorname{sig}(-z^3 g_2)\}$ is detected by $x \in S$, thus $(xg_4, -z^3 g_2)$ is discarded.

▷ $y^3 e_3 = \max_{<}\{\operatorname{sig}(y^2 g_4), \operatorname{sig}(-xz^3 g_1)\}$ is detected by $y^2 \in S$, thus $(y^2 g_4, -xz^3 g_1)$ is discarded.

In our example, due to the fact of generating only 1 new element throughout the whole computation, (RW) does not reject any critical pair at all. We show its usefulness in bigger examples illustrating G2V and F5 in the following.

Nevertheless, we see that using (NM) in IncSigCrit we end up with only 1 zero reduction, which we actively use: Adding x to S enables us to discard the pair $(xg_4, -z^3 g_2)$. Also note that this pair is checked and not detected to be useless in Line 29 of IncSigCrit *before* the zero reduction of (xg_3, yzg_2) is known. Thus the recheck in Line 17 is really necessary to reject $(xg_4, -z^3 g_2)$.

Remark 5.1.9.

(1) Note that the efficiency of (NM) and (RW) depend on the order, in which the elements of the set of initial generators of the ideal are entered to IncSigCrit. The problem is that one cannot predefine the best possible way. This problem is part of further discussions on optimizations, especially of F5, given later on. For the moment let us assume to order the set of initial generators F always by increasing leading terms. Speaking in terms of the incremental behaviour of SigStd this clearly holds for all G_i used as input data of IncSigCrit, too.

(2) As one can easily realize from looking at the pseudo codes of Algorithm 37 and Algorithm 38 computational time and memory storage are much higher for Rewrite? than they are for NonMin?. Later on we see that adding leading terms of signatures of zero reducions is one benefit of G2V over F5, which can be easily adopted to F5 and optimizes its performance in some classes of example sets immensely.

5.2 Reducing computational overhead in SigStd

The main problem of SigStd is the combination of sig–safe reductions with incremental computations: The intermediate standard bases G_i IncSigCrit returns are neither reduced nor minimal in general. As these G_i are the starting point for the next iteration step, taking f_{i+1} into account, the overhead of

(1) multiples of leading terms as well as

(2) quite dense, not tail–reduced polynomials

affects upcoming computations and thus generates even more useless data.

Let us try to understand where this computational overhead is inherited and how to avoid it as much as possible in a sensible way. The ideas given in this section are based on [58], where Perry and the author have presented the idea of interreducing intermediate bases in F5. As it is a common tool in nearly all available signature–based algorithms these days, we decided to present the idea at this point of the thesis for a better understanding on how to optimize signature–based algorithms in general.

Let us start with the problem of having a non–minimal standard basis G_i at the end of the i–th call of IncSigCrit.

(1) Due to the fact that the signatures of the labeled polynomials must be kept valid during the reductions taking place in SigRed, some leading term reductions do not take place immediately, but are postponed. These reductions, which are needed to ensure correctness of SigStd are computed when generating new critical pairs. Thus at the end we could have the 3 polynomials $\operatorname{poly}(f)$, $\operatorname{poly}(g)$, and $\operatorname{poly}(h)$ in G_i in SigStd such that

- ▷ $\operatorname{lt}(g) \mid \operatorname{lt}(f)$, but the reduction $f - tg$ for some term $t \in \mathcal{P}$ has not taken place due to sig–unsafeness.
- ▷ h is the result of the later on constructed s–vector $tg - f$, which is sig–safe due to changing the order of tg and f.

In the end, we only need two out of these three elements for a standard basis; in a minimal standard basis we would discard f. The problem is that for the correctness of IncSigCrit the computation and addition of the labeled polynomial f is important: Without adding f to G in IncSigCrit the critical pair (tg, f) would not be generated at all, thus the element h, possibly needed for the correctness of the standard basis in the end, would never be computed. So we are not able to remove f during the actual iteration step.

Clearly, in the same vein the problem of non–reducedness of the standard basis G_i, i.e. the missing tail–reductions, can be understood.

(2) Since SigRed computes only reductions of the leading terms of the polynomial parts of the labeled polynomials, elements with non–reduced tails can be entered to G_i. The main argument for not doing complete reductions in SigRed is the requirement of sig–safeness: Comparing the signatures must also be done before each possible tail–reduction. This can lead to quite worse timings. From the point of view of the already computed standard basis G_i, returned by IncSigCrit, which consists only of polynomial data, we do not need to take care of sig–safeness and can tail–reduce the elements in G_i as usual without any preprocessed comparison. This is way faster than implementing tail–reductions in SigRed, although we have to use the non–tail–reduced elements during a whole iteration step.

From this discussion we can derive the following:

- ▷ The computational overhead *during* an iteration step is prerequisite for the correctness of IncSigCrit and thus of SigStd.

- ▷ The polynomial standard basis G_i returned by IncSigCrit *after* the ith iteration step is used as input for the $(i+1)$st iteration step. The emphasis lies on the fact that only the *polynomial* structure is used. Each such polynomial gets a new signature at the beginning of IncSigCrit when initializing G in Line 9.

Thus it follows that one can easily reduce the intermediate standard basis G_i after the ith and before the $(i+1)$st call of IncSigCrit. This is illustrated by the pseudo code given in Algorithm 39: The only difference to Algorithm 32 is given in Line 3. Instead of the standard basis for $\langle f_1, \dots, f_i \rangle$ computed in IncSigCrit for the $(i+1)$st iteration step in IncSigCrit, the corresponding reduced standard basis is computed. RedSB takes a standard basis and computes the corresponding, reduced standard basis. Thus in the next iteration step IncSigCrit starts with a reduced standard basis as input.

Algorithm 39 SigStd with reduced standard bases (SigStdRed)

Input: $F = \{f_1, \dots, f_r\}$ a finite subset of $\mathcal{P}$
Output: G a standard basis for $\langle F \rangle$ w.r.t. $<$

```
1: G_1 ← {f_1}
2: for (i = 2, ..., r) do
3:     G_{i-1} ← RedSB (G_{i-1})
4:     f_i ← Reduce(f_i, G_{i-1})
5:     if (f_i ≠ 0) then
6:         G_i ← IncSig(f_i, G_{i-1})
7:     else
8:         G_i ← G_{i-1}
9: G ← G_m
10: return G
```

Clearly, the idea of tail–reducing the polynomials in G_i before starting the $(i+1)$st iteration step gives advantages in the reduction process:

- ▷ The polynomials are possibly sparser, which leads to less operations for multiplying them with terms and comparing with other terms when subtracting polynomials.
- ▷ Some reductions which would have taken place in SigRed, possibly multiple times, are already carried out once.

The only drawback of reducing the intermediate standard bases could be that some useless critical pairs which are detected in SigStd are no longer detected in SigStdRed, but luckily this is not true at all.

Proposition 5.2.1. *Any useless critical pair detected by* NonMin? *or* Rewrite? *in* SigStd *is also detected in* SigStdRed.

Proof. Assume that G_i is the return value of IncSigCrit after the ith iteration step, and let $B_i = \text{RedSB}(G_i)$. By the above discussion we do not need to take the tail–reduction of the elements both in G_i and B_i into account. So it is left to consider elements, which are in G_i, but are removed from B_i (if no such element exists we are done). Let g be such an element.

It is removed from B_i since there exists an element $h \in G_i \cap B_i$ such that $\mathrm{lt}(h) \mid \mathrm{lt}(g)$. Let us assume the $(i+1)$st call of IncSigCrit and investigate the differences between using G_i and B_i as input data. For this we need to look at NonMin? and Rewrite?. Let $g_i, g_j, g_k \in G$ in IncSigCrit such that $\mathrm{poly}(g_i) = g$ and $\mathrm{poly}(g_j) = h$, $\mathrm{index}(g_k)$ be the current index[2]. Moreover, we assume that g_k is just returned by SigRed and new critical pairs with g_k and elements of G need to be generated.

(1) Using G_i, $\mathrm{lt}(g) \in S_{G_i}$. As $g \notin B_i$, $\mathrm{lt}(g) \notin S_{B_i}$. From this it clearly follows that $\#(S_{G_i}) > \#(S_{B_i})$. Whenever $\mathrm{lt}(g_i)$ would detect a useless critical pair in SigStd we know that there exists $\mathrm{lt}(g_j) \in S_{G_i} \cap S_{B_i}$ such that $\mathrm{lt}(g_j) \mid \mathrm{lt}(g_i)$. Moreover, by Remark 5.1.9 (1) it holds that $j < i$. It follows that any useless critical pair detected by NonMin? in SigStd is also detected by NonMin? in SigStdRed.

(2) Next we investigate the differences using Rewrite? in SigStd resp. SigStdRed. Since we generate less critical pairs in SigStdRed than in SigStd we need to check, if the signature of one of these not generated critical pairs could be used to detect more useless critical pairs in SigStd. Since g_i and g_j are both in G the critical pairs $(u_k g_k, u_i g_i)$ and $(v_k g_k, v_j g_j)$ are considered. By our above assumption it holds that

$$\tau(g_k, g_j) \mid \tau(g_k, g_i).$$

Moreover, by $\mathrm{index}(g_k) > \max\{i, j\}$ and by assuming $<_{\mathrm{i}}$ we have that

$$\begin{aligned} u_k \,\mathrm{sig}(g_k) &> u_i \,\mathrm{sig}(g_i) \\ v_k \,\mathrm{sig}(g_k) &> v_j \,\mathrm{sig}(g_j). \end{aligned}$$

As $j < i$ by Remark 5.1.9 (1) (g_k, g_j) is investigated before (g_k, g_i). Moreover, $v_k \mid u_k$. Thus, two situations can happen:

a) If (g_k, g_j) is detected to be useless in Rewrite?, then (g_k, g_i) is detected, too, as $v_k \mid u_k$.

b) If (g_k, g_j) is not detected to be useless in Rewrite?, then (g_k, g_i) is detected to be useless, since $v_k \mid u_k$, i.e.

$$\mathcal{S}(g_k, g_i) = \frac{u_k}{v_k} \mathcal{S}(g_k, g_j) + \sum_{l=1}^{k} w_l g_l,$$

where $u_k \,\mathrm{sig}(g_k) > \max_{\prec} \{ w_l \,\mathrm{sig}(g_l) \mid 1 \le l \le k \}$.

Hence (g_k, g_i) is never used in SigStd to detect a useless critical pair.

□

From Proposition 5.2.1 and our previous discussion on the advantages considering tail–reduced elements it is clear that computations nearly always (see Remark 5.2.2 below for an explanation of *nearly*) benefit from reducing the intermediate standard bases due to the following facts:

[2] Thus $\mathrm{poly}(g_k) \neq g$ and $\mathrm{poly}(g_k) \neq h$.

▷ less reduction steps,

▷ pre–detection of useless critical pairs due to minimalization, and

▷ faster detection with NonMin? due to less elements in S.

Remark 5.2.2.

(1) Note that the solution of reducing computational overhead by interreducing the intermediate standard bases G_i after each iteration step is nowadays standard in incremental signature–based algorithms. Nevertheless it is important to mention that Perry and the author where the first to present this idea by optimizing F5 in [58]. See Section 6.2 for more information on this.

(2) The functioning of this idea is based on the fact that we assume $<$ to be a well–order in the signature–based setting. Otherwise a terminating computation of a reduced standard basis B out of a non–reduced standard basis G via RedSB as it is assumed in Algorithm 39, Line 3, is not provided in general (see Section 1.7). Still, a minimal standard basis can be computed nevertheless, which drops the computational overhead, too.

(3) In SigStdRed, as presented in Algorithm 39, we do not reduce the last standard basis. Thus the result of SigStdRed need not be the reduced standard basis of the ideal generated by the input data. One can do another reduction of G_r at the end before returning the result, but this comptutation can be heavy. Most of the time a standard basis is enough for further computations, thus it saves time and memory to not reduce at the end.

(4) Also note that due to the fact that F5 implements (NM) and (RW) quite more aggressive than all variants presented in the current chapter (being just variants of SigStd) an optimization in the vein of the one given in this section cannot as easily be done as illustrated here.

(5) There are some situations where SigStd can be faster than SigStdRed. Those are quite unusual and not performance–critical at all, but we should mention them here for the sake of completeness of our discussion:

   a) One possibility would be that all intermediate standard bases computed by IncSigCrit are already reduced. Calling RedSB in SigStdRed produces some more computational overhead in this situation, but which can be neglected in comparison to the complete computation of the algorithm w.r.t. memory usage and time.

   b) The second possibility is that the whole computation is done so fast that reducing the standard bases inbetween the iteration steps slows done the algorithm a bit. Considering such ideals the performance of SigStd resp. SigStdRed is not critical at all.

In Section 5.6 we compare signature–based algorithms based on SigStd with those using SigStdRed as basis in various different examples to illustrate the benefits of reducing the standard bases between iteration steps.

In the remaining of this chapter we assume all variants of the generic signature–based standard basis algorithm, presented in Section 5.1, to use SigStdRed instead of SigStd.

5.3 An explicit choice in (RW)

In [7] Arri and Perry have presented an algorithm, which can be understood as a generalization of Faugère's F5 Algorithm by changing F5's implementation of (RW). The nice fact is that from our recent point of view in this thesis their algorithm is (restricted to our predefined module order $<_i$) nothing else but a variant of implementing (NM) and (RW) in IncSigCrit.

Sadly the authors of [7] have not given their algorithm a concrete name. In this thesis we denote it AP , by the first letter of their respective surname, in the vein of other naming conventions in the signature–based world.

The historical reason for this is that our way more general attempt to signature–based algorithms we have presented in this thesis has been developed first in late 2010, preparing [59], whereas the ideas for [7] go back to 2009. We give the connection to F5 in detail in Section 6.3 resp. Section 7.4 (see Remark 5.3.1 below):

$$\text{SigStdRed}(\text{IncSigCrit}) \xrightarrow{\text{Section 5.3}} \text{AP} \xleftarrow{\text{Section 6.3}} \text{F5}$$

The generalizations of F5's criteria by Ars and Hashemi (see [9]), Sun and Wang (see [148]), and Zobnin (see [163]) can be understood as special cases of the algorithm presented in this section.

Remark 5.3.1. It is important to note that AP as presented in [7] is much more general than the restricted version we state in this section. This is due to the fact that AP can be used w.r.t. to any module well–order $<$ and is not restricted to $<_i$. This can lead to non–incemental signature–based algorithms. We consider those in Section 7.4. There we discuss this more general, module order independent version of AP in detail.

We review some of their definitions and state the main theorem of [7]. Afterwards we merge their ideas to our attempt and see that AP differs from SigStdRed just by a special implementation of (RW).

In [7], the notion of a *normal critical pair* is defined, which restricts a critical pair by ensuring some properties on its generators. We show that these properties are just special interpretations resp. implementations of sig–redundancy, (NM), and (RW).

Definition 5.3.2. We denote the variant of SigStdRed calling IncSigCrit, NonMinAP?, and RewriteAP? by AP.

Definition 5.3.3. Let $I = \langle f_1, \ldots, f_r \rangle$ be an ideal in $\mathcal{P}$. We say that a set G of labeled polynomials such that $\{f_1, \ldots, f_r\} \subset \mathrm{poly}(G)$ is an sig–*standard basis for* I, if for each not sig–safe reducible element $f \in G$ there exist $g \in G$ and a term $t \in \mathcal{P}$ such that $\mathrm{lt}(tg) = \mathrm{lt}(f)$ and $\mathrm{sig}(tg) = \mathrm{sig}(f)$.

The main usage of sig–standard bases can be found in Theorem 5.3.5: There a computational approach of sig–standard bases is given. For the complete statement of Theorem 5.3.5 we need some more notation.

Definition 5.3.4. In IncSigCrit we call a critical pair (uf, vg) *normal* if

(1) $\mathrm{lm}(u)\,\mathrm{siglm}(f) \neq \mathrm{lm}(v)\,\mathrm{siglm}(g)$,

(2) neither uf nor vg are sig–redundant, and

(3) $\mathrm{lm}(u)\,\mathrm{siglm}(f) = \mathrm{siglm}(uf)$ and $\mathrm{lm}(v)\,\mathrm{siglm}(g) = \mathrm{siglm}(vg)$.

Theorem 5.3.5. *Let G be a finite set of not sig–safe reducible labeled polynomials whose polynomial parts are in I. If*

(1) for each $i = 1, \ldots, r$ such that $e_i \notin L\big(\mathrm{Syz}(F)\big)$ there exists $g \in G$ such that $\mathrm{siglm}(g) = e_i$*, and*

(2) for any $f, g \in G$ such that (uf, vg) is a normal critical pair there exist a labeled polynomial $h \in G$ and a term $t \in \mathcal{P}$ such that th is not sig–safe reducible and $\mathrm{siglm}(th) = \mathrm{siglm}\big(\mathcal{S}(f,g)\big)$,

then G is an sig–*standard basis for I.*

Proof. See proof of Theorem 18 in [7]. □

Proposition 5.3.6. *From every* sig–*standard basis G for an ideal $I \subset \mathcal{P}$ one can derive a standard basis H of I.*

Proof. See proof of Proposition 14 in [7]. □

These are the main facts of AP, they can be quite easily translated to fit into SigStdRed.

The notion of a sig–standard basis is not important in our approach, it is useful the way the authors describe and embed their ideas in [7], but not needed in our more general attempt to signature–based standard basis algorithms.

Let us have a closer look at the definition of normal critical pairs, which are the ones of interest in Theorem 5.3.5:

▷ Property (1) is included in Theorem 4.1.18 and thus also in SigStd.

▷ (2) discards critical pairs generated by sig–redundant labeled polyomials. Those are also discarded in IncSigCrit.

▷ Property (3) is just a reformulation of (NM), thus it can be implemented via NonMin?.

From this we can follow:

Algorithm 40 AP's implementation of (NM) (NonMinAP?)

Input: uf a labeled polynomial multiplied by a term, vg a labeled polynomial multiplied by a term, $S = \{t_1, \ldots, t_k\}$ a finite set of terms in $\mathcal{P}$

Output: TRUE if $\operatorname{sig}(uf)$ or $\operatorname{sig}(vg)$ is not minimal, FALSE otherwise

1: $s \leftarrow \operatorname{slt}(uf)$
2: $t \leftarrow \operatorname{slt}(vg)$
3: **for** $(i = 1, \ldots, k)$ **do**
4:     **if** $(t_i \mid s \text{ and } i < \operatorname{index}(f))$ **then**
5:         **return** TRUE
6:     **if** $(t_i \mid t \text{ and } i < \operatorname{index}(g))$ **then**
7:         **return** TRUE
8: **return** FALSE

Lemma 5.3.7. AP *implements (NM) similar to* IncSigCrit, *but extends the criteria check to both generators of the critical pair* (uf, vg).

This leads to a new implementation of (NM) we present in Algorithm 40.

The pseudo code should be clear to the most parts, NonMinAP? checks not only the multiplied generator which corresponds to $\max_{\prec}\{\operatorname{sig}(uf), \operatorname{sig}(vg)\}$, but both generators (see Lines 1 and 2 of Algorithm 40). Due to our module order this leads to the extra overhead of checking the index of the generator (Lines 4 and 6). Here we do not know if $\operatorname{index}(f)$ resp. $\operatorname{index}(g)$ is the current index of IncSigCrit. So it is possible that (one of them) has a lower index. Let us assume that $\operatorname{index}(g)$ is smaller than the current index s. To discard a critical pair using (NM) we need to ensure the existence of a principal syzygy whose leading term divides $\operatorname{sig}(vg)$. If we find an element $t_i \in S$ such that $t_i \mid \operatorname{slt}(vg)$ but $i \geq \operatorname{index}(g)$, then we cannot build a principal syzygy:

- ▷ If $i = \operatorname{index}(g)$, then $t_i = \operatorname{lt}(g)$, $w = \frac{\operatorname{slt}(vg)}{\operatorname{lt}(g)}$, and we get a syzygy
$$w(ge_i - ge_i) = 0 \in \mathcal{P}^s$$
- ▷ If $i > \operatorname{index}(g)$, then there exists $j < i$ such that $\pi(e_j) = g$, $w = \frac{\operatorname{slt}(vg)}{\operatorname{lt}(t_i)}$, $t_i = \operatorname{lt}(g_i)$, and we get
$$w(g_i e_j - ge_i).$$
It holds that $\operatorname{lm}(wge_i) \prec \operatorname{lm}(wg_ie_j)$. Thus we cannot assume to lower the signature resp. label of vg as it is done in the proof of Lemma 5.1.3. This is why we need to require the condition on the indices of the divisors found in S.

It is left is to see how (RW) is used in AP. This information is also given by Theorem 5.3.5: Whereas the first property is clearly fulfilled in our incremental approach based on $\prec_i$ as monomial order $\prec$ on the signatures, the second property is the interesting one:

On the one hand it requires only normal pairs to be considered. This is checked in AP by NonMinAP?. It also states that if there are two or more normal critical pairs of the same signature on the corresponding s–vector, we can freely consider just one of them. This is the rewritable criterion of AP. As mentioned already in Remark 5.1.5 one needs to choose which one of the multiple critical pairs corresponding to the same signature should be kept. In AP the critical pair is chosen, whose corresponding s–vector has minimal leading term.

Algorithm 41 AP's implementation of (RW) (RewriteAP?)

Input: uf a labeled polynomial multiplied by a term, vg a labeled polynomial multiplied by a term, $G = \{g_1, \dots, g_t\}$ a finite set of labeled polynomials, $P = \{p_1, \dots, p_k\}$ a finite set of critical pairs of labeled polynomials

Output: TRUE if $h \in \{uf, vg\}$ such that $\mathrm{sig}(h) = \max_{\prec}\left\{\mathrm{sig}(uf), \mathrm{sig}(vg)\right\}$ is detected by (RW), FALSE otherwise

1: **if** $\left(\mathrm{sig}(uf) \succ \mathrm{sig}(vg)\right)$ **then**
2:     $h \leftarrow f, w \leftarrow u$
3:     $\tilde{h} \leftarrow g, \tilde{w} \leftarrow v$
4: **else**
5:     $h \leftarrow g, w \leftarrow v$
6:     $\tilde{h} \leftarrow f, \tilde{w} \leftarrow u$
7: **for** $(j = 1, \dots, t)$ **do**
8:     **if** $\left(g_j \neq h \text{ and } \mathrm{sig}(g_j) \mid \mathrm{sig}(wh)\right)$ **then**
9:         $m \leftarrow \frac{st(wh)}{\mathrm{slt}(g_j)}$
10:         **if** $\left(m\,\mathrm{lt}(g) < \mathrm{lt}(uf - vg)\right)$ **then**
11:             **return** TRUE
12: **for** $(j = 1, \dots, k)$ **do**
13:     $(u'f', v'g') \leftarrow p_j$
14:     **if** $\left(\mathrm{sig}(u'f') \succ \mathrm{sig}(v'g')\right)$ **then**
15:         $h' \leftarrow f', w' \leftarrow u'$
16:         $\tilde{h}' \leftarrow g', \tilde{w}' \leftarrow v'$
17:     **else**
18:         $h' \leftarrow g', w' \leftarrow v'$
19:         $\tilde{h}' \leftarrow f', \tilde{w}' \leftarrow u'$
20:     **if** $\left(h' = h \text{ and } w' \mid w \text{ and } \tilde{h}' \neq \tilde{h}\right)$ **then**
21:         $m \leftarrow \frac{w}{w'}$
22:         **if** $\left(m\,\mathrm{lt}(u'f' - v'g') < \mathrm{lt}(uf - vg)\right)$ **then**
23:             **return** TRUE
24:         **else**
25:             Delete p_j from P
26:             break
27: **return** FALSE

Two main differences to Algorithm 38 can be found:

▷ In both checks done during the computations of RewriteAP? the multiplier m of the element in G resp. critical pair in P must be computed such that the leading terms can be compared (see Lines 9 and 21). If the leading term of the investigated critical pair is smaller, then it is not discarded, but computed. Considering the second check with elements of P one more step must be done: It is possible that the leading term of the investigated critical pair is smaller than those of the critical pair already in P. Then we must remove the critical pair from P and add the investigated one to it.

▷ Moreover, note that in Line 20 we only check for divisibility of $\mathrm{sig}(uf - vg)$ by $\mathrm{sig}(u'f' - v'g')$ and do no longer require equality. This enables us to discard a lot more critical pairs than Rewrite?.

Remark 5.3.8.

(1) Besides the optimizations discussed in (2), correctness and termination of AP follows easily from Theorem 4.2.7.

(2) Note that both, NonMinAP? and RewriteAP? have new properties we have not proved here so far. NonMinAP? checks both generators of the critical pair, whereas RewriteAP? needs only divisibility and not equality on signatures. The correctness of both optimizations of NonMin? resp. Rewrite? can be found in the proof of Theorem 18 in [7]. We see in Chapter 6 that F5 implements (NM) and (RW) even more aggressive, including the optimizations of AP mentioned here. We refer the reader to this part of the thesis for related proofs.

(3) Note that the pseudo code of RewriteAP? is not optimized at all. For example, elements whose signature leading term has index smaller than the current one are never detected by RewriteAP? as for any such wh there exists only $h \in G$ having the same index. Thus in Line 8 the if clause is never fulfilled for such elements.

5.4 A variant of AP using sparser polynomials

This section describes a short variant of AP (and thus of SigStdRed) preferring sparser polynomials. We have already seen that the only real choice one can make in signature–based standard basis algorithms is which critical pair to take in (RW). Clearly, some choices make sense, whereas others, like keeping the pair with largest leading term, does not.

The variant presented here was first mentioned by Perry and the author in [59]. We call it MM, short for *minimal number of monomials*, which describes the main idea: Having several elements of the same signature, then keep the sparsest element. This is in the vein of the ideas behind Brickenstein's SlimGB presented in Section 2.6. There, during the reduction steps, polynomials are dynamically exchanged with sparser equivalents w.r.t. the intermediate standard basis.

Definition 5.4.1. We denote the variant of SigStdRed calling IncSigCrit, NonMinAP?, and RewriteMM? by MM.

Algorithm 42 MM's implementation of (RW) (RewriteMM?)

Input: uf a labeled polynomial multiplied by a term, vg a labeled polynomial multiplied by a term, $G = \{g_1, \ldots, g_t\}$ a finite set of labeled polynomials, $P = \{p_1, \ldots, p_k\}$ a finite set of critical pairs of labeled polynomials

Output: TRUE if $h \in \{uf, vg\}$ such that $\mathrm{sig}(h) = \max_{\prec}\left\{\mathrm{sig}(uf), \mathrm{sig}(vg)\right\}$ is detected by (RW), FALSE otherwise

1: $\delta \leftarrow \#\left(\mathrm{supp}(\mathrm{poly}(uf - vg))\right)$
2: **if** $\left(\mathrm{sig}(uf) \succ \mathrm{sig}(vg)\right)$ **then**
3: $\quad h \leftarrow f, w \leftarrow u$
4: $\quad \tilde{h} \leftarrow g, \tilde{w} \leftarrow v$
5: **else**
6: $\quad h \leftarrow g, w \leftarrow v$
7: $\quad \tilde{h} \leftarrow f, \tilde{w} \leftarrow u$
8: **for** $\left(j = 1, \ldots, t\right)$ **do**
9: $\quad$ **if** $\left(g_j \neq h \text{ and } \mathrm{sig}(g_j) \mid \mathrm{sig}(wh)\right)$ **then**
10: $\quad\quad m \leftarrow \frac{st(wh)}{\mathrm{slt}(g_j)}$
11: $\quad\quad$ **if** $\left(\#\left(\mathrm{supp}(\mathrm{poly}(g_j))\right) < \delta\right)$ **then**
12: $\quad\quad\quad$ **return** TRUE
13: **for** $\left(j = 1, \ldots, k\right)$ **do**
14: $\quad (u'f', v'g') \leftarrow p_j$
15: $\quad$ **if** $\left(\mathrm{sig}(u'f') \succ \mathrm{sig}(v'g')\right)$ **then**
16: $\quad\quad h' \leftarrow f', w' \leftarrow u'$
17: $\quad\quad \tilde{h}' \leftarrow g', \tilde{w}' \leftarrow v'$
18: $\quad$ **else**
19: $\quad\quad h' \leftarrow g', w' \leftarrow v'$
20: $\quad\quad \tilde{h}' \leftarrow f', \tilde{w}' \leftarrow u'$
21: $\quad$ **if** $\left(h' = h \text{ and } w' \mid w \text{ and } \tilde{h}' \neq \tilde{h}\right)$ **then**
22: $\quad\quad m \leftarrow \frac{w}{w'}$
23: $\quad\quad$ **if** $\left(m\,\mathrm{lt}(u'f' - v'g') < \mathrm{lt}(uf - vg)\right)$ **then**
24: $\quad\quad\quad$ **return** TRUE
25: $\quad\quad$ **else**
26: $\quad\quad\quad$ Delete p_j from P
27: $\quad\quad\quad$ break
28: **return** FALSE

We give the pseudo code of this idea in Algorithm 42, again highlighting the differences to Algorithm 41.

The only difference can be found in Line 11 where we compare the number of monomials in $\mathrm{poly}(g_j)$ to those in $\mathrm{poly}(uf - vg)$, instead of comparing the leading terms. Clearly, $\#\left(\mathrm{supp}(\mathrm{poly}(g_j))\right) = \#\left(\mathrm{supp}(\mathrm{poly}(mg_j))\right)$, thus the check in Line 11 is consistent and saves the computation of multiplying $\mathrm{poly}(g_j)$ by m. As one can see there is no change to RewriteAP? in the second *for* loop of RewriteMM?. One can freely choose which critical pair to keep, but we found out that AP's choice is the most efficient in this situation. The question arises why we do not check for minimality of monomials there, too? The point is that one needs to store the lengths of the corresponding s–vectors of the critical pairs. The overhead for all those computations is too high to benefit from them. We see in Section 5.6 that despite the good idea of MM in general, the computations do not benefit from this decision, to the contrary, they get slower.

Clearly, correctness and termination of MM follows from those of AP.

5.5 G2V – Complete reduction, weakened (RW)

In 2010 Gao, Guan, and Volny presented their algorithm G2V in [76]. The algorithm is quite flexible in the sense that one cannot only compute the standard basis of an ideal $I = \langle p_1, \ldots, p_r \rangle \subset \mathcal{P}$, but also the ideal quotients $\langle p_1, \ldots, p_{i-1} \rangle : p_i$. This can also be done by any other signature–based standard basis algorithm (see Section 7.2 for more details). In this section we focus on the standard basis computation and see that G2V is nothing else but a rather straightforward implementation of SigStdRed with a complete sig–safe reduction and a softer variant of (RW).

Definition 5.5.1. We denote the variant of SigStdRed calling IncSigG2V, NonMinG2V?, and RewriteG2V? by G2V.

G2V, as stated in [76], does not keep any label for elements from the previously computed standard basis G_{i-1}, but sets them to zero. In more detail, G2V maintains two lists of polynomials, $U, V \in \mathcal{P}$. The elements the algorithm works with are pairs $(u, v) \in U \times V$, whereas u must be thought of as a signature of the polynomial v. So, assuming the beginning of the ith iteration of IncSigCrit, for each element $p_k \in G_{i-1}$ G2V generates the element $(0, p_k)$. The new element f_i from the initial set of generators of I is stored as $(1, f_i)$.

We can easily translate this into our setting, as illustrated in Algorithm 45, the implementation of IncSigCrit for G2V: All labeled polynomials g_k with polynomial part in G_{i-1} are initialized with $\operatorname{label}(g_k) = 0$ (see Line 6). The labeled polynomial corresponding to the generator f_i of I, entering the computations at this point, is generated as usual by $g_s = (e_s, p_s)$ (see Line 8). The same holds for all newly computed labeled polynomials of the current index s (see Line 32). We can define a map

$$\begin{array}{rccc} \psi: & \mathcal{P}^s \times \mathcal{P} & \longrightarrow & \mathcal{P} \times \mathcal{P} \\ & g & \longmapsto & \big(\operatorname{slt}(g), \operatorname{poly}(g)\big), \end{array}$$

which does nothing else but to extract the term out of the signature of the labeled polynomial, i.e. it forgets the index. Due to the fact that G2V does only keep data $u \neq 0$ if the corresponding polynomial is of current index, we have a one–to–one correspondence between

(1) labeled polynomials, whose labels are not equal to 0 if and only if their index is maximal, and

(2) pairs of polynomials in $U \times V \subset \mathcal{P} \times \mathcal{P}$.

It follows that we can keep our notation of labeled polynomials and discuss G2V in our setting. Having unified notations it is left to compare the main parts of SigStdRed and G2V, namely the reduction process and the criteria checks.

In G2V a new notion is defined, the *super top–reduction*. Again, we can easily translate this to our notation using labeled polynomials.

Definition 5.5.2. A labeled polynomial f is called *super top–reducible* if there exist a labeled polynomial g and a term $t \in \mathcal{P}$ such that

$$t \operatorname{lt}(g) = \operatorname{lt}(f) \quad \text{and} \quad t \operatorname{sig}(g) = \operatorname{sig}(f).$$

In G2V those super top–reductions are not allowed in the sig–safe reduction steps. Thus we get a slightly different implementation of SigRed for G2V:

Algorithm 43 G2V's sig–safe reduction algorithm (SigRedG2V)

Input: f a labeled polynomial, $G = \{g_1, \ldots, g_s\}$ a finite set of labeled polynomials
Output: h a labeled polynomial sig–safe reduced w.r.t. G

1: $s \leftarrow \operatorname{siglm}(f)$
2: $c \leftarrow \operatorname{slc}(f)$
3: $p \leftarrow \operatorname{poly}(f)$
4: **while** $\left(p \neq 0 \text{ and } D_p \leftarrow \left\{g \in G \mid \operatorname{lm}(\operatorname{poly}(g)) \mid \operatorname{lm}(p)\right\} \neq \emptyset\right)$ **do**
5: Choose any $g \in D_p$.
6: $u \leftarrow \frac{\operatorname{lt}(p)}{\operatorname{lt}(\operatorname{poly}(g))}$
7: **if** $\left(\operatorname{lm}(u) \operatorname{siglm}(g) \prec s\right)$ **then**
8: $p \leftarrow p - u \operatorname{poly}(g)$
9: **else if** $\left(\operatorname{lm}(u) \operatorname{siglm}(g) = s \text{ and } \operatorname{lc}(u) \neq c\right)$ **then**
10: $p \leftarrow p - u \operatorname{poly}(g)$
11: $c \leftarrow c - \operatorname{lc}(u)$
12: $h \leftarrow \left(c \operatorname{siglm}(f), p\right)$
13: **return** h

As one can see the only real difference between SigRed and SigRedG2V is given in Lines 9–11. Instead of checking only

$$\operatorname{lm}(u) \operatorname{siglm}(g) \prec \operatorname{siglm}(f)$$

as it is done in SigRed, SigRedG2V also checks the whole signature including the coefficients. So the only reductions which do not take place are sig–unsafe or super top–reductions. Looking again at Definition 4.1.13 we see that this is just the difference between a complete and a semi–complete sig–safe reduction. Moreover, we have shown in Lemma 4.2.5 that there cannot be a complete sig–safe reduction in SigRedG2V without a semi–complete sig–safe reduction in SigRed. Thus comparing the coefficients (Line 9) and adjusting them (Line 11) enables us to do complete sig–safe reductions. In the end, Algorithm 43 returns the reduced labeled polynomial with possibly adjusted label resp. signature $c\,\mathrm{siglm}(f)$ (Line 12). It could be possible that a reduction of f by some element g, which SigRed would not allow, can be processed in SigRedG2V, but there always exists an element $h \in G$ such that f can be reduced by h in SigRed in this situation. So the only question that arises is the one of the *better* reducer at this point of the computations, g or h?

Let us get back to the essential point that G2V does not store any label for elements of G_{i-1}. This is a small change, since all situations where the label resp. its leading term is important it is checked, if the signature of an element is of current index. The signatures of the elements from G_{i-1} have by definition smaller index and thus are not considered NonMin? or Rewrite? at all. Setting the labels of those elements to 0 leads to three main differences to current index elements:

(1) They are not checked by NonMin?, since their signature is 0 and 0 is not divisible by any term in $\mathcal{P}$.

(2) They are checked neither by Rewrite? as there exists no other element of the same index in the current iteration round.

(3) It is safe to reduce with these elements as the leading term of their signature is always smaller than the leading term of the signature of the element to be reduced (which has the current index).

In fact, G2V implements (NM) just like the generic signature–based algorithm in Section 5.1 does. So we can apply the following equality:

$$\text{NonMinG2V?} \quad = \quad \text{NonMin?}\,.$$

As a last step in our discussion of G2V we need to discuss its implementation of (RW). At a first glance it seems that G2V has not implemented (RW) at all, since it is not mentioned in [76]. Having a closer look at the Singular library source code Gao, Guan, and Volny have made publicly available at

`http://www.math.clemson.edu/~sgao/code/g2v.sing`

one can find a rather soft implementation of (RW) at the point where new critical pairs are generated. This leads to the fact that IncSigG2V calls RewriteG2V? only in Line 13 and Line 29 (see Algorithm 45), but not in Line 17, where only NonMinG2V? is called. This lies in the nature of the softer implementation of RewriteG2V?: In G2V (RW) detects a useless critical pair if and only if for a newly generated critical pair (uf, vg) there is another critical pair $(u'f', v'g')$ in the pair set P such that

$$\max_{\prec}\left\{\,\mathrm{sig}(uf), \mathrm{sig}(vg)\right\} = \max_{\prec}\left\{\,\mathrm{sig}(u'f'), \mathrm{sig}(v'g')\right\}.$$

In this situation G2V keeps only one of the two critical pairs (by sig–redundancy it is clear that only one of these is needed). G2V assumes that the newly generated pair (uf, vg) has some better properties than $(u'f', v'g')$, thus it deletes $(u'f', v'g')$ from the pair set and inserts (uf, vg) later on. In Algorithm 44 we present the pseudo code of RewriteG2V?. Due to the fact that it is quite stripped down we do not highlight changed lines, but give a complete new pseudo code. One can see that a check is done with quite less comparisons and multiplications. Note that it does not check (uf, vg) with elements already in G as it is done in Rewrite?, RewriteAP?, and RewriteMM?, thus less useless critical pairs can be detected.

Algorithm 44 G2V's implementation of (RW) (RewriteG2V?)

Input: uf a labeled polynomial multiplied by a term, vg a labeled polynomial multiplied by a term, $P = \{p_1, \ldots, p_k\}$ a finite set of critical pairs of labeled polynomials
Output: FALSE

$s \leftarrow \max_{<} \left\{ \operatorname{sig}(uf), \operatorname{sig}(vg) \right\}$
for $\left(j = 1, \ldots, k \right)$ **do**
  $(u'f', v'g') \leftarrow p_j$
  $t \leftarrow \max_{<} \left\{ \operatorname{sig}(u'f'), \operatorname{sig}(v'g') \right\}$
  if $(s = t)$ **then**
    Delete p_j from P
    break
return FALSE

Remark 5.5.3.

(1) RewriteG2V? always returns FALSE due to its description strongly related to Rewrite?. Whenever a critical pair is detected, the one in P is deleted, but the actual critical pair has to be added to P later on. For the sake of unifying notations and letting the theoretical changes affect the pseudo code as less as possible we keep the boolean framework of Rewrite? also for RewriteG2V?.

(2) Note that the proof of correctness of G2V is straightforward using the results of Section 5.1. Moreover, termination of G2V is proven by Corollary 5.1.7, too. This proof appeared initially in [59] and is the first publicly available proof of G2V's termination.

This finishes our discussion about G2V, which is the last variant of SigStd presented at this point. We close this Chapter with an extensive comparison of all presented variants of SigStd resp. SigStdRed.

Algorithm 45 G2V's implementation of IncSigCrit (IncSigG2V)

Input: f_i a polynomial, $G_{i-1} = \{p_1, \ldots, p_{s-1}\}$ a standard basis for $\langle f_1, \ldots, f_{i-1} \rangle$
Output: B a standard basis for $\langle f_1, \ldots, f_i \rangle$ w.r.t. $<$

$B \leftarrow \varnothing, G \leftarrow \varnothing, P \leftarrow \varnothing$
$S \leftarrow \varnothing$
$p_s \leftarrow f_i$
$t \leftarrow s$
for $(k = 1, \ldots, s-1)$ **do**
  $g_k \leftarrow (0, p_k)$
  $S \leftarrow S \cup \{\mathrm{lt}(p_k)\}$
$g_s \leftarrow (e_s, p_s)$
$G \leftarrow \{g_1, \ldots, g_s\}$
for $(k = 1, \ldots, s-1)$ **do**
  $u \leftarrow \mathrm{lc}(g_k) \frac{\tau(g_s, g_k)}{\mathrm{lm}(g_s)}$
  $v \leftarrow \mathrm{lc}(g_s) \frac{\tau(g_s, g_k)}{\mathrm{lm}(g_k)}$
  if $\big($! NonMinG2V? $(u g_s, v g_k, S)$ and ! RewriteG2V? $(u g_s, v g_k, P)\big)$ **then**
    $P \leftarrow P \cup \{(u g_s, v g_k)\}$
while $(P \neq \varnothing)$ **do**
  Choose (uf, vg) from P with $\max_{\prec} \{u\, \mathrm{sig}(f), v\, \mathrm{sig}(g)\}$ minimal w.r.t. $\prec$.
  if $\big($! NonMinG2V?$(uf, vg, S)\big)$ **then**
    $P \leftarrow P \setminus \{(uf, vg)\}$
    $l \leftarrow \max_{\prec} \{u\, \mathrm{sig}(f), v\, \mathrm{sig}(g)\}$
    $r \leftarrow \big(l, u\, \mathrm{poly}(f) - v\, \mathrm{poly}(g)\big)$
    $r \leftarrow$ SigRedG2V(r, G)
    if $(\mathrm{poly}(r) = 0)$ **then**
      $S \leftarrow S \cup \{\mathrm{slt}(r)\}$
    else if $\big(\mathrm{poly}(r) \neq 0$ and r not sig-redundant w.r.t. $G\big)$ **then**
      for $(k = 1, \ldots, t)$ **do**
        $u \leftarrow \mathrm{lc}(g_k) \frac{\tau(r, g_k)}{\mathrm{lm}(r)}$
        $v \leftarrow \mathrm{lc}(r) \frac{\tau(r, g_k)}{\mathrm{lm}(g_k)}$
        if $\big(\mathrm{lm}(u)\, \mathrm{siglm}(r) \neq \mathrm{lm}(v)\, \mathrm{siglm}(g_k)\big)$ **then**
          if $\big($! NonMinG2V? $(ur, v g_k, S)$ and ! RewriteG2V? $(ur, v g_k, P)\big)$ **then**
            $P \leftarrow P \cup \{(ur, v g_k)\}$
      $t \leftarrow t + 1$
      $g_t \leftarrow r$
      $G \leftarrow G \cup \{g_t\}$
$B \leftarrow \mathrm{poly}(G)$
return B

5.6 Experimental results

After we have given lots of different variants of SigStd we want to compare them. Before we can do this a small interlude of how we compare the algorithms shall give the reader an idea on the accuracy we try to achieve.

All algorithms presented throughout sections 5.1 – 5.5 differ only in minor parts, most of the time only in their implementation of (NM) and (RW). Thus the following underlying structure of our test suite makes sense: We have implemented the overall, generic structure of SigStd resp. SigStdRed without using any criteria (see Section 4.2). Keeping this implementation without any changes to data structures, polynomial representations, etc., we have added the corresponding implementations of (NM) and (RW) for SigStd resp. SigStdRed with generic criteria, AP, MM, and G2V. Using this the best possible comparison can be done. The source code of the different algorithms distinguish in at most 127 lines of code, compared to nearly 3,300 lines of code overall quite neglectable.

The algorithms are implemented in the Singular kernel in the programming language C++. Note that the code is open source and publicly available at

`git@github.com:ederc/Sources.git`[3].

The implementation is done not only in the most optimized way to compare the different algorithms, but also focusses on the efficiency of the computations. Still we should note the following.

Remark 5.6.1. The implementation presented in this section is not intended to be comparable with Singular 's highly efficient and optimized standard basis algorithm implementation. Our implementation is slower due to the following facts:

(1) We do not want to optimize any part of the algorithm all variants are sharing due to the problem that one of the variants could take more an advantage out of this than another variant.

(2) All signature–based algorithms presented here, being derivates of F5, as we see in the next chapter, have to cope with a problem their functioning is based on: They use an incremental structure, which can slow done computations due to not using all input data in an optimal way. We show that this is a field of high research in the signature–based world these days

In Section 7.4 we give more insight in this area, which is of great importance in the signature–based world.

All examples where computed on a computer with the following specifications:

▷ 2.6.31–gentoo–r6 GNU/Linux 64–bit operating system,

▷ INTEL® XEON® X5460 @ 3.16GHz processor,

[3] You can get the git repository by typing `git clone git@github.com:ederc/Sources.git`. Each algorithm has its own branch.

Figure 5.6.1: Coloration of results for variants of SigStd

- ▷ 64 GB of RAM, and
- ▷ 120 GB of swap space.

For all computations we used the latest[4] developer version of Singular 3–1–3, revision 14,372 in the SVN trunk available at

`http://www.singular.uni-kl.de/svn/trunk/`.

A complete list of the test cases can be found in Appendix A. In this series of tests we always compute in the respective polynomial ring over a field of characteristic 32, 003 using the graded reverse lexicographical order $<_{\mathrm{dp}}$.

The series of examples we give results for in the following cover different settings, from complete intersections to overdetermined systems, from inhomogeneous to homogeneous input data.

In Figure 5.6 we explain how the different colors of the results presented in the tables have to be interpreted: The best results are always written in blue, the worst in red. As we give not only timings, but also memory consumption, and various other data, we cannot be more precise with terms like "best" or "worse". This should be no problem for the reader as it is clear from the context of the table.

The table which can be understood easiest is Table 5.2: It shows the number of zero reductions not rejected in each of the 5 algorithms during the corresponding computations. As one can see all algorithms share the same number of zero reductions. This can be interpreted in the following way: The implementation of (NM), which is equal in all variants discards nearly all critical pairs which would lead to a zero reduction. The differences in the implementation of (RW) does not alter the behaviour of the algorithms w.r.t. zero reductions. Note that even SigStd, which does not interreduce the intermediate standard bases, does not compute more zero reductions. This again is based on the strength of NonMin?.

Looking at the timings in Table 5.1 an overall statement is the following: AP and G2V are the fastest of the 5 given algorithms. Whereas G2V is mostly the fastest or second fastest algorithm, AP sometimes loses track, e.g. `F-744(-h)` and `Katsura-11(-h)`. On the other hand, AP is way faster, even than G2V, in examples like `Cyclic-7(-h)` and `Eco-x(-h)`.

Clearly, SigStd must handle all the overhead of not interreducing intermediate standard bases which slows down the algorithm noticeable. Its timings are getting better with

[4] It is the current developer version of Singular at the point we *start* the first computation of the series. To keep the computations comparable we fix this revision number of Singular for all experimental results given in this thesis.

the Katsura-x(-h) examples, as those are regular sequences, where no zero reduction at all takes place and any useless critical pair is already handled by (NM). There, especially in the smaller test sets, SigStd benefits from the overhead of computational time needed interreducing intermediate standard bases which is done by the other algorithms only.

Moreover, one should note that, besides AP which actively looks at the polynomial leading terms in its (RW) implementation, all other algorithms really slow down when computing standard bases of inhomogeneous ideals. In Section 7.1 we give more details on this behaviour of signature–based algorithms in general. Note that the algorithms, as implemented, are not able to compute the inhomogeneous example Eco-11 in a reasonable amount of time, whereas a standard basis for Eco-11-h can be given.

Table 5.3 is the quite opposite of Table 5.1 at first esteem. Giving it a closer look it turns out that nearly all algorithms behave the very similar to the timings table, but AP. In nearly all examples AP needs the most memory during the computations. Comparing this with its quite good timings, this is a strange behaviour and needs some clarification: As explained in the prelude of this section we have implemented all 5 algorithms based on one basic underlying framework. We changed only very few lines of code for the different algorithms to ensure a comparison of the different ways (NM) and (RW) are implemented as precise as possible. However, AP compares the leading terms of the critical pairs for its implementation of (RW). In a specialized implementation one would keep this data stored in the structure of the critical pair. Since we decided to use the same data structures for all 5 algorithms, we cannot do this in AP and need to compute the leading terms again and again when checking (RW). Whereas it does not cost much time to get the leading term out of the critical pair, comparing two terms consumes memory to store those terms and the result of the comparison. Calculating roughly the number of calls of RewriteAP? this explains the memory overhead of AP compared to the other 4 algorithms.

Tables 5.4 and 5.5 need to be taken into account together. Whereas the first one presents the number of critical pairs not detected by any criterion in the algorithm, and thus further processed, the later one gives us the complete number of all single reductions steps that have taken place during the computations. The clear winner of these two properties is AP: It detects the most useless critical pairs and does the fewest reduction steps of all algorithms. The fact that it discards the most critical pairs follows from its sophisticated implementation of (RW). The fact that it computes so much less single reduction steps is not only based on the fact that it handles less critical pairs, but it is also a consequence of AP's implementation of (RW) keeping the elements of lowest possible leading term. This has quite astonishing effects as we can see in Table 5.5: In nearly all examples AP computes less than half the number of reduction steps the other algorithms do. Thinking of MM as being just a variant of AP preferring sparser polynomials the differences in the results are quite big. MM performs even worse than G2V in most of the examples. The main problem of MM is that favouring sparse polynomials it keeps critical pairs of the first labeled polynomials of the iteration steps, since those have the fewest terms in general. This leads to a recomputation of reduction steps the generators of other critical pairs, which e.g. AP keeps instead, have already been undergone.

As a last criterion to distinguish the given algorithms we present the number of elements in the resulting standard bases. Note that SigStd does not interreduce intermediate standard bases, whereas the other does. However, no algorithm reduces the result of the last iteration of IncSigCrit.

Clearly it follows that the number of elements in G computed by SigStd is always the largest one. The astonishing outcome of Table 5.6 are the numbers of the other 4 algorithms: they are all the same! This means that their differences in (RW) does not have an effect on the number of elements computed at all. All start with the very same reduced standard basis as input for their last iteration step. In there, the same number of elements is added to G. This again shows the strength that lies in (NM), but also the impact of the restricting sig–safe reduction process.

With this we finish our discussion of this canonical derived signature–based standard basis algorithms. We have presented different possible attempts and compared them in some basic features that are most interesting in terms of standard basis computations. Problems with inhomogeneous computations as well as the incremental structure of the algorithms are discussed in detail in Section 7.1 resp. 7.4. Next we start an in–depth characterization of Faugère's F5 Algorithm, which turns out to be a way more aggressive variant of SigStd.

Test case	SigStd	SigStdRed	AP	MM	G2V
Cyclic-7-h	67.180	41.280	6.090	31.900	26.900
Cyclic-7	66.210	40.230	5.950	31.230	26.350
Cyclic-8-h	73, 903.890	21, 090.780	14, 645.590	15, 470.990	13, 991.890
Cyclic-8	69, 356.880	19, 468.020	14, 078.000	14, 369.790	12, 973.530
Eco-8-h	0.470	0.450	0.210	0.440	0.410
Eco-8	0.500	0.490	0.080	0.490	0.480
Eco-9-h	13.440	13.020	2.960	12.870	11.870
Eco-9	29.400	28.950	1.630	29.020	29.110
Eco-10-h	420.520	418.410	127.190	407.930	386.680
Eco-10	1, 548.090	1, 554.970	67.410	1, 526.530	1, 492.900
Eco-11-h	14, 948.500	14, 973.920	4, 521.250	14, 486.700	13, 691.810
F-633-h	0.000	0.000	0.010	0.010	0.000
F-633	0.000	0.010	0.000	0.000	0.000
F-744-h	62.330	40.020	44.910	39.200	34.150
F-744	52.840	31.580	42.700	29.370	28.760
F-855-h	2, 414.730	1, 349.680	492.980	1, 266.440	1, 279.630
F-855	7, 005.730	1, 844.650	182.390	1, 437.890	1, 071.850
Gonnet-83-h	37.620	12.760	8.920	11.170	10.890
Katsura-8-h	0.050	0.070	0.050	0.050	0.050
Katsura-8	0.060	0.060	0.050	0.060	0.040
Katsura-9-h	0.560	0.580	0.450	0.550	0.400
Katsura-9	0.540	0.580	0.440	0.540	0.400
Katsura-10-h	5.820	6.220	5.170	6.590	4.430
Katsura-10	5.730	6.150	5.110	6.490	4.360
Katsura-11-h	69.900	84.210	74.050	84.100	62.600
Katsura-11	65.170	76.860	66.590	77.540	56.760
Schrans-Troost-h	6.020	6.580	3.160	6.110	4.590

Table 5.1: Time needed to compute a standard basis, given in seconds.

Test case	SigStd	SigStdRed	AP	MM	G2V
Cyclic-7-h	36	36	36	36	36
Cyclic-7	36	36	36	36	36
Cyclic-8-h	244	244	244	244	244
Cyclic-8	244	244	244	244	244
Eco-8-h	57	57	57	57	57
Eco-8	0	0	0	0	0
Eco-9-h	120	120	120	120	120
Eco-9	0	0	0	0	0
Eco-10-h	247	247	247	247	247
Eco-10	0	0	0	0	0
Eco-11-h	502	502	502	502	502
F-633-h	2	2	2	2	2
F-633	0	0	0	0	0
F-744-h	323	323	323	323	323
F-744	0	0	0	0	0
F-855-h	835	835	835	835	835
F-855	0	0	0	0	0
Gonnet-83-h	2,005	2,005	2,005	2,005	2,005
Katsura-8-h	0	0	0	0	0
Katsura-8	0	0	0	0	0
Katsura-9-h	0	0	0	0	0
Katsura-9	0	0	0	0	0
Katsura-10-h	0	0	0	0	0
Katsura-10	0	0	0	0	0
Katsura-11-h	0	0	0	0	0
Katsura-11	0	0	0	0	0
Schrans-Troost-h	0	0	0	0	0

Table 5.2: Number of zero reductions computed by the algorithms.

Test case	SigStd	SigStdRed	AP	MM	G2V
Cyclic-7-h	305.093	216.132	638.073	180.627	155.201
Cyclic-7	305.594	216.137	637.573	180.627	155.202
Cyclic-8-h	40,514.258	14,550.527	70,500.524	11,049.046	10,387.808
Cyclic-8	34,775.470	12,494.244	60,444.643	9,490.331	8,917.106
Eco-8-h	22.547	21.576	45.065	21.076	20.600
Eco-8	12.515	11.529	12.026	11.529	12.036
Eco-9-h	186.765	179.826	350.688	177.826	176.957
Eco-9	138.072	134.605	90.595	134.105	139.133
Eco-10-h	1,588.217	1,552.831	2,982.387	1,533.829	1,547.097
Eco-10	1,384.822	1,367.438	666.817	1,353.436	1,394.078
Eco-11-h	13,695.384	13,525.094	26,206.020	13,279.561	13,376.611
F-633-h	0.000	0.036	0.536	0.036	0.036
F-633	0.000	0.035	0.535	0.035	0.034
F-744-h	368.775	250.779	458.346	236.260	207.322
F-744	174.631	102.110	133.606	92.110	84.134
F-855-h	4,735.116	3,199.016	5,876.375	3,101.008	3,065.703
F-855	4,299.398	2,361.621	1,228.474	1,904.102	1,518.127
Gonnet-83-h	102.887	81.870	193.013	78.870	76.963
Katsura-8-h	2.500	2.000	8.000	2.000	1.500
Katsura-8	2.500	2.000	8.000	2.000	1.500
Katsura-9-h	9.500	7.000	41.000	7.000	6.500
Katsura-9	9.500	7.000	41.000	7.000	6.500
Katsura-10-h	39.500	28.516	212.046	28.016	25.521
Katsura-10	39.500	28.516	212.546	28.016	25.521
Katsura-11-h	185.545	129.085	1,295.758	127.585	116.098
Katsura-11	162.542	113.583	1,133.735	111.583	101.598
Schrans-Troost-h	45.028	31.537	113.555	31.037	29.541

Table 5.3: Memory used to compute a standard basis, given in Megabyte.

Test case	SigStd	SigStdRed	AP	MM	G2V
Cyclic-7-h	4,868,209	3,194,383	93,742	2,407,166	1,915,620
Cyclic-7	4,868,209	3,194,383	93,742	2,407,166	1,915,620
Cyclic-8-h	624,588,332	189,526,325	49,444,223	127,025,633	118,873,460
Cyclic-8	624,588,332	189,526,325	49,444,223	127,025,633	118,873,460
Eco-8-h	72,348	72,140	15,583	70,961	69,087
Eco-8	72,138	72,107	10,161	71,124	70,144
Eco-9-h	673,572	672,973	112,285	659,967	654,585
Eco-9	862,776	862,683	83,911	847,575	824,967
Eco-10-h	6,259,660	6,258,064	904,936	6,083,158	6,099,258
Eco-10	10,201,164	10,200,912	869,101	9,927,104	9,580,146
Eco-11-h	56,484,340	56,480,296	7,787,226	54,221,419	55,307,100
F-633-h	671	664	389	637	681
F-633	561	554	319	556	540
F-744-h	2,203,159	1,623,992	636,295	1,458,884	1,103,524
F-744	1,224,352	748,585	249,228	606,369	481,477
F-855-h	38,155,078	25,114,845	5,408,786	23,749,896	23,212,624
F-855	39,102,121	19,477,654	1,749,296	12,941,981	8,085,517
Gonnet-83-h	180,061	145,494	84,203	121,245	120,296
Katsura-8-h	3,694	3,647	1,626	3,416	1,780
Katsura-8	3,694	3,647	1,626	3,416	1,780
Katsura-9-h	14,950	14,857	5,309	13,633	5,729
Katsura-9	14,950	14,857	5,309	13,633	5,729
Katsura-10-h	57,479	58,495	17,868	55,317	19,403
Katsura-10	57,479	58,495	17,868	55,317	19,403
Katsura-11-h	238,219	240,294	60,965	213,224	66,760
Katsura-11	238,219	240,294	60,965	213,224	66,760
Schrans-Troost-h	59,817	57,127	14,167	52,946	19,628

Table 5.4: Number of all reduction steps during the computations.

Test case	SigStd	SigStdRed	AP	MM	G2V
Cyclic-7-h	3,496	3,072	914	3,072	3,072
Cyclic-7	3,496	3,072	914	3,072	3,072
Cyclic-8-h	37,260	24,600	20,086	24,600	24,600
Cyclic-8	37,260	24,600	20,086	24,600	24,600
Eco-8-h	2,147	2,012	694	2,012	2,012
Eco-8	821	796	398	796	796
Eco-9-h	6,141	5,794	1,852	5,794	5,794
Eco-9	1,997	1,933	954	1,933	1,933
Eco-10-h	17,379	16,535	5,148	16,535	16,535
Eco-10	4,742	4,587	2,337	4,587	4,587
Eco-11-h	49,079	47,105	14,994	47,105	47,105
F-633-h	80	80	56	80	80
F-633	74	74	54	74	74
F-744-h	4,090	3,451	2,221	3,451	3,452
F-744	1,467	1,280	899	1,280	1,280
F-855-h	16,442	14,197	6,532	14,197	14,197
F-855	7,206	6,365	3,309	6,365	6,366
Gonnet-83-h	12,407	10,241	7,573	10,241	10,248
Katsura-8-h	120	120	120	120	120
Katsura-8	120	120	120	120	120
Katsura-9-h	247	247	247	247	247
Katsura-9	247	247	247	247	247
Katsura-10-h	502	502	502	502	502
Katsura-10	502	502	502	502	502
Katsura-11-h	1,013	1,013	1,013	1,013	1,013
Katsura-11	1,013	1,013	1,013	1,013	1,013
Schrans-Troost-h	469	461	397	461	461

Table 5.5: Number of critical pairs not detected by the respective criteria used.

Test case	SigStd	SigStdRed	AP	MM	G2V
Cyclic-7-h	749	658	658	658	658
Cyclic-7	749	658	658	658	658
Cyclic-8-h	3,865	2,611	2,611	2,611	2,611
Cyclic-8	3,865	2,611	2,611	2,611	2,611
Eco-8-h	356	249	249	249	249
Eco-8	294	187	187	187	187
Eco-9-h	721	499	499	499	499
Eco-9	595	373	373	373	373
Eco-10-h	1,438	979	979	979	979
Eco-10	1,184	725	725	725	725
Eco-11-h	2,901	1,968	1,968	1,968	1,968
F-633-h	58	56	56	56	56
F-633	58	56	56	56	56
F-744-h	1,252	204	204	204	204
F-744	694	87	87	87	87
F-855-h	2,277	688	688	688	688
F-855	2,012	148	148	148	148
Gonnet-83-h	2,673	1,035	1,035	1,035	1,035
Katsura-8-h	128	105	105	105	105
Katsura-8	128	105	105	105	105
Katsura-9-h	256	202	202	202	202
Katsura-9	256	202	202	202	202
Katsura-10-h	512	399	399	399	399
Katsura-10	512	399	399	399	399
Katsura-11-h	1,024	784	784	784	784
Katsura-11	1,024	784	784	784	784
Schrans-Troost-h	398	189	189	189	189

Table 5.6: Size of the resulting standard basis.

6 Faugère's F5 Algorithm

In the proceedings of the `ISSAC'02` conference Faugère published his F5 Algorithm ([63]). This algorithm is nothing else but the ancestor of all (incremental) signature–based algorithms presented in this thesis. Nevertheless F5 can be understood as the most aggressive variant of SigStd, as we see in the following. Due to this fact we decided to discuss F5 *after* we have already discussed the other variants, although F5 is scheduled *before* each algorithm presented in Chapter 5.

F5 is famous for its great performance computing standard bases. Especially in the situation of regular sequences as input, F5's criteria rejecting useless data are very powerful, not computing any zero reduction at all. We see that this holds for all variants of SigStd previously mentioned, too. Moreover, using his initial implementation of F5, Faugère was the first who computed a standard basis for `Cyclic-10` over a field of characteristic p, p prime. Also in the field of cryptography and cryptanalysis F5 is well–known, e.g. for breaking some previously untractable HFE systems ([19, 64, 68]).

There are a lot of open questions left after reading [63]. Whereas Faugère presented his ideas in full, some gaps in the essential proofs as well as some issues considering performance were given. First improvements in understanding F5 were done by Stegers in [144].

In this chapter F5 is not only presented and explained in detail, we also show the connections and differences to the algorithms presented in Chapter 5.

The main points of our discussion are:

(1) How to prove the correctness of F5? There exist various attempts of this task, but none of them provide a gapless and correct proof. We give a complete proof together with an in–depth presentation of F5 in Section 6.1.

(2) Do there exist optimizations of F5? We present various in this chapter, the most signifcant being the variant F5C in Section 6.2. It shows how to reduce computational overhead to a minimum and improve F5's performance by a large factor.

(3) One of the hardest problems around F5 is to prove its termination. Until now no complete and correct proof of this exists to the knowledge of the author. Different variants ensuring termination are already known, but all of them lack F5's performance due to introducing a huge overhead in the computations. In Section 6.5 we present another variant of F5, ensuring termination with no penalty on performance at all.

The author has tackled the above mentioned problems in [55, 56], together with John Perry in [58], and together with Justin Gash and John Perry in [57]. The ideas discussed in this chapter are initially presented in those publications, whereby lots of generalizations of these attempts are presented for the first time in this thesis.

Some other ideas of generalization and optimization, sometimes just specializations of our research can be found in various publications of the last years, e.g. [8–10,71,78,148,163] et al. Signature–based standard basis algorithms are a field of high research these days, a lot of new ideas and constructions can be expected in the near future, understanding the special behaviour of these algorithms more and more.

6.1 Faugère's initial presentation of F5

In this section we start our discussion of F5 giving the algorithm in its initial form as presented in [63]. Explaining the underlying ideas in constructing an efficient standard basis algorithm we use the notations previously defined in Chapters 4 and 5. Based on this we can easily use already found aspects and compare F5 with SigStd and its variants.

Since we guess that most readers are either not at all familiar with F5 or possibly only aware of [63], we focus on introducing F5 in this vein. A complete classification of the algorithm in the field of signature–based standard basis algorithms is postponed to Section 6.3.

In this section we give a full proof of the correctness of F5, including both its criteria used. John Perry and the author are the first who published a complete proof in [58].

Due to our intention to present F5 as much as possible unaltered in this section, we need to agree on the following:

Convention. In this section we restrict ourselves to homogeneous polynomials and ideals in $\mathcal{P}$. So the input $F = \{f_1, \ldots, f_r\}$ of the presented algorithms are always homogeneous.

Remark 6.1.1. Note that in [63] a slightly different module order is used,

$$\begin{aligned} \boldsymbol{x}^\alpha e_i <_{\text{neg-i}} \boldsymbol{x}^\beta e_j :\Longleftrightarrow\ & i > j \text{ or,} \\ & i = j \text{ and } \boldsymbol{x}^\alpha < \boldsymbol{x}^\beta. \end{aligned}$$

We keep stuck to $<_{\text{i}}$ being the order on the signatures. The main difference between $<_{\text{neg-i}}$ and $<_{\text{i}}$ concerning F5 is that we compute incrementally standard bases for $\langle f_1 \rangle$, $\langle f_1, f_2 \rangle$, etc., whereas Faugère's F5 goes the other way around starting with a basis for $\langle f_r \rangle$, $\langle f_r, f_{r-1} \rangle$, etc. So this is only a difference in notation, but not in the mathematical approach and should not irritate the reader at all.

The F5 Algorithm is an incremental standard basis algorithm, in particular, we can define a main loop iterating over all elements f_i of the input data:

Algorithm 46 The F5 Algorithm(F5)

Input: $F = \{f_1, \ldots, f_r\}$ a finite subset of $\mathcal{P}$
Output: B a standard basis for $\langle F \rangle$ w.r.t. $<$

1: $G_1 \leftarrow \{(e_1, f_1)\}$
2: S = empty list
3: R = empty list
4: **for** $(i = 2, \ldots, r)$ **do**
5: $\quad f_i \leftarrow$ REDUCE $(f_i, \text{poly}(G_{i-1}))$
6: $\quad$ **if** $(f_i \neq 0)$ **then**
7: $\quad\quad G_i, S, R \leftarrow$ INCF5(f_i, G_{i-1}, S, R)
8: $\quad$ **else**
9: $\quad\quad G_i \leftarrow G_{i-1}$
10: $B \leftarrow \text{poly}(G_r)$
11: **return** B

Algorithm 46 coincides with Algorithm 32 besides some small, but quite essential differences:

▷ The `while` loop runs over sets of labeled polynomials G_i, so G_1 is initialized in Line 1 by $G_1 = \{(e_1, f_1)\}$.

▷ In Line 7 we do not call INCSIG or INCSIGCRIT, but the incremental F5 routine denoted INCF5. This differs from the previously presented incremental algorithms mostly in its usage of the signature–based criteria, which we explain in detail in the following. At this point it is important that the kth call of INCF5 returns a set of labeled polynomials G such that $\text{poly}(G)$ is a standard basis for $\langle f_1, \ldots, f_k \rangle$. This is a very important change since it enables us to reuse signatures computed during previous iteration steps in upcoming one.

- ▷ Due to the fact of using sets of labeled polynomials, we need to extract the polynomial part of G_r at the end. With this we return a set of polynomials which is just a standard basis for $I = \langle f_1, \ldots, f_r \rangle$ with respect to the given monomial order $<$.

- ▷ S is a list of list of terms in $\mathcal{P}$. It is used for F5's implementation of (NM). One should think of it as a more structured way of storing the needed criteria. This more in structure can be used to implement (NM) way more aggressive as we show in the following.

- ▷ R is some data structure describing lists of terms in $\mathcal{P}$. Those are needed for F5's implementation of (RW) in RewriteF5?. It is initialized to the empty set in Line 3. We give a detailed explanation when discussing the Rewritten Criterion below. In the meantime the reader should just think about some data which is updated in IncF5 and is used in RewriteF5? to detect useless critical pairs.

Remark 6.1.2. Note that in the description of F5 in [63] Line 5 does not appear. As we have already pointed out in Section 4.2 this is an quite obvious improvement of the algorithm.

Potentially the most known feature of F5 is the fact that it can compute standard bases highly efficient, given some minor restrictions to its input data: Let $F = (f_1, \ldots, f_r)$ be a regular sequence of homogeneous polynomials $f_i \in \mathcal{P}$. Computing a standard basis, to be more precise, a Gröbner basis in this setting, G for $I = \langle f_1 \ldots, f_r \rangle$, F5 does not compute any reduction to zero at all.

This property of F5 is based on the so–called F5 Criterion, for which the notion of *normalized* elements is essential:

Definition 6.1.3 (F5 Criterion)**.**

(1) A labeled polynomial f with $\operatorname{sig}(f) = t e_k$, $t \in \mathcal{P}$, and $\operatorname{poly}(f) \in I$ is called *normalized (w.r.t.* $\langle f_1, \ldots, f_{k-1} \rangle$*)*, if
$$t \notin L\big(\langle f_1, \ldots, f_{k-1} \rangle\big).$$

(2) A critical pair (uf, vg) is called *normalized* if

a) $u \operatorname{sig}(f) \neq v \operatorname{sig}(g)$[1], and

b) uf and vg are normalized.

(3) We say that a critical pair *is detected by the* F5 *Criterion* iff it is not normalized.

Remark 6.1.4. Note that in IncF5 $\operatorname{slt}(g_i)$ needs not be 1 if $i < s$ as it used to be in IncSig resp. IncSigCrit. F5 reuses the signatures computed in previous iteration steps. This is due to the more aggressive implementation of (RW) in F5, which makes use of those "old" signatures.

[1]In [63] $u \operatorname{sig}(f) > v \operatorname{sig}(g)$ is claimed. This is due to the fact that there the first entry of the critical pair is assumed to be the element giving the corresponding s–vector its signature. In our setting we do not require this for critical pairs, thus inequality is enough to ask for.

Let us have a closer look at IncF5, the incremental part of F5 presented in Algorithm 50. Doing this we lay a focus on how the F5 Criterion and the Rewritten Criterion are used to detect useless critical pairs.

It seems quite clear that normalized critical pairs are in a strong connection to the non–minimality property of signatures we discovered in Section 5.1. In particular, the F5 Criterion is much stronger than SigStd's implementation of (NM) due to the fact that in F5 also the second generator of the critical pair is checked. This means that we cannot implement the detection of non–normalized critical pairs just by using NonMin? (Algorithm 37), but need a special implementation presented in the pseudo code of Algorithm 47:

Algorithm 47 F5's implementation of (NM) (NonMinF5?)

Input: uf a labeled polynomial multiplied by a term, S a finite list of finite lists of terms in $\mathcal{P}$
Output: TRUE if $u\,\mathrm{sig}(f)$ is detected by the F5 Criterion, FALSE otherwise

$l \leftarrow \mathrm{index}(f)$
$t \leftarrow u\,\mathrm{slm}(f)$
for $(i = 1, \ldots, l-1)$ **do**
  $m \leftarrow$ length $(S[i])$
  for $(j = 1, \ldots, m)$ **do**
    if $\big(S[i][j] \mid t\big)$ **then**
      return TRUE
return FALSE

The main idea is to implement S in F5 not as a set, but as a list. Due to the fact that both generators of a critical pair are checked we must ensure that we do not discard an element of index 3 by a term in S whose corresponding labeled polynomial has index 4 during a computation of IncF5 of actual index 11. This is ensured by keeping all terms corresponding to a given index j in a list $S[j]$ and storing all those lists in one big list S. This is done in Line 1 of IncF5.

Thus by ensuring the correct initialization of S in Line 4 we can use NonMinF5? in Lines 9 and 29 when new critical pairs are generated. Note that as contrast from IncSigCrit we do not need to use NonMinF5? in Line 15, too, when entering the critical pair to the reduction process. This is owing to the fact that S is not updated when a zero reduction takes place in the presented, original version of F5.

Whenever a new element is added to G in IncF5, we add its leading term to $S[i]$ (Line 23) as it will be useful in the next iteration round for detecting useless critical pairs generated by elements of index $i+1$. This is done in Algorithm 48.

Algorithm 48 F5's F5 Criterion adding algorithm (addF5Crit)

Input: t a term, S a list of terms in $\mathcal{P}$
Output: S a list of terms in $\mathcal{P}$

append $\big(t, S\big)$
return S

IncF5 seems to be quite similar to IncSigCrit, but differs in some substantial points:

(1) Due to the fact that F5 reuses already computed signatures from previous iteration steps, there is no need to initialize all elements of G as it is done in IncSigCrit (Lines 5–7). There is only one initialization, namely those of g_s in Line 3, all other elements are just copied from G_{i-1}.

(2) In Line 12 IncF5 preselects a bunch of critical pairs out of the pair set P. These elements are determined by their degree: P' is the set of critical pairs of P, which have the minimal possible degree. It follows that the computation process does not only loop over elements of P, but goes through an inner loop over the elements in P'. In P' the order in which the elements are sorted to proceed with the reduction steps is the same as in IncSigCrit: At each start of a new reduction process we choose the critical pair of P' for which the signature of the corresponding s–vector is minimal w.r.t. P' (if there are several pairs of the same signature, take the one which was added to P first).

(3) This storage of elements of smallest possible degree, say d, in P' has some consequences for IncF5: In Line 30 any new critical pair (ur, vg_k) is added to P, but not to P'. Thus it must be ensured that the degree of (ur, vg_k) is greater than d. This is caused by the test that $\mathrm{lm}(g_k) \nmid \mathrm{lm}(r)$ in Line 25, and the fact that we are restricting the input to homogeneous data only.

(4) From (3) it follows that it is, in contrast to IncSigCrit, not possible to introduce sig–unsafe reduction steps disguised as critical pairs after a new element is added to G. The problem is that some of the sig–unsafe reductions might be essential for the correctness of F5's computations, so we must ensure that these are computed nevertheless. In F5 this is handled by SigRedF5.

(5) Moreover, in Line 20 of IncF5 the polynomial part of r is reduced w.r.t. $\mathrm{poly}(G_{i-1})$. Note that this "labelless" reduction does not pose a problem for the sig–safeness at all! Since we are using $<_\mathrm{i}$ on the signatures all elements in G_{i-1} have a lower index, and thus all corresponding signatures are ensured to be smaller than those of r.

The last three points give us an idea of the capabilities SigRedF5 must offer: Besides performing only sig–safe reductions with elements of current index (which is not the complete truth as we explain below) it needs to generate new critical pairs representing sig–unsafe reductions and enters them to P'. In addition another crucial difference to SigRed can be found in Algorithm 49: The testing of (NM) and (RW) on the reducers.

Let us discuss the above mentioned essential points in detail:

(1) In Lines 1 and 2 the sets D and B_{i-1} are constructed: D is the set of all labeled polynomials of current index, which are possible reducers of f and must be checked for sig–safeness before performing a reduction with them (Line 11). B_{i-1}, on the other hand, is a polynomial set, consisting of the polynomial parts of labeled polynomials in G, whose index is smaller than the current one. This set is used in Line 10. The idea behind this is the following:

Algorithm 49 F5's semi–complete sig–safe reduction algorithm (SIGREDF5)

Input: f a labeled polynomial, $G = \{g_1, \ldots, g_t\}$ a finite set of labeled polynomials, S a list of lists of terms in $\mathcal{P}$, R a list of lists of terms in $\mathcal{P}$, s the index of the first labeled polynomial of current index, P' a set of critical pairs

Output: h a labeled polynomial sig–safe reduced w.r.t. G, P' a set of critical pairs

1: $D \leftarrow \{g_s, \ldots, g_t\}$
2: $B_{i-1} \leftarrow \{\mathrm{poly}(g_1), \ldots, \mathrm{poly}(g_{s-1})\}$
3: $l \leftarrow \mathrm{siglm}(f)$
4: $p \leftarrow \mathrm{poly}(f)$
5: **while** $\left(p \neq 0 \text{ and } D_p \leftarrow \left\{g \in D \mid \mathrm{lm}\left(\mathrm{poly}(g)\right) \mid \mathrm{lm}(p)\right\} \neq \emptyset\right)$ **do**
6: Choose any $g \in D_p$.
7: $u \leftarrow \frac{\mathrm{lt}(p)}{\mathrm{lt}(\mathrm{poly}(g))}$
8: **if** $(!\,$NONMINF5?(ug, S) and $!\,$REWRITEF5?$(u, g, R))$ **then**
9: **if** $\left(\mathrm{lm}(u)\,\mathrm{siglm}(g) < l\right)$ **then**
10: $q \leftarrow$ REDUCE $\left(u\,\mathrm{poly}(g), B_{i-1}\right)$
11: $p \leftarrow p - q$
12: **else if** $\left(\mathrm{lm}(u)\,\mathrm{siglm}(g) > l\right)$ **then**
13: $P' \leftarrow P' \cup \{(ug, (\mathrm{sig}(f), p))\}$
14: $h \leftarrow \left(\mathrm{sig}(f), p\right)$
15: **return** (h, P')

- ▷ One the one hand $\mathrm{poly}(f)$ is completely reduced w.r.t. B_{i-1} before it enters SIGREDF5 (see Line 20 of INCF5). Thus it is enough to reduce f with labeled polynomials of the current index, i.e. we only need to search in D for possible reducers, not in the whole G.
- ▷ On the other hand, when we reduce $\mathrm{poly}(f)$ with some multiple u of some $\mathrm{poly}(g)$, $g \in D$, it is possible to introduce terms in $\mathrm{poly}(f) - u\,\mathrm{poly}(g)$ which can be reduced w.r.t. B_{i-1}. By construction these terms come from $u\,\mathrm{poly}(g)$. Thus reducing $u\,\mathrm{poly}(g)$ w.r.t. B_{i-1} to an element q before reducing $\mathrm{poly}(f)$ with it ensures that $\mathrm{poly}(f) - q$ is still completely reduced w.r.t. B_{i-1}.
- ▷ Since $\mathrm{poly}(f)$ is completely reduced w.r.t. B_{i-1} and $\mathrm{lm}(f) = u\,\mathrm{lm}(g)$, it is not possible that $\mathrm{lm}(q) < u\,\mathrm{lm}(g)$. It follows that

$$\mathrm{lm}\left(\mathrm{poly}\left(f\right) - q\right) < \mathrm{lm}\left(\mathrm{poly}\left(f\right)\right).$$

(2) The second crucial change of SIGREDF5 compared to SIGRED is that whenever a sig–unsafe reduction $f - ug$ would happen a new critical pair (ug, f) is generated and added to P'. This process is well–defined since $\deg(ug - f) = \deg(f)$ assuming all elements to be homogeneous. Thus (ug, f) must be part of the current preselection P', therein sorted by increasing signature. We already know that $\mathrm{sig}(ug) \succ \mathrm{sig}(f)$, thus the reduction of (ug, f) is scheduled after the current reduction of f for sure.

(3) We see that not all possible sig–safe or sig–unsafe reductions take place. In Line 8 SIGREDF5 checks the possible reducer ug for minimality of its signature (NM) as

well as for its non–rewritability (RW). This is something SIGRED does not perform, with tremendous impacts on the performance as we see later on.

Remark 6.1.5.

(1) Note that in the description of the F5 Algorithm in [63] the so–called top–reduction process, which parallels SIGREDF5, does not take care of the reduction of elements w.r.t. G_{i-1}. This is outsourced to another wrapper algorithm in [63]. We want to keep the description of the algorithm as comprehensible as possible, in this sense we have chosen this more unsophisticated presentation.

(2) On the other hand, F5, as presented in [63], keeps recently computed new elements of degree d in a pool R_d until the whole degree step is done, i.e. until $P' = \emptyset$. Afterwards those elements are added to G. We do this right away to conform to our notation introduced in the presentation of SIGSTD since it seems to be more fluent. By all means this difference does not change any computational aspect of F5.

(3) The idea of using NONMINF5? and REWRITEF5? in SIGREDF5 for testing the reducers ug is quite natural. Any such reduction step can be interpreted as a critical pair (f, ug). For such a pair it is self–evident in our context to test both criteria.

(4) Moreover, note that we clearly do not need to recheck f with both criteria, since we have done this already before entering SIGREDF5. g itself, as an element already in G is also already tested, but multiplying g with a term u it is not clear if ug still passes the tests.

(5) Another important fact we should mention is the different treatment of current index labeled polynomials and those of lower index in F5. Whereas those of current index are always checked by the criteria before reducing with them, F5 does not check the reducers of lower index. Moreover, whereas F5 performs complete reductions w.r.t. the elements of previous iteration steps, it only reduces leading terms with the current index ones. In Section 6.2 we see that this handling of the overall reduction process is quite essential improving F5.

What is left is a discussion of how F5 resp. INCF5 implements (RW) in REWRITEF5?. For this we need to give some more background on rewritability. As we have already mentioned in Chapter 5 F5 implements (RW) way more aggressive than all other signature–based algorithms. It checks, as AP, not only for equality (i.e. checking if another critical pair of the same signature exists), but for divisibility (i.e. an element whose signature divides the one of the critical pair in question). We see that this Rewritten Criterion influences two main parts of F5:

▷ On the one hand, it improves its performance quite a lot detecting more useless critical pairs than all variants of SIGSTD.

▷ On the other hand, it seems to be "too aggressive" in an algorithmic sense: Until now there is no full proof of F5's termination. Although we present some variants of F5 that ensure termination with nearly no overhead in Section 6.5, showing termination for the original version of F5 is still an open problem.

Algorithm 50 Incremental F5 step (IncF5)

Input: f_i a polynomial, $G_{i-1} = \{g_1, \ldots, g_{s-1}\}$ a set of labeled polynomials such that $\operatorname{poly}(G_{i-1})$ is a standard basis for $\langle f_1, \ldots, f_{i-1} \rangle$, S a list of $(i-1)$ lists of terms in $\mathcal{P}$, R a list of $(i-1)$ lists of terms in $\mathcal{P}$

Output: G a set of labeled polynomials such that $\operatorname{poly}(G)$ is a standard basis for $\langle f_1, \ldots, f_i \rangle$, S a list of i lists of terms in $\mathcal{P}$, R a list of i lists of terms in $\mathcal{P}$

$B \leftarrow \varnothing, G \leftarrow \varnothing, P \leftarrow \varnothing, P' \leftarrow \varnothing, R[i] \leftarrow$ empty list, $S[i] \leftarrow$ empty list
$t \leftarrow s$
$g_s \leftarrow (e_i, f_i)$
$S[i] \leftarrow \text{addF5Crit}\left(\operatorname{lt}(g_s), S[i]\right)$
$G \leftarrow \{g_1, \ldots, g_s\}$
for $(k = 1, \ldots, s-1)$ **do**
  $u \leftarrow \operatorname{lc}(g_k)\frac{\tau(g_s,g_k)}{\operatorname{lm}(g_s)}$
  $v \leftarrow \operatorname{lc}(g_s)\frac{\tau(g_s,g_k)}{\operatorname{lm}(g_k)}$
  if $\left(!\,\text{NonMinF5?}\,(ug_s, S) \text{ and } !\,\text{NonMinF5?}\,(vg_k, S)\right)$ **then**
    $P \leftarrow P \cup \{(ug_s, vg_k)\}$
while $(P \neq \varnothing)$ **do**
  $P' \leftarrow \text{Select}(P)$ (critical pairs of minimal degree)
  while $(P' \neq \varnothing)$ **do**
    Choose (uf, vg) from P' with $\max_{\prec}\{u \operatorname{sig}(f), v \operatorname{sig}(g)\}$ minimal w.r.t. $\prec$.
    if $\left(!\,\text{RewriteF5?}(u, f, R) \text{ and } !\,\text{RewriteF5?}(v, g, R)\right)$ **then**
      $P' \leftarrow P' \setminus \{(uf, vg)\}$
      $l \leftarrow \max_{\prec}\{u \operatorname{sig}(f), v \operatorname{sig}(g)\}$
      $R[i] \leftarrow \text{addRule}(l, R[i])$
      $r \leftarrow \left(l, u \operatorname{poly}(f) - v \operatorname{poly}(g)\right)$
      $\operatorname{poly}(r) \leftarrow \text{Reduce}\left(\operatorname{poly}(r), \operatorname{poly}(G_{i-1})\right)$
      $(r, P') \leftarrow \text{SigRedF5}(r, G, S, R, s, P')$
      if $\left(\operatorname{poly}(r) \neq 0 \text{ and } r \text{ not sig-redundant w.r.t. } G\right)$ **then**
        $S[i] \leftarrow \text{addF5Crit}(\operatorname{lt}(r), S[i])$
        for $(k = 1, \ldots, t)$ **do**
          if $(\operatorname{lm}(g_k) \nmid \operatorname{lm}(r))$ **then**
            $u \leftarrow \operatorname{lc}(g_k)\frac{\tau(r,g_k)}{\operatorname{lm}(r)}$
            $v \leftarrow \operatorname{lc}(r)\frac{\tau(r,g_k)}{\operatorname{lm}(g_k)}$
            if $\left(\operatorname{lm}(u) \operatorname{siglm}(r) \neq \operatorname{lm}(v) \operatorname{siglm}(g_k)\right)$ **then**
              if $\left(!\,\text{NonMinF5?}\,(ur, S) \text{ and } !\,\text{NonMinF5?}\,(vg_k, S)\right)$ **then**
                $P \leftarrow P \cup \{(ur, vg_k)\}$
        $t \leftarrow t + 1$
        $g_t \leftarrow r$
        $G \leftarrow G \cup \{g_t\}$
return (G, S, R)

Next we give a definition of how (RW) is implemented in F5 using so-called *rules*, a

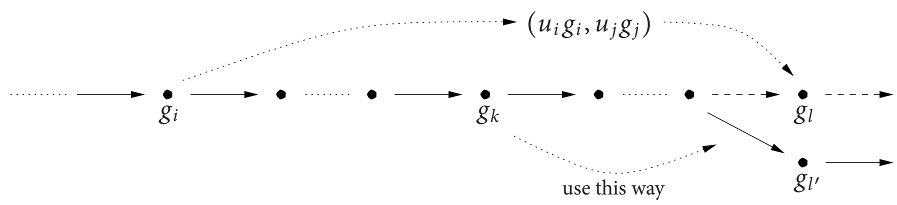

Figure 6.1.1: Illustration of the Rewritten Criterion

data structure which collects all already known signatures in lists.

Definition 6.1.6 (Rewritten Criterion). Let $I = \langle f_1, \ldots, f_r \rangle$, let g_i, g_j be two labeled polynomials in G, computed in by IncF5, and let u_i, u_j be two terms in $\mathcal{P}$.

(1) A *rule* r is $\text{slm}(u_i g_i - u_j g_j)$ for an s–vector $u_i g_i - u_j g_j$ considered in IncF5.

(2) The *rules list* $R[m]$ *corresponding to some index* m *w.r.t.* G is a list $R[m] = (r_1, \ldots, r_k)$ of rules r_i which are signature monomials of elements g considered in IncF5 such that $\text{index}(g) = m$ and $r_i > r_{i-1}$ for all $i \in \{2, \ldots, k\}$. Moreover we define the complete list of rules by

$$R = (R[1], R[2], \ldots, R[r-1], R[r]).$$

(3) We say that a critical pair $(u_i g_i, u_j g_j)$ *is detected by the Rewritten Criterion* if the following holds: There exists $k \in \{i, j\}$ with $l = \text{index}(g_k)$ such that there exists $t \in R[l]$ with

$$t > \text{slm}(g_k), \text{ and}$$
$$t \mid u_k \, \text{slm}(g_k).$$

The basic idea behind the Rewritten Criterion can be illustrated as in Figure 6.1.1: Thinking of G as a list of labeled polynomials appended to its end whenever a reduction process stops with a nonzero remainder, we see that after having added g_i and g_k to G we are at the point where the critical pair $(u_i g_i, u_j g_j)$ should lead to a new element g_l. Now we can assume that the pair is detected by the Rewritten Criterion, in particular, let us say that the rule corresponding to g_k rewrites $u_i g_i$. This means that there are elements being generated after g_i which can contribute to get an element g_m (during the actual degree step, i.e. before IncF5 jumps back to Line 12) with the same signature as g_l, but generated of another critical pair. So our way of constructing G changes a bit, getting a possible different element $g_{l'}$ instead of g_l, but hopefully performing less reduction steps in the following due to using elements, which entered G after g_i had done so.

We have already seen that F5 takes care of R, using it as parameter for IncF5. The main construction and usage of R takes place there and is quite similar to the usage of S in F5:

In Line 18 the algorithm addRule is called, which appends the signature monomial of l to the rules list $R[i]$. The important point is to add the rule exactly at this point of the

computation: The critical pair (uf, vg) has already passed both criteria checks and the corresponding s–vector is constructed out of it. This element is investigated by the algorithm and further reduced. Even if $uf - vg$ reduces to zero in the end its signature monomial should be added to $R[i]$ and can thus be used to reject other useless critical pairs with the Rewritten Criterion. Moreover, due to Line 14 the rules appended to $R[i]$ are always greater than the ones already in the list, thus an increasing list of rules is constructed in this way.
The steps of addRule are presented in the pseudo code of Algorithm 51 and should be clear without any further explanation.

Algorithm 51 F5's rule adding algorithm (addRule)

Input: l a signature, R a list of terms in $\mathcal{P}$
Output: R a list of terms in $\mathcal{P}$

```
append(lm(l), R)
return R
```

As a last step we discuss RewriteF5?, which implements the Rewritten Criterion. In detail, this algorithm is called once in IncF5, namely in Line 15. This is the optimal choice, since at this point all possible rules which could be useful to detect the critical pair are stored in R already. RewriteF5? takes the two generators of the critical pair (uf, vg) and the complete rules list R. It checks Rewritten Criterion for uf and vg separately, by

- ▷ extracting the index of the labeled polynomial (Line 1), and
- ▷ looping over all rules in the respective lists until the rule coming from the corresponding labeled polynomial is reached (Line 4).

Whenever a divisibility check is fulfilled the algorithm returns TRUE, the critical pair is detected by the Rewritten Criterion. Otherwise, the critical pair seems to be useful and its computation in IncF5 goes on.

Algorithm 52 F5's implementation of the Rewritten Criterion (RewriteF5?)

Input: u a term, f a labeled polynomial, R a list of lists of terms in $\mathcal{P}$
Output: TRUE if uf is detected by the Rewritten Criterion, FALSE otherwise

```
1: k ← index(f)
2: t ← u slm(f)
3: m ← length(R[k])
4: while (slm(f) < R[k][m]) do
5:     if (R[k][m] | t) then
6:         return TRUE
7:     m ← m − 1
8: return FALSE
```

Remark 6.1.7.

(1) Note that it is crucial that $R[l]$ are lists, not sets. The order of the list is essential for the correctness of the Rewritten Criterion. We have already seen that REWRITEF5? appends a new rule to the list whenever a new s–vector is prepared to be reduced. This order is not allowed to change.

(2) In the kth call of INCF5 only the lists $S[l]$ for $1 \leq l < k$ are used detecting useless critical pairs by the F5 Criterion. The list $S[k]$ is only initialized and new elements are added to it. It is first used in the next iteration step.

(3) In the definition of the Rewritten Criterion we see for the first time in this thesis why F5 wants to keep the signatures of the previous iteration steps: Looking at a critical pair, both generators are checked by the criterion. Thus also a generating labeled polynomial of lower index, i.e. from the previous iteration round could be detected to be rewritable and help to reject useless critical pairs.

(4) Comparing Definition 6.1.6 (3) to Lemma 5.1.2 one obviously sees that the later one is a very special situation of the Rewritten Criterion for F5: Firstly, only one generator is tested in REWRITE? and its derivatives, namely the one giving the signature for the corresponding s–vector. Secondly, another critical pair of the same signature must exist. In this situation the Rewritten Criterion clearly holds, too.

(5) Note that REWRITEF5? can be implemented in parallel, just like NONMINF5?, checking both generators uf and vg separately at the same time, since the computations are independent from each other. Clearly this cannot be done on the level of different processes, but must be implemented on the level of threads. One task which is not straightforward in this setting is how one of the rewrite algorithms can tell the other one that a useless critical pair is found without consuming too much computational time.

(6) In an optimized implementation one would compute the coresponding indices of f and g beforehand and pass the corresponding lists $R[\operatorname{index}(f)]$ and $R[\operatorname{index}(g)]$ to REWRITEF5? only.

We need to prove that F5 computes a correct standard basis for any input. This we do in several steps, preparing the main theorem 6.1.13.

Lemma 6.1.8. *Let uf be a multiple u of a labeled polynomial $f \in G$ in INCF5. Assume that uf is detected by the Rewritten Criterion. In particular, there exists a rule $r \in R[\operatorname{index}(f)]$ such that $r \mid u \operatorname{slm}(f)$ and $r > \operatorname{slm}(f)$. If P' becomes the empty set, then there exist terms $\delta_j \in \mathcal{P}$, and $g_j \in G$ such that*

$$u \operatorname{poly}(f) = t \operatorname{poly}(h) + \sum_{g_j \in G, g_j \neq h} \delta_j \operatorname{poly}(g_j)$$

such that

(1) $h \in G$ or $\operatorname{poly}(h) = 0$,

(2) for all g_j with $t_j \neq 0$ $t_j \operatorname{siglm}(g_j) \prec u \operatorname{siglm}(f)$, and

(3) $\mathrm{lm}(u)\,\mathrm{siglm}(f) = \mathrm{lm}(t)\,\mathrm{siglm}(h)$.

Proof. Assume that $P' = \varnothing$, after we have considered uf in some critical pair, i.e. the current degree step in F5 has just finished. Since $u\,\mathrm{poly}(f) \in I = \langle f_1, \ldots, f_r \rangle$ we can write

$$u\,\mathrm{poly}(f) = \sum_{i=1}^{k} \lambda_i f_i$$

such that $\mathrm{index}(f) = k \leq r$, $\lambda_k = u\,\mathrm{slt}(f)$. Moreover, let us assume s to be the corresponding s–vector of the labeled polynomial h with $\mathrm{siglm}(h) \mid u\,\mathrm{siglm}(f)$. Since we compute by increasing signatures and $P' = \varnothing$, we can assume that either $\mathrm{poly}(h) = 0$ or $h \in G$ already. In any case the rule $\mathrm{siglm}(h) \in R[k]$ has detected uf to be rewritable. As h is constructed out of s by sig–safe reduction steps we can assume that

$$\mathrm{poly}(s) = \sum_{i=1}^{k} \sigma_i f_i$$

where $\sigma_k = \mathrm{slt}(s) = \mathrm{slt}(h)$. Since we have already computed all elements of P' there exist $\eta_j \in \mathcal{P}$ such that

$$\mathrm{poly}(s) = \mathrm{poly}(h) + \sum_{g_j \in G, g_j \neq h} \eta_j\,\mathrm{poly}(g_j) \text{ with}$$
$$\mathrm{sig}(h) \succ \max_{\prec} \left\{ \mathrm{lt}(\eta_j)\,\mathrm{sig}(g_j) \mid g_j \in G, g_j \neq h \right\}.$$

Let $t \in \mathcal{P}$ be a term such that $t\,\mathrm{sig}(h) = u\,\mathrm{sig}(f)$. Then we can represent $u\,\mathrm{poly}(f)$ in the following way using the two different representations of $\mathrm{poly}(s)$ mentioned before:

$$\begin{aligned} u\,\mathrm{poly}(f) &= \sum_{i=1}^{k} \lambda_i f_i + t\,\mathrm{poly}(s) - t\,\mathrm{poly}(s) \\ &= \sum_{i=1}^{k} (\lambda_i - t\sigma_i)\, f_i - t\,\mathrm{poly}(h) + \sum_{g_j \in G, g_j \neq h} \eta_j\,\mathrm{poly}(g_j) \\ &= t\,\mathrm{poly}(h) + \sum_{g_j \in G, g_j \neq h} \delta_j\,\mathrm{poly}(g_j) \end{aligned}$$

where

$$\delta_j = \begin{cases} \lambda_j - t\left(\sigma_j - \eta_j\right) & \text{if } \mathrm{poly}(g_j) = f_i \text{ for some } i \in \{1, \ldots, k\}, \\ t\eta_j & \text{otherwise.} \end{cases}$$

Since $(e_i, f_i) \in G$ for all $i \in \{1, \ldots, k\}$ at this point and due to the fact that $\mathrm{sig}(h) \succ \max_{\prec} \left\{ \mathrm{lt}(\eta_j)\,\mathrm{sig}(g_j) \mid g_j \in G, g_j \neq h \right\}$, the statement follows. □

Remark 6.1.9.

(1) Note that it is not a problem if $\mathrm{poly}(h) = 0$ and h is not added to G at all. Then we still have a signature $\neq 0$ of the zero polynomial and the restriction that $g_j \neq h$ for $g_j \in G$ in the representation of uf is trivial.

(2) Note that Lemma 6.1.8 does not claim that the found representation of $u \operatorname{poly}(f)$ is a standard representation w.r.t. $\operatorname{poly}(G)$. The only usefulness lies in the fact that the corresponding labeled polynomials of the elements in the representation of $u \operatorname{poly}(f)$ have a smaller signature than uf, besides possibly th. What seems to be a completely useless statement in a usual standard basis computation is of greatest importance in the signature–based world as we see in the proof of the main theorem of this section, Theorem 6.1.13.

Lemma 6.1.10. *Let g_i and g_j be two labeled polynomials in G computed by* F5 *such that* $\operatorname{index}(g_i) = \operatorname{index}(g_j) = k$ *and $i < j$. If there exist terms $u, v \in \mathcal{P}$ such that* $u \operatorname{sig}(g_i) = v \operatorname{sig}(g_j)$*, then ug_i (and thus any critical pair it is generating) is detected by the Rewritten Criterion*

Proof. Clearly, g_i was considered before the s–vector s which leads to g_j, thus the corresponding rule $\operatorname{slm}(g_j)$ is in $R[k]$ and it also checks to rewrite any multiple of g_i in the following. Since ug_i is considered and $u \operatorname{sig}(g_i) = v \operatorname{sig}(g_j)$ there exists a monomial $m \in \operatorname{Mon}(x_1, \ldots, x_n)$ such that $m \operatorname{slm}(g_j) = \operatorname{lm}(u) \operatorname{slm}(g_i)$. In particular, $m = \operatorname{lm}(v)$. □

Also Lemma 6.1.10 seems quite clear, neither Faugère ([63]) nor Stegers ([144]) mention that the signature monomials of the corresponding generators of critical pairs must be different. Otherwise one would get sig–equivalent critical pairs, which need not be considered as we have shown in Corollary 4.1.18. The point of Lemma 6.1.10 is that even though they do not check for sig–equivalence at all they discard those critical pairs by the Rewritten Criterion. Nevertheless this can have a bad influence on the performance of the algorithm since the Rewritten Criterion is checked much later than the sig–equivalence check is done in IncSig, this means more data must be stored and carried. Thus in our presentation of IncF5 in Algorithm 50 we have kept the sig–equivalence check in Line 28 from IncSig due to optimization reasons.

Lemma 6.1.11. *Let uf be a multiple u of a labeled polynomial $f \in G$ considered in IncF5 where* $\operatorname{index}(f) = k$*. Assume that uf is detected by the* F5 *Criterion. Then there exists a principal syzygy $s \in \mathcal{P}^k$ such that* $\operatorname{lt}(s) \mid u \operatorname{sig}(f)$.

Proof. We have shown this already in the proof of Lemma 5.1.1. □

Corollary 6.1.12. *Let uf be a multiple u of a labeled polynomial $f \in G$ in IncF5 where* $\operatorname{index}(f) = k$*. Assume that uf is detected by the* F5 *Criterion. Then there exists a syzygy $s \in \mathcal{P}^k$ such that*

$$\operatorname{lt}\left(u \operatorname{label}(f) - s\right) = \operatorname{sig}\left(u \operatorname{poly}(f)\right) < u \operatorname{sig}(f).$$

Proof. As long as uf is detected by the F5 Criterion rewrite uf by $uf - vg$ where

$$v \operatorname{lt}(s) = u \operatorname{sig}(f), \text{ and}$$
$$g = \big(s, \pi(s)\big),$$

with s being the principal syzygy from Lemma 6.1.11 and $\pi : \mathcal{P}^k \to \mathcal{P}$, $e_i \mapsto f_i$ for $i \in \{1, \ldots, k\}$, $k \leq r$. Then $u \operatorname{poly}(f) = u \operatorname{poly}(f) - v \operatorname{poly}(g)$, since $\operatorname{poly}(g) = \pi(s) = 0$. Due to $<$ being a well–order this process of rewriting uf terminates at some point. □

Now we are ready to prove the main theorem of this section.

Theorem 6.1.13. *Let $F = \{f_1, \ldots, f_r\}$, a finite set of homogeneous polynomials in $\mathcal{P}$ equipped with a well-order $<$, be the input of* F5. *If the kth iteration of* IncF5 *terminates with output (G, R), then $\mathrm{poly}(G)$ is a standard basis for $\langle f_1, \ldots, f_k \rangle$ w.r.t. $<$.*

Proof. Looking at the elements in G three types of labeled s–vectors can occur:

(1) $uf - vg$ has a standard representation w.r.t. G.

(2) $uf - vg$ is detected by the F5 Criterion.

(3) $uf - vg$ is detected by the Rewritten Criterion.

We need to show that any corresponding polyomial s–vector of two elements $p, q \in \mathrm{poly}(G)$ has a standard representation w.r.t. $\mathrm{poly}(G)$.

We have already seen in Section 4.1 that having a standard representation w.r.t. G implies having a standard representation w.r.t. $\mathrm{poly}(G)$. Thus it is left to show that an s–vector of one of the other two types has a standard representation w.r.t. $\mathrm{poly}(G)$, too.

Let $\mathcal{S}$ be the set of all s–vectors of labeled polynomials of G. Choose $uf - vg$ out of $\mathcal{S}$ to be the element of maximal signature. Note that there might be a choice of such s–vectors. For any such elements f, g, and $h \in G$ choose the s–vector $uf - vg$ such that $u\,\mathrm{lm}(f) = v\,\mathrm{lm}(g) = w\,\mathrm{lm}(h)$ and $u\,\mathrm{siglm}(f) > v\,\mathrm{siglm}(g) > w\,\mathrm{siglm}(h)$. Thus $uf - vg$ is uniquely determined and we can assume that $\mathrm{sig}(uf - vg) = u\,\mathrm{sig}(f)$.

Two situations are possible:

(1) While there exists a component of $uf - vg$, which is detected by the F5 Criterion, we can construct a syzygy s as in Corollary 6.1.12 such that we can rewrite $uf - vg$ using $h = \big(s, \pi(s)\big)$ such that

$$u\,\mathrm{poly}(f) - v\,\mathrm{poly}(g) - w\,\mathrm{poly}(h) = u\,\mathrm{poly}(f) - v\,\mathrm{poly}(g), \text{ and}$$

$\mathrm{sig}\,(uf - vg - wh) = \mathrm{sig}\,(u\,\mathrm{poly}(f) - v\,\mathrm{poly}(g))$. So in the end we can assume that $uf - vg - wh$ has minimal signature.

(2) While there exists a component of $uf - vg$, which is detected by the Rewritten Criterion, we can rewrite it as shown in Lemma 6.1.8.

We do this until there does no longer exist intermediate s–vectors in the representation of $uf - vg$. At this point, we receive a standard representation of $uf - vg$ w.r.t. G due to the fact that all rewritings cancel out some multiple leading terms and do not introduce higher signatures. Thus it is left to show that this iterative process of rewriting terminates after finitely many steps.

Let us have a closer look at what can happen during the two types of rewriting:

(1) In the rewriting triggered by the F5 Criterion we have seen in Corollary 6.1.12 that the considered component of $uf - vg$ is rewritten with a lower signature.

(2) If the rewriting is induced by the Rewritten Criterion we have seen in Lemma 6.1.8 that all but one of the introduced elements have smaller signature. The only element which can have the same signature is th.
   - ▷ If $\operatorname{poly}(h) = 0$, then we have rewritten the component of $uf - vg$ completely with lower signature elements.
   - ▷ Otherwise $h \in G$, but h was added to G after the component it rewrites by definition of the Rewritten Criterion. We choose h to be the element added to G latest rewriting the component. Thus we can assume that th itself is not detected by the Rewritten Criterion.

In each of this rewritten representations of $uf - vg$ let λ be the larger signature and σ be the smaller one.

In most rewritings λ does not increase. There is one exception:[2] Assume that the component corresponding to λ has already been rewritten and λ has been decreased in this situation to λ'. Now, if a rewriting for the component corresponding to σ is applied it is possible that the leading term of the component corresponding to λ' (introduced during the beforehand rewriting) cancels out a non–leading term of the actual rewriting. In this situation it is possible that λ' increases back to λ.

(1) If this rewriting is based on the F5 Criterion, the value of σ must decrease. As $<$ is a well–order, σ can decrease only finitely many times.

(2) If it is evoked by the Rewritten Criterion, the rewriter was added to G after the element it is rewriting. Since we assume that IncF5 has terminated, G has only finitely many elements. Thus also this process must terminate after finitely many steps.

Hence λ can increase to any previous already taken value only finitely many times.

This means that each iteration of the rewriting process either decreases one of λ or σ, or gives us an element with signature λ resp. σ, which was added to G at a later point of the computations. Since we choose the rewriter element to be the one added to G latest the Rewritten Criterion can invoke a rewriting of a given value of λ or σ at most once. Since G is finite, this process of finding elements with signature λ resp. σ, which are added to G at a later point during the computations has to terminate eventually. In other words, λ must decrease permanently below any given level at once the element added to G at the latest possible point is found.

As $<$ is a well–order, λ cannot decrease indefinitely. Hence the iteration must terminate with a standard representation of $uf - vg$ w.r.t. G. □

Remark 6.1.14.

(1) In [63] and [144] proofs of the correctness of the F5 Algorithm are given, too. These proofs do not cover the usage of the Rewritten Criterion in F5, but only take the F5 Criterion into account. Proving the correctness including F5's aggressive implementation of (RW) was one of the main problems in the last couple of years and was first achieved in [55] resp. [58]. Later on, different variants of the proof of Theorem 6.1.13 and / or new proofs have been published, see, for example, [78, 148].

[2] Thanks to Vasily Galkin for pointing out this exception.

(2) Note that we must assume termination of IncF5 in Theorem 6.1.13. As we see in Section 6.5 the problem proving termination of F5 is quite difficult and not solved until now. At this point let us just point out that all proofs of F5's termination given until now either have some errors or have some not completed gaps. The main reason why proving termination for F5 is way more complicated than proving termination of SigStd or G2V lays in the aggressive implementation of (RW) using the Rewritten Criterion.

As a last step in this introduction to F5 let us show why F5, and also any other signature–based standard basis algorithm, is very efficient when it comes to the computation of bases for input corresponding to regular sequences.

Proposition 6.1.15. *If the polynomials generating the input ideal $I = \langle f_1, \ldots, f_r \rangle \subset \mathcal{P}$ form a regular sequence $F = (f_1, \ldots, f_r)$, then* F5 *does not compute any zero reduction.*

Proof. If F is a regular sequence, then $\operatorname{Syz}(I)$ is generated by the principal syzygies in $\mathcal{P}^r$, w.r.t. $<_{\mathrm{i}}$ in this situation. Assume there exists a zero reduction in F5, say $\operatorname{poly}(r)$ is reduced to zero w.r.t. G for some labeled polynomial r in IncF5. This means that there exist terms $u_i \in \mathcal{P}$ and $g_i \in G = \{g_1, \ldots, g_t\}$ such that

$$\operatorname{poly}(r) = \sum_{i=1}^{t} u_i \operatorname{poly}(g_i), \text{ and}$$
$$\operatorname{siglm}(r) > \max_{<} \{u_i \operatorname{siglm}(g_i) \mid i = 1, \ldots, t\}.$$

So it follows for the corresponding syzygy $s \in \mathcal{P}^r$ that $\operatorname{lt}(s) = \operatorname{sig}(r)$. Since $s \in \operatorname{Syz}(I)$ and as $\operatorname{Syz}(I)$ is generated by principal syzygies it follows that there exists a $g_k \in G$ such that $\operatorname{index}(g_k) < \operatorname{index}(r)$ and $\operatorname{lt}(g_k) \mid \operatorname{slt}(r)$. But this means that r, in its initial form as a critical pair, must have been detected by the F5 Criterion. Thus F5 has not considered r at all. A contradiction to our assumption that F5 has reduced r to zero. ☐

Corollary 6.1.16. *If the polynomials generating the input ideal $I = \langle f_1, \ldots, f_r \rangle \subset \mathcal{P}$ form a regular sequence $F = (f_1, \ldots, f_r)$, then none of the signature–based standard basis algorithms presented in this thesis does compute a zero reduction during the computation of a standard basis G of I.*

Proof. The statement follows from Proposition 6.1.15 together with Lemma 5.1.3. ☐

We finish this section with an example computation for a standard basis for an ideal generated by elements forming a regular sequence.

Example 6.1.17. Let us give an example computation of a standard basis using F5. We use the example given in [63] in Section 8. We have already considered this computation in Example 3.3.5 using SyzStd. This example is very useful in two ways:

- ▷ On the one hand, we can compare the syzygy–based attempt of Möller, Mora, and Traverso (see Section 3.3) with the signature–based one of Faugère.

▷ On the other hand, this example also shows the problem of proving F5's termination (see Section 6.5 for more details).

What we need to do in order to use F5 on Example 3.3.5 is to homogenize the polynomials. Assume that $\mathcal{P}$ is equipped with $<_{\mathrm{dp}}$, and let $F = \{f_1, f_2, f_3\} \subset \mathcal{K}[x, y, z, t]$ where

$$\begin{aligned} f_1 &= x^2 y - z^2 t, \\ f_2 &= xz^2 - y^2 t, \\ f_3 &= yz^3 - x^2 t^2. \end{aligned}$$

Note that we use the slim representation of labeled polynomials in this example. Moreover, note that we use a slightly different numbering of the elements due to our notations. Also note that in our computations coefficients can be part of signatures due to our more general definition. We start our computations setting $G_1 = \{(e_1, f_1)\}$ since f_1 cannot be reduced w.r.t. $\{f_2, f_3\}$, and $S[1] = (\mathrm{lt}(f_1))$. The first real iteration round starts entering IncF5 with f_2 and G_1. There we start by initializing 3 data structures,

$$\begin{aligned} G &= \{ \underbrace{(e_1, f_1)}_{g_1}, \underbrace{(e_2, f_2)}_{g_2} \}, \\ S[2] &= \left(\mathrm{lt}(f_2)\right), \text{ and} \\ P &= \{(xyg_2, z^2 g_1)\}. \end{aligned}$$

In this situation, $P' = P$. Since $(xyg_2, z^2 g_1)$ is detected neither by the F5 nor the Rewritten Criterion we start the computation of the corresponding s–vector:

▷ $r = \left(xye_2, xy \operatorname{poly}(g_2) - z^2 \operatorname{poly}(g_1)\right) = \left(xye_2, -xy^3 t + z^4 t\right)$

▷ The rule xy is appended to the as yet empty list of rules of index 2, $R[2]$.

There exists no reducer in $\operatorname{poly}(G_1)$ nor of current index of r thus we add the new element $g_3 = (xye_2, -xy^3 t + z^4 t)$ to G, $-xy^3 t$ to $S[2]$, and compute new critical pairs:

▷ On the one hand, $(xg_3, y^2 t g_1)$ is not added to P, since $x \operatorname{sig}(g_3) = x^2 y e_2$ and $x^2 y \in S[1]$

▷ On the other hand, $(z^2 g_3, -y^3 t g_2)$ is added to P.

Thus after setting $G = \{g_1, g_2, g_3\}$ the computations go on, again with $P' = P$: Computing $r = \left(xyz^2 e_2, z^2 \operatorname{poly}(g_3) + y^3 t \operatorname{poly}(g_2)\right) = \left(xyz^2 e_2, z^6 t - y^5 t^2\right)$. We add the rule xyz^2 to $R[2]$. There is no possible reduction we can perform, thus

$$\begin{aligned} g_4 &= \left(xyz^2 e_2, z^6 t - y^5 t^2\right), \\ G &= G \cup \{g_4\}, \text{ and} \\ S[2] &= \left(xz^2, -xy^3, z^6 t\right). \end{aligned}$$

The 3 possible critical pairs generated by g_4 are all rejected:

▷ $(x^2 y g_4, z^6 t g_1)$ is detected by the F5 Criterion as $x^2 y \operatorname{sig}(g_4)$ is clearly divisible by $x^2 y$.

▷ $(x g_4, z^4 t g_2)$ is also detected by the F5 Criterion due to $x \operatorname{slm}(g_4) = x^2 y z^2$, which is divisible by $x^2 y$.

▷ $\left(x y^3 g_4, -z^6 g_3\right)$ is also detected by the F5 Criterion since $x y^3 \operatorname{slm}(g_4) = x^2 y^4 z^4$.

Thus IncF5 terminates at this point and returns, besides S and R the set G_2 of labeled polynomials. $\operatorname{poly}(G_2)$ is a standard basis for $\langle f_1, f_2 \rangle$.

The next and final iteration step starts, initializing

$$\begin{aligned} g_5 &= (e_3, f_3)\,, \\ G &= \{g_1, g_2, g_3, g_4, g_5\}\,, \text{ and} \\ S[3] &= \left(\operatorname{lt}(g_5)\right). \end{aligned}$$

Next, the first bunch of critical pairs is generated:

$$P = \left\{(x^2 g_5, z^3 g_1), (x g_5, yz g_2), (x y^2 t g_5, -x z^3 g_3), (z^3 t g_5, y g_4)\right\}.$$

Note that none of the critical pairs is detected by the F5 Criterion. We take those of lowest possible degree, in this situation 5, and move them to P':

$$P' = \left\{(x g_5, yz g_2)\right\}.$$

Since there is no rule added until now, the corresponding s–vector is computed:

$$r = (x e_3, x \operatorname{poly}(g_5) - yz \operatorname{poly}(g_2)) = (x e_3, y^3 z t - x^3 t^2)\,.$$

We add x to $R[3]$ and add the new element to G as there is no possible reducer in G:

$$\begin{aligned} g_6 &= (x e_3, y^3 z t - x^3 t^2)\,, \\ S[3] &= (y z^3, y^3 z t)\,. \end{aligned}$$

Of the 4 possible new critical pairs only 3 are added to P:

$$P = P \cup \left\{(x^2 g_6, y^2 z t g_1), (x z g_6, y^3 t g_2), (x g_6, -z g_3), (z^5 g_6, y^3 g_4)\right\}.$$

$(z^2 g_6, y^2 t g_5)$ is detected by the F5 Criterion using $x z^2$ from $S[2]$. The next bunch of critical pairs of minimal degree 6 is moved to P':

$$P' = \left\{(x^2 g_5, z^3 g_1), (x g_6, -z g_3)\right\}.$$

The corresponding s–vectors have the same signature $x^2 e_3$, so we take $(x^2 g_5, z^3 g_1)$ first, since it was added to P earlier than the other one. $x^2 g_5$ is detected by the Rewritten Criterion using $x \in R[3]$. Thus this pair is deleted and we go on with $(x g_6, -z g_3)$. Although the corresponding s–vector has the same signature $x^2 e_3$ as the former critical pair it is not detected by the Rewritten Criterion. This is due to the fact that we check only for those rules t which fulfill that $t > \mathrm{slm}(g_6)$. Since $\mathrm{slm}(g_6) = x$ and x is the only rule in $R[3]$ until now $xg_6 + zg_3$ is not detected.[3] We go on generating $r = (x^2 e_3, z^5 t - x^4 t^2)$ and add x^2 as rule to $R[3]$. Next we see that there is no reduction w.r.t. $\mathrm{poly}(G_2)$ possible, and also no element of index 3 reduces $\mathrm{lt}(r)$, thus we are done with the degree 6 step and update our data set:

$$\begin{aligned} g_7 &= (x^2 e_3, z^5 t - x^4 t^2)\,, \\ S[3] &= (yz^3, y^3 zt, z^5 t)\,. \end{aligned}$$

Building new critical pairs we see that $(x^y g_7, z^5 tg_1), (xy^3 g_7, -z^5 g_3), (yg_7, z^2 tg_5)$, as well as $(y^3 g_7, z^4 g_6)$ are detected by the F5 Criterion using $x^2 y \in S[1]$.

$$P = P \cup \left\{ (xg_7, z^3 tg_2), (zg_7, g_4) \right\}.$$

The next step takes all pairs of degree 7:

$$P' = \left\{ (x^2 g_6, y^2 ztg_1), (xzg_6, y^3 tg_2), (xg_7, z^3 tg_2), (zg_7, g_4). \right\}$$

Examining the signatures, we go on with $(zg_7, g4)$, which is not detected to be useless: After adding the rule $x^2 z$ to $R[3]$ we compute the 8th element for G by constructing the corresponding s–vector:

$$\begin{aligned} g_8 &= (x^2 ze_3, y^5 t^2 - x^4 zt^2), \\ S[3] &= (yz^3, y^3 zt, z^5 t, y^5 t^2)\,, \\ P &= P \cup \left\{ (xg_8, -y^2 tg_3), (x^2 g_8, y^4 t^2 g_1) \right\}. \end{aligned}$$

Besides these two, above mentioned new critical pairs, all others generated by g_8 are detected by the F5 Criterion.
Using the Rewritten Criterion we can now reject $(xzg_6, y^3 tg_2)$ by $x^2 z \in R[3]$ corresponding to g_8. Next we compute $(xg_7, z^3 tg_2)$, adding the rule x^3 to $R[3]$ and generating $g_9 = (x^3 e_3, -x^5 t^2 + y^2 z^3 t^2)$ we see that a reduction of $\mathrm{poly}(r)$ w.r.t. $\mathrm{poly}(G_2)$ is possible, whereas no further reductions with current index elements take place. We end up with

$$\begin{aligned} g_9 &= (x^3 e_3, -x^5 t^2 + z^2 t^5), \\ S[3] &= (yz^3, y^3 zt, z^5 t, y^5 t^2, -x^5 t^2)\,. \end{aligned}$$

All critical pairs generated by g_9 are detected by the F5 Criterion, thus P is not updated at all.
With the rule coming from g_9 we can reject $(x^2 g_6, y^2 ztg_1)$ and finish this degree step.
For degree 8 we pick

$$P' = \left\{ (xy^2 tg_5, -xz^3 g_3), (z^3 tg_5, yg_4), (xg_8, -y^2 tg_3) \right\}.$$

We start with $(z^3 tg_5, yg_4)$, add the rule $z^3 t$ to $R[3]$ and get the new element

$$g_{10} = (z^3 te_3, y^6 t^2 - xy^2 zt^4).$$

[3] Note that otherwise we would have detected xg_6 by g_6 as the rule x corresponds to g_6. Clearly this cannot be a correct way deleting critical pairs.

Note that the sig–unsafe reduction of g_{10} by g_8 in SIGREDF5 does not take place since $y \operatorname{sig}(g_8) = x^2 y z e_3$ is divisible by $x^2 y \in S[2]$. Thus no new critical pair is generated in SIGREDF5.
At this point all already constructed and all to be generated critical pairs are detected by F5's criteria. Thus INCF5 finishes and returns to F5 with

$$G = \{g_1, \ldots, g_{10}\}.$$

F5 extracts the polynomial part of G and returns

$$\begin{aligned} B \quad = \quad & \{x^2 y - z^2 t, xz^2 - y^2 t, -xy^3 t + z^4 t, z^6 t - y^5 t^2, \\ & yz^3 - x^2 t^2, y^3 z t - x^3 t^2, z^5 t - x^4 t^2, y^5 t^2 - x^4 z t^2, \\ & x^5 t^2 - z^2 t^5, y^6 t^2 - xy^2 z t^4\}, \end{aligned}$$

a standard basis for $I = \langle x^2 y - z^2 t, xz^2 - y^2 t, yz^3 - x^2 t^2 \rangle$.

Remark 6.1.18.

(1) Note that whereas SYZSTD computes 1 zero reduction, F5 does not reduce any s–vector to zero.

(2) Thinking about optimizing F5 while looking at the example computation given above one could get the idea to use the Rewritten Criterion even more aggressively: Why not adding the rules whenever a new critical pair is generated and added to P ? The problem is that such a critical pair, although it passes the F5 Criterion, can still be detected by the Rewritten Criterion later on. Let us look at the degree 6 step of the third iteration round of the above example: Assume that we have added the rule x^2 coming from the signature $x^2 e_3$ of the critical pair $(x^2 g_5, z^3 g_1)$ to $R[3]$. This pair is rewritten by the rule $x \in R[3]$, but now also the next critical pair in P', $(xg_6, -zg_3)$, is detected by the rule x^2 and thus the corresponding s–vector is not computed. We have seen that the data stored in $xg_6 + zg_3$ is essential for the correctness of the computations of F5. We see that, although $(xg_6, -zg_3)$ delivers the very same rule x^2 to $R[3]$ it is crucial that the rule does not correspond to a critical pair which is possibly rejected later on. Thus we need to wait adding rules to R until the critical pair is completely checked and stamped as useful.

Based on this discussion we can start not only comparing F5 to SIGSTD and its variants, but we can also optimize F5 and have a closer look at the problem of proving its termination.

6.2 F5C – F5 USING REDUCED BASES

As a first step on our way improving the initial F5 Algorithm, we want to use a similar attempt as in Section 5.2. The main computational drawback of F5 lies in the overhead produced during each iteration step due to sig–safe reduction steps. This leads to the computation of intermediate standard bases $\text{poly}(G_i)$, neither reduced nor minimal. The problem is that we use these bases in the next iteration step for further computations, which results in two disadvantages:

(1) An overhead of possible reducers of lower index, as well as

(2) an overhead of newly generated critical pairs are generated.

Whereas the idea of interreducing the intermediate standard bases $\text{poly}(G_i)$ in SigStd is straightforward due to the fact that there only the polynomial data of G_i is further used in the $(i+1)$st iteration step of IncSigCrit, it is not so easy to achieve such an result in F5. Note that the main differences between both classes of algorithms (see Section 6.3 for more details) lies in the implementations of (NM) and (RW). In both F5 is way more aggressive, rejecting lots more critical pairs. The main problem interreducing G_i in F5 is that the data stored in the signatures of the elements in G_i become corrupted once we interreduce the basis due to performing sig–unsafe reductions.

Understanding how to do this effectively is the main content of this section. John Perry and the author have presented these ideas first in [58]. It should be mentioned that the idea of interreducing SigStd and its variants, mainly G2V, is just taken from the attempt presented here.

Convention. Explaining the ideas of interreducing previously computed standard bases in F5 we need to talk a lot about different iteration steps. Let us agree for this section on the following notations: We always assume that the current iteration step is the kth one. This means that labeled polynomials computed during this iteration step have index k. Thus G_{k-1} denotes the set of all labeled polynomials computed in the first $(k-1)$ iteration steps and $\text{poly}(G_{k-1})$ is a standard basis for $\langle f_1, \ldots, f_{k-1} \rangle$.

As mentioned above there are two main points one would like to optimize thinking of F5's computations: Less reduction steps and less critical pairs. The first problem is the easier one and is first solved by Till Stegers in his diploma thesis ([144]). It is mainly based on the fact that in F5 we split up the reduction process of an s–vector in two parts:

(1) All reductions of elements of index k, i.e. elements generated during the current iteration step, must be done sig–safe (or must generate new critical pairs, if a sig–unsafe reduction takes place). Here the elements of G_{k-1} have no influence at all, thus this part of F5 is completely independent to any change of G_{k-1}.

(2) Whereas the above mentioned current index reductions are done in SigRedF5, reductions w.r.t. G_{k-1} are performed plainly based on polynomial data. In Line 20 of Algorithm 50 the polynomial part $\text{poly}(r)$ of the current s–vector is reduced w.r.t. $\text{poly}(G_{k-1})$. All these reductions are sig–safe as all reducers have lower index. Moreover it is quite important to note that, in spite of the current index reducers in SigRedF5, the reducers of $\text{poly}(G_{k-1})$ are not checked by the F5 or the Rewritten Criterion. Thus only their polynomial part is used in terms of reduction.

These facts about F5's reduction process have provided Stegers the idea of his variant of F5, denoted F5R. The "R" in F5R stands for *reduction* and means, in short, that F5R uses reduced standard bases for lower index reduction purposes. In [144] he describes his discovery. We give a brief summary of it:

When the $(k-1)$st call of IncF5 returns a new set of labeled polynomials G, $\text{poly}(G)$ is a standard basis for $\langle f_1, \ldots, f_{k-1} \rangle$. As IncF5 needs G, now as G_{k-1} in its kth call for generating new critical pairs the signatures of the elements in G_{k-1} are crucial to the correctness of further computations, as otherwise the F5 Criterion and the Rewritten Criterion do not work properly any longer. Thinking about interreducing $\text{poly}(G_{k-1})$ to a reduced standard basis B_{k-1} one must keep in mind that the reductions taking place there are exactly those which were not allowed in the previous iteration steps of IncF5 due to either the F5 Criterion or the Rewritten Criterion or the sig–unsafeness of the reduction. Not performing these reduction steps keep the signatures as well as the rules list R correct, so interreducing the standard basis at the end corrupts these data and we cannot use them in any upcoming iteration step anymore. Thus, if we want to keep the signatures and rules already computed, we really need to stuck to G_{k-1} when talking about generators of new critical pairs.

On the other hand, we have seen that the reduction with elements of index $< k$ is done plainly on the polynomial side, thus no information about signatures or rules is important. So we can optimize F5 in the following way: Whenever IncF5 returns we take the computed standard basis $\text{poly}(G_{k-1})$ and reduce it to B_{k-1}. Besides passing G_{k-1} to the kth instance of IncF5 we also pass B_{k-1} to it. There we use the elements of G_{k-1} to build new critical pairs. This is possible, because the signatures and rules are correct for these elements. When it comes to a prereduction of a newly computed s–vector in IncF5 we use B_{k-1}. This has the advantage over reducing w.r.t. $\text{poly}(G_{k-1})$ that there are no redundant reducers to be checked and that the reducers are completely reduced. This can lead to a lot less divisibility checks and reduction steps.

The problem of this attempt is that one has only an advantage on the reduction process, and there not even on the complete one, but only on the part of lower index. Moreover, one needs to store B_{k-1} besides G_{k-1}, which consumes more memory. For solving these problems we developed the idea of completely switching from G_{k-1} to B_{k-1}.

F5R is fully integrated in our idea of interreducing intermediate standard bases, a variant we call F5C. The "C" in F5C is derived from F5R and means that we do all *computations* of F5 using reduced bases. So the main obstacle to leap is how to handle the signatures and rules when having interreduced $\text{poly}(G_{k-1})$? We have already explained that those are useless after that step, i.e. there is no longer a connection between the signatures resp. rules from the $(k-1)$st iteration step and the polynomials in B_{k-1}. Thus we need to throw away those data sets and generate new ones, which are appropriate for B_{k-1}. In this sense we need to start again with counting indices: An element $b_i \in B_{k-1}$ gets the label e_i. The whole process keeps our computations correct since going on to the next iteration step k we want to compute a standard basis for $\langle f_1, \ldots, f_k \rangle$. But it clearly holds that

$$\langle f_1, \ldots, f_k \rangle = \langle b_1, \ldots, b_{s-1}, f_k \rangle$$

where $s - 1 = \#(B_{k-1})$. Thus everything that needs to be adjusted in IncF5 is that the new labeled polynomial for f_k does not get label e_k, but label e_s. Looking to IncSigCrit one sees how this fits quite smoothly in the initialization of G there. With this operation we receive correct labels resp. signatures for the elements in B_{k-1}, but we still have lost the information stored in R which are useful for detecting useless critical pairs being partly generated by elements of B_{k-1}. How to recover at least some rules that help us on this task? The idea is simply to loop over all s–vectors of elements of B_{k-1} and store the corresponding signatures in a newly created list of lists of rules R:

Algorithm 53 F5C's interreduction process (ReduceF5)

Input: G a finite set of labeled polynomials, S a list of lists of terms, R a list of lists of terms
Output: G' a finite set of labeled polynomials, S' a list of lists of terms, R' a list of lists of terms

1: $B \leftarrow \mathrm{poly}(G)$
2: Delete G, R and S.
3: $G' \leftarrow \varnothing$, $R' \leftarrow$ empty list, $S' \leftarrow$ empty list
4: $B \leftarrow \mathrm{Reduce}(B)$
5: $t \leftarrow \#(B)$
6: $G' \leftarrow G' \cup \left\{(e_1, b_1)\right\}$
7: $S'[1] \leftarrow \left(\mathrm{lt}(b_1)\right)$
8: $R'[1] \leftarrow$ empty list
9: **for** $\left(i = t, \dots, 2\right)$ **do**
10: $\quad G' \leftarrow G' \cup \left\{(e_i, b_i)\right\}$
11: $\quad S'[i] \leftarrow \left(\mathrm{lt}(b_i)\right)$
12: $\quad$ **for** $\left(j = i-1, \dots, 2\right)$ **do**
13: $\quad\quad \lambda \leftarrow \frac{\tau(b_i, b_j)}{\mathrm{lt}(b_i)}$
14: $\quad\quad R'[i] \leftarrow \mathrm{append}(\lambda, R'[i])$
15: **return** (G', S', R')

We know that the s–vector of any critical pair (b_i, b_j) reduces to zero, because B is a reduced standard basis. Thus we can add the corresponding signatures of these s–vectors to the rules lists. We see in Line 13 how easily a rule is computed: For any element $g' \in G'$ it holds that $\mathrm{slm}(g') = 1$. Thus the rule is nothing else but the multiplier of the corresponding s–vector generator. Moreover, j is always bigger than i, thus we explicitly know that the rule is just the multiple of b_j.

With this we can present Algorithm 54, F5C, as a slightly variant of F5. Note that IncF5C differs to IncF5 in exactly one point: Instead of initializing

$$g_s \leftarrow (e_i, f_i)$$

in Line 3 we need to initialize it by

$$g_s \leftarrow (e_s, f_i)$$

since the indices have changed due to the reduction of the intermediate standard basis.

The following theorem is quite clear from the above discussion.

Theorem 6.2.1. *Let $F = \{f_1, \dots, f_r\}$, a finite set of homogeneous polynomials in $\mathcal{P}$ equipped with a well–order $<$, be the input of* F5C. *If the kth iteration of* IncF5 *terminates with output (G, R), then* $\mathrm{poly}(G)$ *is a standard basis for $\langle f_1, \dots, f_k \rangle$ w.r.t. $<$.*

Algorithm 54 The F5 Algorithm using reduced standard bases(F5C)

Input: $F = \{f_1, \ldots, f_r\}$ a finite subset of $\mathcal{P}$
Output: B a standard basis for $\langle F \rangle$ w.r.t. $<$

$G_1 \leftarrow \{(e_1, f_1)\}$
S = empty list
R = empty list
for $(i = 2, \ldots, r)$ **do**
  $f_i \leftarrow$ Reduce $\left(f_i, \text{poly}(G_{i-1})\right)$
  if $(f_i \neq 0)$ **then**
    $G_i, S, R \leftarrow$ IncF5C(f_i, G_{i-1}, S, R)
    $G_i, S, R \leftarrow$ ReduceF5(G_i, S, R)
  else
    $G_i \leftarrow G_{i-1}$
$B \leftarrow \text{poly}(G_r)$
return B

Proof. Let $(G'_{k-1}, S', R') =$ ReduceF5(G_{k-1}, S, R). S' and R' are valid for G'_{k-1} due to their constructions in ReduceF5. Moreover,

$$\langle f_1, \ldots, f_{k-1} \rangle = \langle \text{poly}(G_{k-1}) \rangle = \langle \text{poly}(G'_{k-1}) \rangle.$$

Thus our proof of Theorem 6.1.13 holds for the new parameters (G'_{k-1}, S', R') passed to IncF5 in the kth iteration, too. □

Remark 6.2.2. Note that in contrast to the variants of SigStd it is not so clear that reducing the intermediate standard bases is really an optimization. On the hand, thinking about the Rewritten Criterion it is possible that all those rules, which are deleted in ReduceF5, are carrying a lot more data and information about the ideal. This could lead to the detection of more useless pairs. On the other hand, one cannot prove which attempt is the better one due to the fact that F5 and F5C compute different critical pairs and use different signatures. Thus a complete comparison is not possible. We see in the experimental results presented in Section 6.4 that, in practice, F5C does not compute more zero reductions than F5. Moreover, it needs less memory and is faster than F5.[4]

In Section 5.2 we have seen that interreducing intermediate standard bases in SigStd no extra computations are needed. Clearly, since F5 is based on its criteria, S and R must be recomputed. At least, it seems so.

The following convention seems a bit strange, but makes sense in the world of signature–based algorithms as we see in Lemma 6.2.3.

Convention. Note that considering F5's criteria it is a bit tricky to keep the good properties of F5 alive in F5C. For this let us agree on the following way we reduce $\text{poly}(G_{k-1})$: Let $\text{poly}(G_{k-1}) = \{p_1, \ldots, p_{s-1}\}$. When minimizing $\text{poly}(G_{k-1})$, we remove an element p_i because there exists some other element p_j such that $\text{lm}(p_j) \mid \text{lm}(p_i)$. If we do so, we keep

[4] Again, as in Remark 5.2.2, in some not important cases F5 is faster than F5C due to the weighting of the overhead of reducing the intermediate standard bases in comparison to the whole computation.

$p_1, \ldots, p_{i-1}$ and move p_l to p_{l-1} for all $l > j$. When we go on reducing elements completely and normalizing the p_is we do not change their position or order the elements in any other way.

Lemma 6.2.3. *Any critical pair detected by the* F5 *Criterion in* F5 *is also detected by the* F5 *Criterion in* F5C.

Proof. Any element of current index k which is detected in F5 is also detected in F5C due to the fact that whenever we have deleted some $p_i \in \text{poly}(G_{k-1})$ during the interreduction process there exists a p_j in the reduced standard basis such that $\text{lm}(p_j) \mid \text{lm}(p_i)$. So any element detected by $\text{lm}(p_i)$ is also detected by $\text{lm}(p_j)$. Due to our above convention this also holds for elements of index $< k$. □

Corollary 6.2.4. *If the polynomials generating the input ideal $I = \langle f_1, \ldots, f_r \rangle \subset \mathcal{P}$ form a regular sequence $F = (f_1, \ldots, f_r)$, then* F5C *does not compute any zero reduction.*

Proof. Clear by Lemma 6.2.3. □

Remark 6.2.5. Note that we can even improve the F5 Criterion on elements of index $< k$ due to the following:Let $g_i, g_j \in G_{k-1}$ with index $(g_i) = \text{index}\left(g_j\right)$ such $i < j$ and such that $\text{poly}(g_i)$ reduces to p_l and $\text{poly}(g_j)$ reduces to p_m in the reduced intermediate standard basis B_{k-1}. It follows that $l < m$ by Convention 6.2, thus the correspoding labeled polynomials are $h_l = (e_l, p_l)$ and $h_m = (e_m, p_m)$ in G'_{k-1}. Now when checking the F5 Criterion for some multiple of h_m we can also use $\text{lt}(g_l)$ for a possible detection. This is not possible in F5, but in F5C only.

Also the above holds it is still a bit tricky to ensure the order of the elements in the reduced standard bases, since such a restriction can slow down computations. The nice thing is that we do not need to check any multiple of a lower index labeled polynomial by the F5 Criterion at all in F5C:

Lemma 6.2.6. *Let (uf, vg) be a critical pair in* F5C *such that $k = \text{index}(f) > \text{index}(g)$. If vg is detected by the* F5 *Criterion, then uf is also detected by the Rewritten Criterion.*

Proof. If vg is detected by the F5 Criterion, then there must exist some $h \in G$ such that $\text{index}(h) < \text{index}(g)$ and

$$\text{lm}(h) \mid \frac{\tau(f,g)}{\text{lm}(g)}.$$

This means that $\tau(g,h) \mid \tau(f,g)$. By Lemma 2.4.1 it follows that $\tau(f,h) \mid \tau(f,g)$ and thus

$$\frac{\tau(f,h)}{\text{lm}(f)} \mid \frac{\tau(f,g)}{\text{lm}(f)}.$$

By the algorithm's design the critical pair generated by f and h is considered before (uf, vg). The following two situations are possible:

(1) The critical pair generated by f and h is computed, then the rule $\frac{\tau(f,h)}{\text{lm}(f)}$ is added to $R[k]$. It follows that (uf, vg) is detected by this rule.

(2) The critical pair generated by f and h is detected either by the F5 or the Rewritten Criterion. In any case, this implies that (uf, vg) is detected, too.

□

From Lemma 6.2.6 we can follow that we do not need to recompute S in ReduceF5. Even more, S does no longer need to be a list of lists of terms, but just a set of terms. Assume we are in the kth iteration of IncF5, then it is enough to check labeled polynomials of index k by the F5 Criterion. This means we do not need to distinguish between different index levels in S, as all leading terms of elements of $\operatorname{poly}(G_{k-1})$ are allowed to be used for an element of index k.

We can do even better, namely we do not need to recompute new rules at all in ReduceF5. The next lemma shows that it is enough to check those generators of a critical pair, which have the current index.

Lemma 6.2.7. *Let (uf, vg) be a critical pair in* F5C *such that* $k = \operatorname{index}(f) > \operatorname{index}(g)$. *If vg is detected by the Rewritten Criterion, then uf is also detected by the Rewritten Criterion.*

The proof is similar to the one given for Lemma 6.2.6.

Proof. If vg is detected by the Rewritten Criterion, then there must exist some $h \in G$ such that $\operatorname{index}(h) < \operatorname{index}(f)$ and

$$\frac{\tau(g,h)}{\operatorname{lm}(g)} \mid \frac{\tau(f,g)}{\operatorname{lm}(g)}.$$

This means that $\tau(g,h) \mid \tau(f,g)$. By Lemma 2.4.1 it follows that $\tau(f,h) \mid \tau(f,g)$ and thus

$$\frac{\tau(f,h)}{\operatorname{lm}(f)} \mid \frac{\tau(f,g)}{\operatorname{lm}(f)}.$$

By the algorithm's design the critical pair generated by f and h is considered before (uf, vg). The following two situations are possible:

(1) The critical pair generated by f and h is computed, then the rule $\frac{\tau(f,h)}{\operatorname{lm}(f)}$ is added to $R[k]$. It follows that (uf, vg) is detected by this rule.

(2) The critical pair generated by f and h is detected either by the F5 or the Rewritten Criterion. In any case, this implies that (uf, vg) is detected, too.

□

Corollary 6.2.8. *In* F5C *there is no need to recompute rules in* ReduceF5*.*

Proof. As we have already seen the one place where rules of lower index are needed is when checking generators of critical pairs, which have lower index. By Lemma 6.2.7 we see that this is not needed at all. □

Algorithm 55 Incremental F5C step (IncF5C)

Input: f_i a polynomial, $G_{i-1} = \{p_1, \ldots, p_{s-1}\}$ a standard basis for $\langle f_1, \ldots, f_{i-1} \rangle$
Output: B a standard basis for $\langle f_1, \ldots, f_i \rangle$ w.r.t. $<$

```
B ← ∅, G ← ∅, P ← ∅, R ← empty list
S ← ∅
p_s ← f_i
t ← s
for (k = 1, ..., s − 1) do
    g_k ← (0, p_k)
    S ← S ∪ {lt(p_k)}
g_s ← (e_s, p_s)
G ← {g_1, ..., g_s}
for (k = 1, ..., s − 1) do
    u ← lc(g_k) τ(g_s, g_k) / lm(g_s)
    v ← lc(g_s) τ(g_s, g_k) / lm(g_k)
    if (!NonMinF5?(u g_s, S)) then
        P ← P ∪ {(u g_s, v g_k)}
while (P ≠ ∅) do
    P' ← Select(P) (critical pairs of minimal degree)
    while (P' ≠ ∅) do
        Choose (uf, vg) from P' with max_< {u sig(f), v sig(g)} minimal w.r.t. <.
        if (!RewriteF5?(u, f, R)) then
            if ((index(g) < s) or !RewriteF5?(v, g, R)) then
                P' ← P' \ {(uf, vg)}
                l ← max_< {u sig(f), v sig(g)}
                R ← addRule(l, R)
                r ← (l, u poly(f) − v poly(g))
                poly(r) ← Reduce(poly(r), poly(G_{i−1}))
                (r, P') ← SigRedF5(r, G, S, R, s, P')
                if (poly(r) ≠ 0 and r not sig–redundant w.r.t. G) then
                    for (k = 1, ..., t) do
                        if (lm(g_k) ∤ lm(r)) then
                            u ← lc(g_k) τ(r, g_k) / lm(r)
                            v ← lc(r) τ(r, g_k) / lm(g_k)
                            if (lm(u) siglm(r) ≠ lm(v) siglm(g_k)) then
                                if (!NonMinF5?(ur, S)) then
                                    if ((index(g_k) < s) or !NonMinF5?(v g_k, S)) then
                                        P ← P ∪ {(ur, v g_k)}
                    t ← t + 1
                    g_t ← r
                    G ← G ∪ {g_t}
B ← poly(G)
return B
```

All in all we have stripped down the complexity of the interreduction step quite a lot. Interreducing the intermediate standard bases does not only optimize reductions and reduces the number of useless critical pairs, it also provides a lot easier handling of the criteria and less checks. In the end we see that F5C is nothing else but a variant of SigStdRed. For this let us present the incremental part of F5C in Algorithm 55.

The main changes from IncF5 to Algorithm 55 should be clear:

- ▷ easier structures for S and R,
- ▷ initializing G completely similar to IncSigCrit,
- ▷ criteria checks only for elements of current index.

After all this optimizations, let us clarify what we mean when talking about F5C in the following:

Definition 6.2.9. F5C denotes the variant of SigStdRed calling IncF5C, NonMinF5?, and RewriteF5?.

Besides having achieved an optimized variant of F5, F5C represents an algorithm which is quite similar to SigStdRed and its derivatives. Thus we are ready to give a detailed discussion on similarities and differences of the various attempts in the signature–based world presented in this thesis.

6.3 Classifying F5 in the signature–based world

After this extensive introduction to the F5 Algorithm, also including optimizations due to interreducing intermediate standard bases, let us take a small break and try to collect differences and similarities of F5, SigStd, and all their derivatives. This is also helpful when we go on optimizing and generalizing F5 in the following sections.

We start with a comparison of F5 to SigStd. As SigStd is the common core of all other algorithms presented in Chapter 5, this is a natural point to start at.

(1) Comparing Algorithm 46 and Algorithm 32 with each other we see that F5 keeps track of two global lists, S and R, which consist of the criteria used to reject useless critical pairs. Due to the fact that the F5 Criterion and the Rewritten Criterion in F5 are a lot more aggressive than NonMin? and Rewrite? in SigStd, this bookkeeping is crucial.

(2) Based on the first point, F5 handles and passes always sets of labeled polynomials to IncF5, and not just polynomial sets, i.e. previously computed intermediate standard bases. It follows that as opposed to IncSigCrit IncF5 does not initialize the elements of G_{i-1} since they already carry their signatures.

(3) The Rewritten Criterion detects way more useless critical pairs than SIGSTD's implementation of (RW). In (RW) it is essential that another critical pair with the same signature exists, for the Rewritten Criterion only divisibility is neccessary. This is some optimization we have already found in AP, which requires in REWRITEAP? also only divisibility for detecting useless critical pairs. Whereas AP rewrites elements based on comparing the corresponding leading terms, F5 uses the information stored in the signatures. If F5 detects a critical pair (uf, vg) to be useless based on the Rewritten Criterion, then there exists some other combination of elements which have a signature greater than the one of the detected generator of (uf, vg).[5] In this sense the algorithm assumes that there exist "better" elements in G describing the polynomial data of (uf, vg). This "better" can be interpreted as *better reduced*, *sparser*, and so on. It is mainly based on the fact that the rewriting labeled polynomials are added to G later than f resp. g, thus the likelihood to get a better representation is taken into account.

(4) Looking at the F5 Criterion two diverging statements must be done:
   a) On the one hand, the F5 Criterion is way stronger than the general (NM) Criterion. NONMINF5? does check both generators of the critical pairs, whereas NONMIN? checks only the one corresponding to the signature of the resulting s–vector. This means that F5 must be able, in contrary to SIGSTD, to check also labeled polynomials of lower index. This leads to the more complex structure of S in F5. Whereas S is just a set of leading terms of labeled polynomials of lower index in SIGSTD, S is a list of lists $S[i]$ carrying the leading terms of the labeled polynomials of index i. This enables NONMINF5? to check an element of any given index correctly.
   b) On the other hand, SIGSTD and all its derivatives use zero reductions actively by adding the corresponding signature to the set S. F5 does not do this, but also uses the corresponding signatures as rules in the Rewritten Criterion. The disadvantage of this attempt is that searching in lists of rules is a bigger computational effort than just checking for the F5 Criterion.

(5) F5 chooses critical pairs of P by lowest possible degree, not by lowest possible signature. Thus the computations can only be done for homogeneous polynomials, otherwise later on computed elements of lower degree could destroy correctness of F5.

(6) F5 distincts between current index reductions, which must be ensured to be sig–safe, and lower index reductions, which are processed on the polynomial side only, since all elements to be reduced have the current index k, and all reducers of G_{k-1} have lower index. Thus no sig–unsafe reduction can happen. Note that this is true for SIGSTD and all its derivatives, too. Thus we can adopt this idea easily.

(7) Instead of allowing only sig–safe reductions in SIGREDF5 as it is done in SIGRED, F5 also allows sig–unsafe reductions. These are not real reductions, but a sig–unsafe

[5] Clearly the signature of the combination of those multiplied labeled polynomials has a signature smaller or equal to the one of the s–vector corresponding to the rejected critical pair.

reduction leads to a new critical pair for P'. In SIGSTD the construction of these critical pairs is postponed to INCSIGCRIT at the point the new element is already reduced and prepared for addition to G. The only real differences can be found in the following situation:

Assume a labeled polynomial f to be reduced by some other labeled polynomial of the same index, say g. Furthermore, assume that

$$\mathrm{lt}(f) = u\,\mathrm{lt}(g) \quad \text{and} \quad \mathrm{siglm}(f) < u\,\mathrm{siglm}(g)$$

for some $u \in \mathcal{P}$. Moreover, there exists a third labeled polynomial of the same index, h, and a term $v \in \mathcal{P}$ such that

$$\mathrm{lt}(f) = v\,\mathrm{lt}(h) \quad \text{and} \quad \mathrm{siglm}(f) > v\,\mathrm{siglm}(h).$$

What actions take place in the two algorithms?

- ▷ In F5 the new critical pair (ug, f) is added to P' and later on f is reduced by vh.
- ▷ In SIGSTD f is not reduced by ug, but by vh. Thus in the end an element not equal to f is added to G. It follows that SIGSTD does not generate the critical pair (vg, f) due to the lack of existence of f resp. $\mathrm{lt}(f)$ in G.

This means that the way of searching a possible reducer of current index is quite important for the following steps. A sig–safe reduction always takes place, correctness of the algorithms is ensured. But in a situation like the one above different critical pairs can be generated by the two algorithms. Finding heuristics for this selection of possible reducers of G is an area of active research these days.

(8) In SIGREDF5 the possible reducers are checked by the two criteria. This is the one difference which leads to several important facts:

a) F5 computes way less reduction steps than SIGSTD and all other variants.

b) Proving termination of F5 cannot be done the same way as the corresponding proof for SIGSTD. We show in Section 6.5 how to handle the termination issues of F5 quite elegant and without losing any performance at all. There we also give a more detailed discussion on how this process of rejecting reducers really works in the interior of F5.

We have seen at the end of Section 6.2 that SIGSTDRED and F5C are not so far apart as it seems from the above discussion of the respective basic algorithms.

AP, MM, and G2V are using SIGSTDRED as a basis, not SIGSTD, thus it is clear that when comparing Faugère's attempt to these we should switch from F5 to F5C. Looking at G2V, its description is a lot easier than F5's. This is based on the fact that coming from SIGSTDRED G2V uses mainly (NM) only. So assuming SIGSTDRED as some zero point between F5C and G2V we describe the algorithms' connection in the following way:

- ▷ G2V strips the criteria checks down: (RW) is only used in some special situation when generating new critical pairs. Thus it does more computations, but has less interrupts checking labeled polynomials.

- ▷ F5C focusses on more checks, less computations. This is not only present in the aggressiveness of the Rewritten Criterion, but also in the idea of checking the possible current index reducers in SigRedF5.

AP however has a really different origin than G2V. AP is rather a variant of F5 resp. F5C than a variant of SigStd. As it is described in [7] it is more or less a fork of F5. Besides giving a non–incremental description of the algorithm (see Section 7.4 for more details), the main purpose of AP is to illustrate the F5 Criterion with a simpler version of the Rewritten Criterion. This has two effects:

(1) AP's inner workings are readily understood in contrast to F5's quite complicated sub-algorithms.

(2) One can easily prove termination of AP. We see in Section 6.5 that the corresponding proof is a real problem for F5.

Based on the above discussion we can optimize the F5 Criterion even more: By (4)b we can achieve an easy, but as we see in the experimental results quite useful optimization: In the same way as SigStdRed uses zero reductions in its implementation of (NM) we can add the signature of an element which reduced to zero in F5 to S. As we have already mentioned, checking the F5 Criterion is, from a computational point of view, much faster than searching in a list of rewrite rules.

Thus we can define the following variant of F5C:

Definition 6.3.1. F5E denotes the variant of SigStdRed calling IncF5E, NonMinF5?, and RewriteF5?.

The "E" in F5E stands for "enhanced" as it incorporates not only the optimization of interreducing intermediate standard bases of F5C, but also the active usage of zero reductions introduced in SigStdRed. The differences between IncF5C and IncF5E are quite clear, but due to its impact on the algorithm let us give the pseudocode in detail in Algorithm 56:

The steps in Lines 39– 41 are clear from our discussion: We can delete the last rule from R, because we add this signature to S. The effect on the whole algorithm can be found in a new addition of calls of NonMinF5? in Lines 19 and 20: As we dynamically update S it is useful to check the F5 Criterion again when entering the reduction process of a critical pair.

With this last optimization of F5 the differences between F5E and the other signature–based standard basis algorithms drop down to

- ▷ using homogeneous input data in F5 only (we see in Section 7.1 that even this restriction can be removed from F5.),
- ▷ checking elements by a more aggressive implementation of (RW), and
- ▷ checking possible reducers in the current index reduction steps, too.

Algorithm 56 Incremental F5E step (IncF5E)

Input: f_i a polynomial, $G_{i-1} = \{p_1, \ldots, p_{s-1}\}$ a standard basis for $\langle f_1, \ldots, f_{i-1}\rangle$
Output: B a standard basis for $\langle f_1, \ldots, f_i\rangle$ w.r.t. $<$

$B \leftarrow \emptyset, G \leftarrow \emptyset, P \leftarrow \emptyset, R \leftarrow$ empty list
$S \leftarrow \emptyset$
$p_s \leftarrow f_i$
$t \leftarrow s$
for $(k = 1, \ldots, s-1)$ **do**
  $g_k \leftarrow (0, p_k)$
  $S \leftarrow S \cup \{\mathrm{lt}(p_k)\}$
$g_s \leftarrow (e_s, p_s)$
$G \leftarrow \{g_1, \ldots, g_s\}$
for $(k = 1, \ldots, s-1)$ **do**
  $u \leftarrow \mathrm{lc}(g_k)\frac{\tau(g_s,g_k)}{\mathrm{lm}(g_s)}$
  $v \leftarrow \mathrm{lc}(g_s)\frac{\tau(g_s,g_k)}{\mathrm{lm}(g_k)}$
  if $\left(!\,\text{NonMinF5?}\,(ug_s, S)\right)$ **then**
    $P \leftarrow P \cup \{(ug_s, vg_k)\}$
while $(P \neq \emptyset)$ **do**
  $P' \leftarrow \text{Select}(P)$ (critical pairs of minimal degree)
  while $\left(P' \neq \emptyset\right)$ **do**
    Choose (uf, vg) from P' with $\max_{<}\{u\,\mathrm{sig}(f), v\,\mathrm{sig}(g)\}$ minimal w.r.t. $<$.
    if $\left(!\,\text{NonMinF5?}(uf, S) \text{ and } !\,\text{RewriteF5?}(u, f, R)\right)$ **then**
      if $\left((\mathrm{index}(g) < s) \text{ or } (!\,\text{NonMinF5?}(vg, S) \text{ and } !\,\text{RewriteF5?}(v, g, R))\right)$ **then**
        $P' \leftarrow P' \setminus \{(uf, vg)\}$
        $l \leftarrow \max_{<}\{u\,\mathrm{sig}(f), v\,\mathrm{sig}(g)\}$
        $R \leftarrow \mathrm{addRule}(l, R)$
        $r \leftarrow \left(l, u\,\mathrm{poly}(f) - v\,\mathrm{poly}(g)\right)$
        $\mathrm{poly}(r) \leftarrow \text{Reduce}\left(\mathrm{poly}(r), \mathrm{poly}(G_{i-1})\right)$
        $(r, P') \leftarrow \text{SigRedF5}(r, G, S, R, s, P')$
        if $\left(\mathrm{poly}(r) \neq 0\right.$ and r not sig–redundant w.r.t. $\left.G\right)$ **then**
          for $(k = 1, \ldots, t)$ **do**
            if $(\mathrm{lm}(g_k) \nmid \mathrm{lm}(r))$ **then**
              $u \leftarrow \mathrm{lc}(g_k)\frac{\tau(r,g_k)}{\mathrm{lm}(r)}$
              $v \leftarrow \mathrm{lc}(r)\frac{\tau(r,g_k)}{\mathrm{lm}(g_k)}$
              if $\left(\mathrm{lm}(u)\,\mathrm{siglm}(r) \neq \mathrm{lm}(v)\,\mathrm{siglm}(g_k)\right)$ **then**
                if $\left(!\,\text{NonMinF5?}\,(ur, S)\right)$ **then**
                  if $\left((\mathrm{index}(g_k) < s) \text{ or } !\,\text{NonMinF5?}(vg_k, S)\right)$ **then**
                    $P \leftarrow P \cup \{(ur, vg_k)\}$
          $t \leftarrow t + 1$
          $g_t \leftarrow r$
          $G \leftarrow G \cup \{g_t\}$
      else if $(\mathrm{poly}(r) = 0)$ **then**
        Delete last rule from R.
        $S \leftarrow S \cup \{\mathrm{slm}(l)\}$
$B \leftarrow \mathrm{poly}(G)$
return B

Figure 6.4.1: Coloration of the results for different variants of F5

All in all it is quite amazing how all these different algorithms, with their various approaches and origins fit together so well. Optimizing F5 does not only have a positive effect on its performance, but yields suprisingly to even more similarities with SigStd resp. SigStdRed.

6.4 Experimental results

Let us try to give a comparison of the 3 variants of F5 we have discussed until now:

(1) the initial F5 Algorithm,

(2) the variant F5C interreducing the intermediate standard bases, and

(3) the variant F5E, an F5C Algorithm actively using zero reductions.

As in Section 5.6 we use colors to classify the timings. This time we compare only 3 different algorithms, Figure 6.4 illustrates the coloration.

The presented variants of F5 are implemented in Singular and can be also downloaded from

`git@github.com:ederc/Sources.git`[6].

We used the same revision of Singular as for the algorithms of Chapter 5. Moreover, the very same computer was used for the following computations. In this series of tests we always compute in the respective polynomial ring over a field of characteristic $32,003$ using the graded reverse lexicographical order $<_{\mathrm{dp}}$.

Looking at the timings given in Table 6.1 one sees that, besides the `Katsura-n-h` examples, F5C is always faster than F5, and F5E is always faster than F5C. Looking at examples like `Eco-n-h` there is a factor of nearly 10 between F5 and F5E. Whereas we see a clear benefit of interreducing the intermediate standard bases in F5C in all examples, again besides the `Katsura-n-h` ones, the positive effect of using zero reductions actively in F5E differs for the examples. For `Cyclic-n-h` the benefit is much less than, for example, for `Eco-n-h` or `F-xxx-h`. The reason for this lies in the fact that whereas in `Cyclic-n-h` not so many zero reductions appear, F5E does not benefit as much as it does in `Eco-n-h`

[6] You can get the git repository by typing `git clone git@github.com:ederc/Sources.git`.

where the number of zero reductions is going down by a factor of 10 switching from F5C to F5E (see Table 6.2). Moreover, F5 and F5C are not able to compute a standard basis for `Eco-11-h`. Here the idea of using zero reductions actively, improving the F5 Criterion check, is fundamental for the computations.

Why do the `Katsura-n-h` examples not take advantage when using F5C or F5E? The explanation can be found in Table 6.2: Those examples correspond to complete intersection, that means that neither F5 nor F5C nor F5E compute any zero reduction. With this in mind none of the optimizations mentioned in the last sections can improve the computations, as all useless critical pairs are already found by the default F5 Criterion. Moreover, interreducing the intermediate standard bases and bookkeeping of the zero reductions create an overhead on the computations such that the timings of F5C and F5E are even worse than those of F5. On the other hand, one really should point out that F5C and F5E have a much smaller memory footprint than F5 due to the fact that whereas the interreduction and the newly added criteria do not lead to more rejections of critical pairs or less reduction steps (see tables 6.5 and 6.4), they detect those useless elements faster and more efficient.

Comparing the 3 variants of F5 to the algorithms presented in Chapter 5 we should mention that F5C and F5E are implemented in the same framework as AP, MM, and G2V. Thus comparing the results of this section with the ones given in Section 5.6 is possible.

Clearly, the fact that F5C does not actively use zero reductions in the F5 Criterion check is a huge drawback, leading to worse timings than most of the algorithms of Chapter 5, besides examples like `Cyclic-n-h` and `Katsura-n-h`. The variants of F5 benefit from using the criteria to detect also useless reducers in SigRedF5. This leads to less reduction steps and thus less memory consumption. Having a closer look at G2V, F5E is always faster and less memory hungry. The only algorithm of Chapter 5 beating F5E in some examples is AP: AP is the fastest algorithm for the `Eco-n-h` examples as well as the `F-855-h` example. There it takes advantage of chosing critical pairs in (RW) by the least common multiple of the polynomial leading terms.

All in all one can summarize the experimental results to the following statement: F5E is the fastest algorithm in a wide range of example classes. In some settings AP, which is also just a variant of F5[7], gives the best results. In these examples the choice by a minimal leading term seems to be the best possible one. It boils down to use these two variants of F5 for an efficient signature–based standard basis computation. Having a heuristic to decide when to use which of the two variants is an ongoing research project of the author.

6.5 Termination–ensured variants of F5

An open question surrounding F5 regards its termination. In a traditional standard basis algorithm, like the ones presented in chapters 1 and 2, termination is based on the ability of the algorithm to exploit the fact that the polynomial ring is Noetherian: Each

[7] See Section 6.3 for more details.

Test case	F5	F5C	F5E
Cyclic-7-h	13.270	5.880	5.630
Cyclic-8-h	77,789.770	7,247.690	5,266.440
Eco-8-h	2.920	1.930	0.300
Eco-9-h	226.830	61.500	6.650
Eco-10-h	12,121.180	2,592.830	198.230
Eco-11-h	–.–	–.–	8,367.680
F-633-h	0.000	0.000	0.000
F-744-h	72.970	36.740	20.520
F-855-h	18,174.560	3,019.870	620.880
Gonnet-83-h	7,037.670	220.540	11.860
Katsura-8-h	0.050	0.050	0.050
Katsura-9-h	0.390	0.400	0.400
Katsura-10-h	4.070	4.430	4.410
Katsura-11-h	49.880	61.080	61.170
Schrans-Troost-h	2.860	3.600	3.550

Table 6.1: Time needed to compute a standard basis, given in seconds.

Test case	F5	F5C	F5E
Cyclic-7-h	76	76	36
Cyclic-8-h	1,540	1,540	244
Eco-8-h	322	322	57
Eco-9-h	929	929	120
Eco-10-h	2,544	2,544	247
Eco-11-h	-	-	502
F-633-h	2	2	2
F-744-h	498	498	323
F-855-h	2,829	2,829	835
Gonnet-83-h	8,129	8,129	2,005
Katsura-8-h	0	0	0
Katsura-9-h	0	0	0
Katsura-10-h	0	0	0
Katsura-11-h	0	0	0
Schrans-Troost-h	0	0	0

Table 6.2: Number of zero reductions computed by the algorithms.

Test case	F5	F5C	F5E
Cyclic-7-h	29.022	19.031	18.031
Cyclic-8-h	1, 833.085	1, 250.106	569.513
Eco-8-h	23.046	17.062	4.061
Eco-9-h	198.151	151.168	26.684
Eco-10-h	1, 655.040	1, 345.006	168.827
Eco-11-h	–.–	–.–	1, 046.751
F-633-h	0.000	0.036	0.036
F-744-h	133.256	79.814	54.314
F-855-h	1, 375.004	985.874	313.612
Gonnet-83-h	160.621	52.533	15.889
Katsura-8-h	2.000	1.500	1.500
Katsura-9-h	8.500	6.000	6.000
Katsura-10-h	36.500	23.018	23.018
Katsura-11-h	174.051	105.602	105.602
Schrans-Troost-h	34.029	20.037	20.037

Table 6.3: Memory used to compute a standard basis, given in Megabyte.

Test case	F5	F5C	F5E
Cyclic-7-h	1,018	1,018	978
Cyclic-8-h	7,066	7,066	5,770
Eco-8-h	830	830	565
Eco-9-h	2,087	2,087	1,278
Eco-10-h	5,123	5,123	2,826
Eco-11-h	-	-	6,219
F-633-h	56	56	56
F-744-h	2,089	2,089	1,914
F-855-h	7,922	7,922	5,928
Gonnet-83-h	12,111	12,111	5,987
Katsura-8-h	120	120	120
Katsura-9-h	247	247	247
Katsura-10-h	502	502	502
Katsura-11-h	1,013	1,013	1,013
Schrans-Troost-h	393	393	393

Table 6.4: Number of critical pairs not detected by the respective criteria used.

Test case	F5	F5C	F5E
Cyclic-7-h	100,569	100,569	83,880
Cyclic-8-h	14,823,873	14,823,873	3,403,874
Eco-8-h	186,854	186,854	18,514
Eco-9-h	1,996,849	1,996,849	136,842
Eco-10-h	19,755,560	19,755,560	1,019,439
Eco-11-h	-	-	7,374,779
F-633-h	366	366	366
F-744-h	789,072	789,072	435,869
F-855-h	12,294,951	12,294,951	2,633,666
Gonnet-83-h	278,419	278,419	64,788
Katsura-8-h	1,634	1,634	1,634
Katsura-9-h	5,371	5,371	5,371
Katsura-10-h	18,343	18,343	18,343
Katsura-11-h	63,194	63,194	63,194
Schrans-Troost-h	14,010	14,010	14,010

Table 6.5: Number of all reduction steps during the computations.

Test case	F5	F5C	F5E
Cyclic-7-h	949	758	758
Cyclic-8-h	5,534	3,402	3,402
Eco-8-h	516	249	249
Eco-9-h	1,167	499	499
Eco-10-h	2,589	979	979
Eco-11-h	-	-	1,968
F-633-h	62	60	60
F-744-h	1,602	204	204
F-855-h	5,106	688	688
Gonnet-83-h	3,999	1,156	1,156
Katsura-8-h	128	105	105
Katsura-9-h	256	202	202
Katsura-10-h	512	399	399
Katsura-11-h	1,024	784	784
Schrans-Troost-h	401	189	189

Table 6.6: Size of the resulting standard basis.

polynomial added to the standard basis G during the computations expands the leading ideal of G. This can happen only a finite number of times.

Moreover, even for the signature–based algorithms presented in Chapter 5 termination can be shown (see Theorem 4.2.7; termination of all variants of SigStd is proven in the very same way).

In F5 an optimization during the reduction step introduces unforeseen issues regarding termination: Possible, also sig–safe reductions are rejected due to F5's criteria. This leads to the fact that F5 can add new elements to G which are redundant for the standard basis. On the one hand, until today no non–terminating example for F5 is known[8], but on the other hand there is still a correct and gapless proof of termination missing.

Different approaches tackling this problem have been published:

(1) In [8] Ars tries to use Buchberger's criteria to determine a degree bound for F5.

(2) In [78] Gash uses the Macaulay bound and breaks the computations down to a basic standard basis algorithm without any criteria checks for everything above the bound.

Besides presenting the above mentioned methods to ensure termination, and discussing their drawbacks from a computational point of view, we give a variant of F5 which ensures termination using data F5 computes itself. This variant changes just a few lines of code in an existing F5 implementation, but terminates for sure and introduces no penalty to performance at all. Later on we even combine our ideas with Ars' leading to a terminating algorithm which is sometimes even faster than F5.

Remark 6.5.1. Proving termination of Faugère's attempt computing standard bases is not a mere problem of F5, but of all its variants. It is mainly based on the usage of the Rewritten Criterion and the fact that possible reductions are rejected not only due to sig–unsafeness, but also based on criteria checks. Thus speaking about F5 in the following, the very same holds for F5C and F5E.

Definition 6.5.2. A labeled polynomial f computed in F5 is called *redundant* if there exists $g \in G$ at the moment SigRedF5 returns f such that $\operatorname{lt}(g) \mid \operatorname{lt}(f)$.

Remark 6.5.3.

(1) Note that also in a Buchberger–like algorithm a standard basis G can be computed such that there exist $p, q \in G$ with $\operatorname{lt}(q) \mid \operatorname{lt}(p)$. The important difference to the above definition is that if this situation appears in Buchberger–like algorithm p is added to G *before* q, not the other way around. In Definition 6.5.2 we explicitly require that f is added to G *after* g. This is a situation which is only possible in the signature–based world due to the fact that some reductions are not allowed.

(2) Furthermore the above situation really is a problem only in F5 as we see in the following: Whenever a redundant labeled polynomial is added to any of the algorithms presented in Chapter 5 a sig–unsafe reduction has been rejected. Thus in the next round of critical pairs the corresponding reduction is available and computed. So termination of the algorithm can be ensured as shown in Theorem 4.2.7. In contrast

[8] In the past, non–terminating examples were always based on implementational errors.

to this F5 can reject a reduction not only based on its sig–unsafeness, but also due to detecting the multiplied reducer by one of its criteria. In this situation no critical pair corresponding to this reduction step is generated later on, which leads to problems in proving termination of F5.

Even using the restriction to homogeneous input in F5 does not help us tackling this problem:

Convention. Let us assume that the critical pairs in P' have degree d during a call of IncF5. It is clear that all critical pairs left in P have a degree $> d$. This means that at the moment $P' = \emptyset$ in the kth call of IncF5 G is a d–standard basis for $\langle f_1 \ldots, f_k \rangle$.

Definition 6.5.4. We denote the set of elements added to G during the step of degree d in a certain call of IncF5 by R_d.

Let us define the problematic situation.

Situation 6.5.5. Suppose that $R_d \neq \emptyset$ and for every element $f \in R_d$ f is redundant.

At first glance this situation seems to be completely theoretical, but it does appear in practice: Reviewing Example 6.1.17 we see that at degree 7 F5 adds

$$g_8 = (x^2 z e_3, y^5 t^2 - x^4 z t^2)$$

to G. At degree 8, however, $R_8 = \{g_{10}\}$ with

$$g_{10} = (z^3 t e_3, y^6 t^2 - x y^2 z t^4).$$

The reduction of g_{10} by $y g_8$ in SigRedF5 is rejected due to the F5 Criterion. Thus Situation 6.5.5 occurs even in small examples.

Lemma 6.5.6. *There exists a finite subset $F = \{f_1, \ldots, f_r\} \subset \mathcal{P}$ of homogeneous polynomials as input of* F5*, a degree d, and a point of IncF5's computation where* poly(G) *is* $(d-1)$*–standard basis for $\langle f_1, \ldots, f_r \rangle$ such that*

(1) $R_d \neq \emptyset$, *and*

(2) $L(\operatorname{poly}(G \cup R_d)) = L(\operatorname{poly}(G))$.

Proof. Such an input F is given in Example 6.1.17: Computing a standard basis for $I = \langle x^2 y - z^2 t, xz^2 - y^2 t, yz^3 - x^2 t^2 \rangle$ we end with a set $G = \{g_1, \ldots, g_9\}$ where $\operatorname{poly}(G)$ is 7–standard basis for I. $R_8 = \{g_{10}\}$ with $\operatorname{lt}(g_8) \mid \operatorname{lt}(g_{10})$, so $L(\operatorname{poly}(G \cup R_8)) = L(\operatorname{poly}(G))$. $\square$

Remark 6.5.7. In [63] Faugère argues in Corollary 2 that termination of F5 follows from the (unproven) assertion that if F5 does not reduce any polynomial to zero, then for any d $L(\operatorname{poly}(G)) \neq L(\operatorname{poly}(G \cup R_d))$ where $\operatorname{poly}(G)$ is a $(d-1)$–standard basis for the input ideal. We see that although there is no zero reduction in Example 6.1.17 $L(\operatorname{poly}(G \cup R_8)) = L(\operatorname{poly}(G))$. Thus we have found a counterexample showing that Theorem 2 and, by association, Corollary 2 of [63] are incorrect. It follows that termination of F5 remains unproven, even for regular sequences: There could be infinitely many steps where redundant labeled polynomials are added to G.

By contrast, Lemma 6.5.6 is not true for a Buchberger–like algorithm. Such an algorithm always expands the leading ideal when a polynomial does not reduce to zero, which ensures termination.

In the last couple of years two approaches to solve the termination issue of F5 have been published. We discuss them shortly, give the rough ideas and show the drawbacks of the attempts:

(1) In 2005 Ars presented a variant of F5 in [8] which suggests to determine a degree bound using Buchberger's 2nd Criterion. The following three facts outline the general idea:

▷ A global variable $d_{\max} = 0$ is initialized, which stores a degree of a polynomial.

▷ P' is a second set of the critical pairs, like P. This set is used to determine a degree bound only, not giving any other impact on the standard basis computation at all.

▷ Whenever a new element f is added to G in IncF5 a copy of each critical pair generated by f and not detected by Buchberger's 2nd Criterion is stored in P'. Moreover, the elements already in P' are again checked by Buchberger's 2nd Criterion, and removed from P' when detected to be useless. After this process has finished, $d_{\max}$ is set to be the highest degree of an element of P'.

If the degree of all critical pairs in P exceeds $d_{\max}$, then Buchberger's 2nd Criterion implies that the algorithm has already computed a standard basis, and can terminate.

It is important to maintain the distinction between the two sets of critical pairs, as otherwise the correctness of the algorithm is no longer assured: Buchberger's 2nd Criterion does not take the signatures of the labeled polynomials into account.

Two major drawbacks of this approach are clear:

a) Every critical pair is computed and checked twice: once for Buchberger's 2nd Criterion, and again for F5's criteria. Although the F5 Criterion also checks for divisibility, it checks only labeled polynomials of smaller index, whereas Buchberger's 2nd Criterion checks all labeled polynomials. For most input data the number of elements of the current index in IncF5 is much larger than the total of all labeled polynomials of smaller index. We see in the experimental results of the algorithm that this inconspicuous check can accumulate a significant time penalty.

b) To make things even worse, the standard F5 Algorithm generally terminates from its own internal mechanisms before the degree $d_{\max}$ is even reached. Thus, except for pathological cases, the penalty for this short–circuiting machinery is not compensated by a discernible benefit.

We denote this variant of F5 by F5B, the "B" stands for "Buchberger" in this setting.

(2) Gash presented another approach in [78], which reintroduces some more reductions to zero due to switching between different criteria checks of useless critical pairs. Gash denotes his variant by F5t, where the "t" indicates an ensured termination of the algorithm.

Again, let us give a short summary of the ideas behind F5t:

- ▷ As in F5B a degree bound is initialized. Here the Macaulay bound M for regular sequences (see [16, 114] and Section 1.9 for more details) is used.
- ▷ Once the degree of the labeled polynomials cosidered in F5t exceeds $2M$, redundant labeled polynomials are stored in a set D different from G.
- ▷ Whenever Situation 6.5.5 happens for degrees $d > 2M$ all elements of R_d are reduced completely w.r.t. $G \cup D$ not taking care of sig–safeness at all. Any polynomial which does not reduce to zero in this process is added to D instead of G.
- ▷ Due to the possible sig–unsafeness of the reduction described in the previous point all rewrite rules in R coming from such labeled polynomials in R_d are deleted.
- ▷ Subsequently, s–vectors generated using an element of D are reduced without regard to any criterion, and those that do not reduce to zero are also added to D, generating new critical pairs.

There are 4 major drawbacks of this approach:

a) The reintroduction of zero reductions incurs a performance penalty. In Gash's experiments this penalty is minimal, but these were performed on relatively small systems without many redundant polynomials. In some systems, like `Katsura-9-h`, F5 generates hundreds of redundant labeled polynomials.

b) F5t needs to keep track of two different sets of labeled polynomials, G and D, for generating critical pairs. Moreover, it uses a completely new reduction process at some point of the computations. Trying to incorporate the ideas of F5t to an existing F5 implementation adds a significant amount of complicated code, which is quite hard to handle in an optimized way.

c) It has to abandon some signatures due to the new, sig–unsafe reduction process. Thus, a large number of useless critical pairs can be left undetected and increase the computational overhead leading to zero reductions.

d) The doubled Macaulay bound $2M$ controls the point at which D is introduced to the computation, and thus the number of elements in D. Nevertheless this bound is quite imprecise and ad–hoc: In some experiments from [78], F5t terminates on its own before polynomials reach degree $2M$. For other input, F5t yields polynomials of degrees well beyond $2M$, and a higher bound would be desirable.

Both approaches are based on the idea to get some information from outer sources about the standard basis computation F5 is doing. This leads to the fact that a terminating variant of F5 can only be achieved by being highly dependent on different algorithms. Moreover, implementing a high–performance, low–level F5 Algorithm is a long–term attempt. Adding this high amount of changes, regardless of whether going to Ars' or Gash's approach, can be nearly impossible without rewriting a greater part of already existing code, besides the implementation of the new ideas.

So we need to find a less thwarting way to solve the issue of termination for F5. The idea presented in the following was first published in [57] by Gash, Perry, and the author. Starting the search for a solution, the first idea that comes into one's mind is to ignore the redundant labeled polynomials in F5's computations. Sadly it is not that easy to sort the wheat from the chaff.

Example 6.5.8. Suppose we modify the F5 Algorithm to discard all critical pairs that have at least one generator being a redundant labeled polynomial. Furthermore, consider for the following two examples polynomial rings over a ground field of characteristic 7583:

(1) For `Katsura-5-h`, the algorithm no longer terminates. It computes an increasing chain of labeled polynomials with leading terms $x_2 x_3^k x_5 x_6$ and signatures $x_2^k x_4$ for $k \geq 1$.

(2) For `Cyclic-8-h`, the algorithm terminates, but its output is not a standard basis at all.

This is a quite amazing fact: How can critical pairs involving "redundant" polynomials be necessary? To answer this question, let us define a more consistent notation.

Definition 6.5.9. Let f and g be two labeled polynomials computed in F5. A critical pair (uf, vg) is called an *SB–critical pair* if neither f nor g is redundant. If a critical pair is not an SB–critical pair, then we call it an F5–*critical pair*.

In the following we show that F5–critical pairs are necessary for the correctness of the F5 Algorithm. On this way, the intention of the above definition gets clearer.

Lemma 6.5.10. *Let f and g be two labeled polynomials in G and assume that* $\mathrm{lt}(g) \mid \mathrm{lt}(f)$. *Then the s–vector $f - vg$ is not generated in* IncF5.

Proof. Before entering IncF5 the next generator of the input ideal is reduced w.r.t. the intermediate standard basis in F5. So it follows that $\mathrm{poly}(f)$ is not in the input of IncF5. This means that the reduction of f by vg must have been rejected, i.e. vg is detected by one of F5's criteria. But then it is also detected, either when generating the critical pair (f, vg) or before constructing $f - vg$. ☐

Lemma 6.5.11. *If R_d satisfies Situation 6.5.5 and $f \in R_d$, then we find an element $g \in G$ such that g is not redundant and* $\mathrm{lt}(g) \mid \mathrm{lt}(f)$.

Proof. If a reducer h of f is redundant, then there needs to exist another element $g \in G$ such that $\mathrm{lt}(g) \mid \mathrm{lt}(h)$. Clearly, then also $\mathrm{lt}(g) \mid \mathrm{lt}(f)$ holds. Due to the homogeneous input of F5 we can go down this chain of reducers to the minimal degree, say d. At this point it is left to show that there do not exist two polynomials g, h with $\deg(g) = \deg(h) = d$ such that $\mathrm{lm}(g) = \mathrm{lm}(h)$.

Let us prove this by contradiction: Assume that g and h with the above properties exist in G. It follows that the reduction of one by the other in SigRedF5 was rejected. W.l.o.g. we assume that g was computed before h, so the reduction of h by g was forbidden. There are three possibilities:

(1) If $\operatorname{index}(g) < \operatorname{index}(h)$, then the reduction of h by g would always take place and SIGREDF5 could not reject it at all. So we can assume that $\operatorname{index}(g) = \operatorname{index}(h)$.

(2) If g is rejected by the F5 Criterion, then g should not have been computed in the first place.

(3) If g is rejected by the Rewritten Criterion, then there exists an element $r \in G$ such that $\operatorname{slt}(r) \mid \operatorname{slt}(g)$ and r has been computed after g. As F5 computes incrementally on the degree of the homogeneous elements it follows that $\deg(r) = \deg(g)$. Hence $\operatorname{slt}(r) = \operatorname{slt}(g)$ and due to the homogeneity of the elements r cannot be a predecessor of g. Thus the computation of r would have been rejected by the Rewritten Criterion.

Thus $\operatorname{lm}(g) \neq \operatorname{lm}(h)$. It follows that we arrive at a reducer, which is not redundant, after finitely many steps. □

Using the above two lemmata we can prove the following main result concerning redundancy in F5:

Theorem 6.5.12. *Let f and g be labeled polynomials computed in* F5. *If (uf, vg) is an* F5*–critical pair, then one of the following statements holds at the moment of creation of the corresponding s–vector $uf - vg$:*

(1) $u \operatorname{poly}(f) - v \operatorname{poly}(g)$ has a standard representation w.r.t. $\operatorname{poly}(G)$.

(2) There exist an SB–critical pair $(u'f', v'g')$, a finite set $W \subset \{1, \dots, m\}$ with $m = \#(G)$, and terms t_w for all $w \in W$ such that

$$u \operatorname{poly}(f) - v \operatorname{poly}(g) = u' \operatorname{poly}(f') - v' \operatorname{poly}(g') + \sum_{w \in W} t_w \operatorname{poly}(g_w), \qquad (6.5.1)$$

where $\tau(f, g) = \tau(f', g')$, and $\tau(f', g') > t_w \operatorname{lt}(g_w)$ for all $w \in W$.

Proof. W.l.o.g. we assume that both f and g are redundant, the case where only one of them is redundant is similar. By Lemma 6.5.11 there exists for f resp. g at least one reducer f' resp. g' which is not redundant. By Lemma 6.5.10 we can assume that $d = \max\{\deg(f), \deg(g)\} < \deg(uf - vg)$. Let

$$\lambda = \frac{\operatorname{lt}(f)}{\operatorname{lt}(f')},$$
$$\sigma = \frac{\operatorname{lt}(g)}{\operatorname{lt}(g')}.$$

We know that $\operatorname{poly}(G)$ is a d–standard basis, thus we can represent

$$\operatorname{poly}(f) = \lambda \operatorname{poly}(f') + \sum_{u \in U} t_u \operatorname{poly}(g_u),$$
$$\operatorname{poly}(g) = \sigma \operatorname{poly}(g') + \sum_{v \in V} t_v \operatorname{poly}(g_v),$$

where $\lambda \operatorname{lt}(f') > t_u \operatorname{lt}(g_u)$ for all $u \in U$, $\sigma \operatorname{lt}(g') > t_v \operatorname{lt}(g_v)$ for all $v \in V$, and $U, V \subset \{1, \ldots, m\}$. By construction $\tau(f', g') \mid \tau(f, g)$, so there exists a term $\gamma \in \mathcal{P}$ such that

$$u \operatorname{poly}(f) - v \operatorname{poly}(g) = \gamma \left(u' \operatorname{poly}(f') - v' \operatorname{poly}(g')\right) + \sum_{w \in W} t_w \operatorname{poly}(g_w) \qquad (6.5.2)$$

where $W = U \cup V$, and $t_w = \frac{\tau(f,g)}{\operatorname{lt}(f)} t_u$ if $w \in U \setminus V$, $t_w = \frac{\tau(f,g)}{\operatorname{lt}(g)} t_v$ if $w \in V \setminus U$, and $t_w = \frac{\tau(f,g)}{\operatorname{lt}(f)} t_u - \frac{\tau(f,g)}{\operatorname{lt}(g)} t_v$ if $w \in U \cap V$. In Equation 6.5.2 we must distinguish two cases:

(1) If $\gamma > 1$, then $\deg(u'f' - v'g') < \deg(uf - vg)$. Thus $u'f' - v'g'$ is already computed (or rewritten after detection by one of F5's criteria) using a lower degree computation, which has already finished. It follows that there exists a standard representation of $u' \operatorname{poly}(f') - v' \operatorname{poly}(g')$ w.r.t. $\operatorname{poly}(G)$, and thus also a standard representation of $u \operatorname{poly}(f) - v \operatorname{poly}(g)$ w.r.t. $\operatorname{poly}(G)$.

(2) If $\gamma = 1$, then two things can happen: Statement (1) holds if $u'f' - v'g'$ is already computed in IncF5, otherwise Statement (2) holds.

□

Remark 6.5.13.

(1) Theorem 6.5.12 implies that an F5–critical pair might not generate a redundant polynomial, but it can be used to rewrite an SB–critical pair which is *not* computed. For example, suppose that F5 adds f to G, where f is redundant, as there already exists $g \in G$ such that $u \operatorname{lt}(g) = \operatorname{lt}(f)$ for some term $u \in \mathcal{P}$. This means that the reduction of f by ug was, for example, rejected by one of F5's criteria. In the following it is not uncommon that the algorithm encounters some $h \in G$, h not redundant, such that $v = \frac{\tau(g,h)}{\operatorname{lt}(g)}$, $w = \frac{\tau(g,h)}{\operatorname{lt}(h)}$, and $u \mid v$. In this situation the SB–critical pair (vg, wh) is not computed, since it is rejected as vg is again detected by the very same criterion ug was beforehand. Moreover, assume that $\operatorname{lt}(f) \mid \tau(g, h)$. Now it is possible that (vg, wh) is necessary for the correctness of the standard basis $\operatorname{poly}(G)$ in the end, but it is not computed as F5 renders it as useless under the assumption that $(v'f, w'h)$ exists.

(2) Due to these facts, the notions of "necessary" and "redundant" critical pairs or labeled polynomials are somewhat ambiguous in F5. On the other hand, the notions of F5– and SB–critical pairs are absolute and do not change during the ongoing computations of F5.

Let us state the situation, essential for the understanding of the necessity of F5–critical pairs, covered by Theorem 6.5.12:

Situation 6.5.14. Let (uf, vg) be an F5–critical pair. Suppose that all SB–critical pairs $(u'f', v'g')$ corresponding to Statement (2) of Theorem 6.5.12 are rejected by one of F5's criteria, but lack a standard representation w.r.t. G.

Note that Situation 6.5.14 is possible, for example, the Rewritten Criterion can reject all SB–critical pairs $(u'f', v'g')$. With this we can state the main observation made from our discussion on F5– and SB–critical pairs:

Corollary 6.5.15. *In Situation 6.5.14 the computation of a standard representation of* (uf, vg) *w.r.t. G is necessary for the correctness of* F5.

Proof. Since $(u'f', v'g')$ is an SB–critical pair rejected by at least one of F5's criteria and $u'\operatorname{poly}(f') - v'\operatorname{poly}(g')$ has no standard representation w.r.t. $\operatorname{poly}(G)$, the only possibility to receive a standard representation for $u'\operatorname{poly}(f') - v'\operatorname{poly}(g')$ is to compute a standard representation for $uf - vg$ w.r.t. G and rewrite Equation 6.5.1. □

Now our task is to use the information from the connection between F5– and SB–critical pairs we have found in Theorem 6.5.12: Since we cannot rely on an expanding of the leading ideal when adding new labeled polynomials to G in F5, we use a similar starting point as F5B and F5t, a degree bound d_{SB} up to which computations in F5 are done. d_{SB} stores the maximal possible degree of an SB–critical pair found until that point of the computation. By Theorem 6.5.12 we know that we need to include all SB–critical pairs in the computation of d_{SB}, not only the ones not detected by F5's criteria. The idea is to process new elements until the minimal degree d of elements in the pair set P (i.e. the degree of the elements in P') is greater than d_{SB}. At this point only F5–critical pairs are left, thus no SB–critical pair relies on their computations and reductions. The information stored in the F5–critical pairs left over is not relevant for G, thus we can terminate the computations of F5.

In the following we describe how one has to adjust F5 to receive a variant incorporating the above mentioned ideas and ensures termination. As we see in the presented pseudo code the changes to be made are minimal, just adding a few more lines to the code. Based on this we denote the variant F5+, illustrating the little topping needed to achieve a solution to the termination issue of F5.

The motivation of F5+'s attempt is that F5 knows that a labeled polynomial f is redundant if a reduction of f in SigRedF5 is not processed due to one of F5's criteria. Thus F5 *knows* that f is redundant at this point of the computation. Our aim is to ensure in F5+ that the algorithm does *not forget* this fact. As long as this information remains available to the algorithm, identifying F5– and SB–critical pairs is trivial.

We explain how to achieve this task presenting the differences in the pseudo code of F5+'s implementation of IncF5 in Algorithm 57.

(1) As a first step we need to modify the data structure of a labeled polynomial f in order to distinguish redundant and not redundant elements. For this we add a third entry, a boolean flag b such that

$$b = \begin{cases} 1 & \text{if f is redundant,} \\ 0 & \text{otherwise.} \end{cases}$$

With this the data structure of a labeled polynomial handled in the kth iteration of the algorithm changes to

$$f = (l, p, b) \in \mathcal{P}^k \times \mathcal{P} \times \{0, 1\}.$$

The new input to the current iteration step of IncF5+ is assumed to be not redundant[9], so it is initialized with $b = 0$ in Line 3.

[9] It is already reduced w.r.t. $\operatorname{poly}(G_{i-1})$ in F5.

(2) Having added the information about redundancy to the labeled polynomials we can update the value of d_{SB} whenever we are in the process of generating an SB–critical pair. This happens at two different points of IncF5+: Once when initializing the first batch of critical pairs (Lines 9 and 10). There we know that one generator of the critical pairs is always g_s, which is assumed to be not redundant, thus checking for redundancy is enough on the second generator. Later on, when having added a new labeled polynomial r to G we again update the data structure when we recover an SB–critical pair. Note that this time it is important to check both generators of the pair for redundancy, since r could be a redundant element added to G (Lines 32 and 33).

(3) At this point it is left to set the redundancy flag correctly for the newly generated element r. Clearly, this needs to be done during the reduction process. We illustrate this in the pseudo code of F5+'s variant of SigRedF5 given in Algorithm 58. There we initialize the redundancy flag b by 0 in Line 1. From this on we only need to update b's value based on the execution of the reduction step:

- ▷ Whenever a reduction is rejected by F5's criteria, or due to sig–unsafeness, we set b to 1, the labeled polynomial is redundant at this moment (Line 18).
- ▷ If a reduction takes place, set b to 0, the labeled polynomial is not redundant at this point of the algorithm (Line 13).

In the end, the returned element h gets the correct redundancy flag (Line 19).

Remark 6.5.16. Note that it is not necessary to set the redundancy flag for an s–vector correctly at its initialization in Line 23 in Algorithm 57. Not until r is returned by SigRedF5 the redundancy flag b of r can be set. Being redundant is by Definition 6.5.2 a question of not allowed reductions and not based on the redundancy of the elements of which the s–vector arises from.

With this we can give a more precise definition of our naming convention:

Definition 6.5.17. We denote the variant of F5 calling IncF5+, and SigRedF5+ by F5+.

Let us show the main property of F5+, namely the fact that it is an algorithm:

Theorem 6.5.18. *Let $F = \{f_1, \ldots, f_r\} \subset \mathcal{P}$ be a finite set of homogeneous polynomials, the input of* F5+*. If* F5+ *terminates, then its result is a standard basis for $I = \langle f_1, \ldots, f_r \rangle$.*

Proof. The statement is clear by the correctness of F5, which is proven in Theorem 6.1.13. □

Algorithm 57 Termination ensured incremental F5 step (IncF5+)

Input: f_i a polynomial, $G_{i-1} = \{g_1, \dots, g_{s-1}\}$ a set of labeled polynomials such that $\mathrm{poly}(G_{i-1})$ is a standard basis for $\langle f_1, \dots, f_{i-1} \rangle$, S a list of lists of terms in $\mathcal{P}$, R a list of $(i-1)$ lists of terms in $\mathcal{P}$

Output: G a set of labeled polynomials such that $\mathrm{poly}(G)$ is a standard basis for $\langle f_1, \dots, f_i \rangle$, S a list of i lists of terms in $\mathcal{P}$, R a list of i lists of terms in $\mathcal{P}$

$B \leftarrow \emptyset, G \leftarrow \emptyset, P \leftarrow \emptyset, P' \leftarrow \emptyset, R[i] \leftarrow$ empty list, $S[i] \leftarrow$ empty list, $d_{\mathrm{SB}} \leftarrow 0, d \leftarrow 0$
$t \leftarrow s$
$g_s \leftarrow (e_i, f_i, 0)$
$S[i] \leftarrow \mathrm{addF5Crit}\,(\mathrm{lt}(g_s), S[i])$
$G \leftarrow \{g_1, \dots, g_s\}$
for $(k = 1, \dots, s-1)$ **do**
 $u \leftarrow \mathrm{lc}(g_k) \frac{\tau(g_s, g_k)}{\mathrm{lm}(g_s)}$
 $v \leftarrow \mathrm{lc}(g_s) \frac{\tau(g_s, g_k)}{\mathrm{lm}(g_k)}$
 if (g_k not redundant) **then**
  $d_{\mathrm{SB}} = \max\{d_{\mathrm{SB}}, \deg(u g_k)\}$
 if $\big(!\,\textsc{NonMinF5?}\,(u g_s, S)$ and $!\,\textsc{NonMinF5?}\,(v g_k, S)\big)$ **then**
  $P \leftarrow P \cup \{(u g_s, v g_k)\}$
while $(P \neq \emptyset)$ **do**
 $P' \leftarrow \textsc{Select}(P)$ (critical pairs of minimal degree)
 $d \leftarrow$ degree of critical pairs in P'
 if $(d \leq d_{\mathrm{SB}})$ **then**
  while $(P' \neq \emptyset)$ **do**
   Choose (uf, vg) from P' with $\max_{\prec}\{u\,\mathrm{sig}(f), v\,\mathrm{sig}(g)\}$ minimal w.r.t. $\prec$.
   if $\big(!\,\textsc{RewriteF5?}(u, f, R)$ and $!\,\textsc{RewriteF5?}(v, g, R)\big)$ **then**
    $P' \leftarrow P' \setminus \{(uf, vg)\}$
    $l \leftarrow \max_{\prec}\{u\,\mathrm{sig}(f), v\,\mathrm{sig}(g)\}$
    $R[i] \leftarrow \mathrm{addRule}(l, R[i])$
    $r \leftarrow \big(l, u\,\mathrm{poly}(f) - v\,\mathrm{poly}(g)\big)$
    $\mathrm{poly}(r) \leftarrow \textsc{Reduce}\,\big(\mathrm{poly}(r), \mathrm{poly}(G_{i-1})\big)$
    $(r, P') \leftarrow \textsc{SigRedF5+}(r, G, S, R, s, P')$
    if $\big(\mathrm{poly}(r) \neq 0$ and r not sig–redundant w.r.t. $G\big)$ **then**
     $S[i] \leftarrow \mathrm{addF5Crit}(\mathrm{lt}(r), S[i])$
     for $(k = 1, \dots, t)$ **do**
      if $(\mathrm{lm}(g_k) \nmid \mathrm{lm}(r))$ **then**
       $u \leftarrow \mathrm{lc}(g_k) \frac{\tau(r, g_k)}{\mathrm{lm}(r)}$
       $v \leftarrow \mathrm{lc}(r) \frac{\tau(r, g_k)}{\mathrm{lm}(g_k)}$
       if (r and g_k not redundant) **then**
        $d_{\mathrm{SB}} = \max\{d_{\mathrm{SB}}, \deg(u g_k)\}$
       if $\big(\mathrm{lm}(u)\,\mathrm{siglm}(r) \neq \mathrm{lm}(v)\,\mathrm{siglm}(g_k)\big)$ **then**
        if $\big(!\,\textsc{NonMinF5?}\,(ur, S)$ and $!\,\textsc{NonMinF5?}\,(v g_k, S)\big)$ **then**
         $P \leftarrow P \cup \{(ur, v g_k)\}$
     $t \leftarrow t + 1$
     $g_t \leftarrow r$
     $G \leftarrow G \cup \{g_t\}$
 else
  break
return (G, S, R)

Algorithm 58 F5's semi–complete sig–safe reduction algorithm (SigRedF5+)

Input: f a labeled polynomial, $G = \{g_1, \ldots, g_t\}$ a finite set of labeled polynomials, S a list of lists of terms in $\mathcal{P}$, R a list of lists of terms in $\mathcal{P}$, s the index of the first labeled polynomial of current index, P' a set of critical pairs

Output: h a labeled polynomial sig–safe reduced w.r.t. G, P' a set of critical pairs

1: $b \leftarrow 0$
2: $D \leftarrow \{g_s, \ldots, g_t\}$
3: $B_{i-1} \leftarrow \{\mathrm{poly}(g_1), \ldots, \mathrm{poly}(g_{s-1})\}$
4: $l \leftarrow \mathrm{siglm}(f)$
5: $p \leftarrow \mathrm{poly}(f)$
6: **while** $\left(p \neq 0 \text{ and } D_p \leftarrow \left\{g \in D \mid \mathrm{lm}\left(\mathrm{poly}(g)\right) \mid \mathrm{lm}(p)\right\} \neq \emptyset\right)$ **do**
7:   Choose any $g \in D_p$.
8:   $u \leftarrow \frac{\mathrm{lt}(p)}{\mathrm{lt}(\mathrm{poly}(g))}$
9:   **if** $(!\,\textsc{NonMinF5?}(ug, S)$ and $!\,\textsc{RewriteF5?}(u, g, R))$ **then**
10:     **if** $\left(\mathrm{lm}(u)\,\mathrm{siglm}(g) < l\right)$ **then**
11:       $q \leftarrow \textsc{Reduce}\left(u\,\mathrm{poly}(g), B_{i-1}\right)$
12:       $p \leftarrow p - q$
13:       $b \leftarrow 0$
14:     **else if** $\left(\mathrm{lm}(u)\,\mathrm{siglm}(g) > l\right)$ **then**
15:       $P' \leftarrow P' \cup \{(ug, (\mathrm{sig}(f), p, b))\}$
16:       $b \leftarrow 1$
17:   **else**
18:     $b \leftarrow 1$
19: $h \leftarrow \left(\mathrm{sig}(f), p, b\right)$
20: **return** (h, P')

Theorem 6.5.19. *For any given finite input $F = \{f_1, \ldots, f_r\} \subset \mathcal{P}$ of homogeneous polynomials,* F5+ *terminates after finitely many steps.*

Proof. We need to show that a call of IncF5+ performs only finitely many steps until it terminates.
We first claim that after generating new critical pairs for P IncF5+ satisfies $\#(P) < \infty$ throughout the algorithm's ongoing. To prove this, we show that at any given degree d the algorithm generates only finitely many labeled polynomials and critical pairs. We proceed by induction on d: Certainly, at the beginning of IncF5+, when initializing the first bunch of critical pairs, $\#(P) \leq s - 1 < \infty$. So it follows that $\#(P') < \infty$ for the first call in Line 14. Some of the critical pairs in P' might be rejected by F5's criteria, others generate s–vectors, which are reduced first w.r.t. G_{i-1}, then sig–safe in SigRedF5+. In SigRedF5+ 3 different cases must be distinguished:

(1) Clearly, the case where $\mathrm{poly}(r) = 0$ after calling SigRedF5+ is trivial.

(2) If no reducer of p is found, h is returned and added to G. All new critical pairs have a higher degree, so they are not added to P'. Moreover, only finitely many new pairs are generated due to the fact that $\#(G) < \infty$.

(3) If a reducer ug of p is found and not detected by any of F5's criteria two situations are possible:

a) If $u\,\mathrm{siglm}(g) < l$, then the reduction $p := p - u\,\mathrm{poly}(g)$ takes place. p is again checked for reducers, but $\mathrm{lt}(p)$ has decreased in this step.

b) If $u\,\mathrm{siglm}(g) > l$, then the reduction does not take place directly, but the new critical pair $(ug, (\mathrm{sig}(f), p, b))$ is added to P'. In this case p is kept for further reduction checks. Note that only finitely many such reducers ug could lead to new critical pairs. As $u\,\mathrm{slm}(g)$ is added to the rules list R when this new critical pair is further processed in IncF5+. So the Rewritten Criterion implies that ug is not chosen again as a reducer in the following. At degree d there are only finitely many different signatures of current index, so only finitely many new elements can be added in this way.

Thus only finitely many new labeled polynomials can be generated until $P' = \emptyset$. All in all it follows that the number of labeled polynomials as well as the number of critical pairs is finite throughout IncF5+'s computations.

To finish the proof we have to show that after finitely many steps, only F5–critical pairs are left in P. Generating labeled polynomials of SB–critical pairs have to be not redundant. Since $\mathcal{P}$ is Noetherian, there can only be finitely many not redundant labeled polynomials. It follows that also the number of SB–critical pairs is finite.

Thus F5+ terminates after finitely many steps. □

From a computational point of view it is clear that F5 and F5+ have the very same performance. Computing the degrees of the critical pairs when generating must be done either way, so the only real overhead introduced by F5+ is the following:

▷ Compare d_{SB} with the degree d of the actual SB–critical pairs to be generated.

▷ Keep the redundancy flag correct during the reduction steps in SigRedF5+.

Clearly, thinking about the overall computations done during a run of F5 resp. F5+ this does not make any difference at all. Thus we do not present a table with timings here, since those would be equal regardless of the example. The only real worthwhile information F5+ carries is the value of d_{SB}. It turns out (see Table 6.7) that the values of d_{SB} are mostly too high, i.e. F5 terminates at a degree d which is smaller than d_{SB} in nearly all examples we have checked. So we can summarize our outcomes of F5+ thusly:

▷ F5+ is a variant of F5, whose termination is ensured. This property is obtained by collecting data computed in F5 either way, no outer algorithm or source needs to be used to get this information.

▷ F5+ does not introduce any penalty on performance, the overhead is minimal.

▷ Having already a working implementation of F5, transforming it to F5+ is done adding very few lines of easy code.

The point is that one can see that the degree bound predicted in F5B, i.e. by Buchberger's 2nd Criterion, is way better than d_{SB}. So the question arises if one can combine the ideas of F5+ and F5B to get an algorithm, which is possibly even faster than F5.

We want to achieve a lower degree bound for F5, without introducing computational overhead by bookkeeping another set of critical pairs during the whole computation as it is done in F5B. To reduce the degree we can use the following idea: Instead of covering the maximal degree of all SB–critical pairs handled in F5, we store only the degree d_{F}, which is the degree of all SB–critical pairs not detected by the F5 Criterion. It is clear that $d_{\mathrm{F}} \leq d_{\mathrm{SB}}$. The problem is that we do not know if d_{F} is too low (see Conjecture 6.5.23 at the end of this section). Situation 6.5.14 implies that this choice of a degree bound might be incorrect. In this situation we drop in the idea F5B is based on: Perhaps some of the critical pairs of degrees $> d_{\mathrm{F}}$ are needed for the correctness of the standard basis, but we cannot see this with F5's criteria. Thus we use Buchberger's 2nd Criterion once the algorithm exceeds degree d_{F} and check the remaining critical pairs. If Buchberger's 2nd Criterion verifies that all those critical pairs are not needed for the standard basis, we terminate the algorithm. Otherwise we go on with the next degree step.

This differs in two important ways from F5B's approach:

(1) Rather than checking all critical pairs with Buchberger's 2nd Criterion, it checks only SB–critical pairs that F5 also rejects as unnecessary. After all, it follows from Theorem 6.5.12 that F5–critical pairs can be necessary *only if* they substitue for an SB–critical pair.

(2) It checks the SB–critical pairs only once F5's criteria suggest that it should terminate.

This leads to a way lower overhead in computational time as well as memory consumption. We illustrate the implementation of this attempt in the pseudo code of Algorithm 59.

The main changes to Algorithm 57 are:

(1) The degree bound d_{F} is recomputed only if the corresponding SB–critical pair is not detected by the F5 Criterion (Lines 11 and 38).

(2) If a critical pair is rejected by the F5 Criterion, it is added to a second set of critical pairs, $\hat{P}$ (Lines 15 and 42).

(3) Whenever the computations exceed the degree d_{F} the critical pairs of the current degree in $\hat{P}$ are checked by Buchberger's 2nd Criterion. Only if all elements of $\hat{P}$ are rejected by Buchberger's 2nd Criterion, the algorithm terminates. Otherwise the computations go on (Line 19).

Definition 6.5.20. We denote the variant of F5 calling IncF5B+, and SigRedF5+ by F5B+.

Whereas termination of F5B+ is a trivial corollary of Theorem 6.5.19, also its correctness can be seen easily.

Theorem 6.5.21. *Let $F = \{f_1, \ldots, f_r\} \subset \mathcal{P}$ be a finite set of homogeneous polynomials, the input of* F5B+. *If* F5B+ *terminates, then its result is a standard basis for $I = \langle f_1, \ldots, f_r \rangle$.*

Proof. The statement follows from Buchberger's 2nd Criterion, Lemma 2.3.4. ☐

Figure 6.5.1: Coloration of the results for termination variants of F5

We implemented the presented variants of F5 in the Singular kernel to compare performance. The code is open source and publicly available at

`git@github.com:ederc/Sources.git`[10].

As already mentioned in Section 5.6 we use Singular 3–1–3, revision 14,372 in the SVN trunk available at

`http://www.singular.uni-kl.de/svn/trunk/`.

In Table 6.7 we compare timings and degree bounds for some examples. All systems are homogeneous and computed over a field of characteristic 32003. The random systems are generated using the function `sparseHomogIdeal` from `random.lib` in Singular; generating polynomials with a sparsity of 85 – 90% and degrees ≤ 6.

All examples where computed on a computer with the following specifications:

▷ 2.6.31–gentoo–r6 GNU/Linux 64–bit operating system,

▷ INTEL® XEON® X5460 @ 3.16GHz processor,

▷ 64 GB of RAM, and

▷ 120 GB of swap space.

Remark 6.5.22. Note that due to the decisions made in F5t to start at some point computing new elements without any criteria checks at all, it is clear that the timings of F5t are much worse than those of F5 resp. F5+. As an implementation of F5t needs lots of costumizations in an existing F5, we have abandoned to do so and do not add any computational results of F5t to Table 6.7.

As in sections 5.6 and 6.4 we use colors to classify the timings. This time we compare, similar to Section 6.4 only 3 different algorithms, Figure 6.5 illustrates the coloration.

[10] You can get the git repository by typing `git clone git@github.com:ederc/Sources.git`.

Algorithm 59 Termination ensured incremental F5 step (IncF5B+)

Input: f_i a polynomial, $G_{i-1} = \{g_1, \ldots, g_{s-1}\}$ a set of labeled polynomials such that $\operatorname{poly}(G_{i-1})$ is a standard basis for $\langle f_1, \ldots, f_{i-1}\rangle$, S a list of lists of terms in $\mathcal{P}$, R a list of $(i-1)$ lists of terms in $\mathcal{P}$

Output: G a set of labeled polynomials such that $\operatorname{poly}(G)$ is a standard basis for $\langle f_1, \ldots, f_i\rangle$, S a list of i lists of terms in $\mathcal{P}$, R a list of i lists of terms in $\mathcal{P}$

$B \leftarrow \varnothing, G \leftarrow \varnothing, P \leftarrow \varnothing, P' \leftarrow \varnothing, \hat{P} \leftarrow \varnothing$, $R[i] \leftarrow$ empty list, $S[i] \leftarrow$ empty list, $d_F \leftarrow 0, d \leftarrow 0$
$t \leftarrow s$
$g_s \leftarrow (e_i, f_i, 0)$
$S[i] \leftarrow \operatorname{addF5Crit}(\operatorname{lt}(g_s), S[i])$
$G \leftarrow \{g_1, \ldots, g_s\}$
for $(k = 1, \ldots, s-1)$ **do**
  $u \leftarrow \operatorname{lc}(g_k) \frac{\tau(g_s, g_k)}{\operatorname{lm}(g_s)}$
  $v \leftarrow \operatorname{lc}(g_s) \frac{\tau(g_s, g_k)}{\operatorname{lm}(g_k)}$
  if $(!\,$NonMinF5?(ug_s, S) and $!\,$NonMinF5?$(vg_k, S))$ **then**
    if (g_k not redundant) **then**
      $d_F = \max\{d_F, \deg(ug_k)\}$
      $P \leftarrow P \cup \{(ug_s, vg_k)\}$
  else
    if (g_k not redundant) **then**
      $\hat{P} \leftarrow \hat{P} \cup (ug_s, vg_k)$
while $(P \neq \varnothing)$ **do**
  $P' \leftarrow$ Select(P) (critical pairs of minimal degree)
  $d \leftarrow$ degree of critical pairs in P'
  if $\big((d \leq d_F)$ or $(\exists p \in \hat{P}$ not satifsying Buchberger's 2nd Criterion$)\big)$ **then**
    while $(P' \neq \varnothing)$ **do**
      Choose (uf, vg) from P' with $\max_{\prec}\{u \operatorname{sig}(f), v \operatorname{sig}(g)\}$ minimal w.r.t. $\prec$.
      if $(!\,$RewriteF5?(u, f, R) and $!\,$RewriteF5?$(v, g, R))$ **then**
        $P' \leftarrow P' \setminus \{(uf, vg)\}$
        $l \leftarrow \max_{\prec}\{u \operatorname{sig}(f), v \operatorname{sig}(g)\}$
        $R[i] \leftarrow \operatorname{addRule}(l, R[i])$
        $r \leftarrow (l, u \operatorname{poly}(f) - v \operatorname{poly}(g))$
        $\operatorname{poly}(r) \leftarrow$ Reduce$(\operatorname{poly}(r), \operatorname{poly}(G_{i-1}))$
        $(r, P') \leftarrow$ SigRedF5+(r, G, S, R, s, P')
        if $(\operatorname{poly}(r) \neq 0$ and r not sig-redundant w.r.t. $G)$ **then**
          $S[i] \leftarrow \operatorname{addF5Crit}(\operatorname{lt}(r), S[i])$
          for $(k = 1, \ldots, t)$ **do**
            if $(\operatorname{lm}(g_k) \nmid \operatorname{lm}(r))$ **then**
              $u \leftarrow \operatorname{lc}(g_k) \frac{\tau(r, g_k)}{\operatorname{lm}(r)}$
              $v \leftarrow \operatorname{lc}(r) \frac{\tau(r, g_k)}{\operatorname{lm}(g_k)}$
              if $(\operatorname{lm}(u) \operatorname{siglm}(r) \neq \operatorname{lm}(v) \operatorname{siglm}(g_k))$ **then**
                if $(!\,$NonMinF5?(ur, S) and $!\,$NonMinF5?$(vg_k, S))$ **then**
                  if (r and g_k not redundant) **then**
                    $d_F = \max\{d_F, \deg(ug_k)\}$
                    $P \leftarrow P \cup \{(ur, vg_k)\}$
                else
                  if (r and g_k not redundant) **then**
                    $\hat{P} \leftarrow \hat{P} \cup (ur, vg_k)$
          $t \leftarrow t + 1$
          $g_t \leftarrow r$
          $G \leftarrow G \cup \{g_t\}$
  else
    break
return (G, S, R)

Let us give a short overview of the values and results presented in Table 6.7:

(1) The notation (a, b, c) denotes a random system of a generators with maximal degree b in a polynomial ring of c variables generated with `random.lib` in SINGULAR .

(2) $d_{\max}$ denotes the maximal degree in the resulting standard basis.

(3) d_{F5} denotes the observed degree of termination of F5.

(4) d_{SB} denotes the maximal degree of the SB–critical pairs taken into account in F5.

(5) d_{B} denotes the maximal degree estimated by Buchberger's 2nd Criterion

(6) d_{F} denotes the maximal degree of all SB–critical pairs not detected by the F5 Criterion.

(7) d_{FR} denotes the maximal degree of all SB–critical pairs not detected by the F5 Criterion *or* the Rewritten Criterion.

In this series of tests we always compute in the respective polynomial ring over a field of characteristic $32,003$ using the graded reverse lexicographical order $<_{\mathrm{dp}}$.

Table 6.7 shows that the tests for F5B+ do not slow it down significantly. But this is expected, since the modifications add trivial overhead, and rely primarily on information that the algorithm already has available.

The computed degrees in Table 6.7 bear some discussion. We have implemented F5B+ in two different ways. Both are the same in that they estimate the maximum necessary degree by counting the maximal degree d_{F} of SB–critical pairs not discarded by the F5 Criterion. However, one can implement a slightly more efficient F5B+ Algorithm by counting the maximal degree d_{FR} only of those SB–critical pairs that pass the F5 Criterion *and* the Rewritten Criterion. We denote the degree where the original F5 terminates by d_{F5}, and the maximal degree of a polynomial generated by d_{maxGB}. Recall also that the maximal degree estimated by F5B is d_B.

It is always the case that $d_{\mathrm{maxGB}} \leq d_{\mathrm{F5}}$; indeed, we will have $d_{\mathrm{maxGB}} \leq d_A$ for any algorithm A that computes a standard basis for a homogeneous system incrementally by degree.

On the other hand, it is always the case that $\max\{d_{\mathrm{F}}, d_{\mathrm{FR}}\} \leq d_{\mathrm{F5}}$; d_{F5} counts F5–critical pairs as well as SB–critical pairs, whereas $d_{\mathrm{F}}, d_{\mathrm{FR}}$ count only SB–critical pairs that are not rejected by one or both of F5's criteria. Thus F5B+ always starts its manual check for termination no later than F5 would terminate, and sometimes terminates before F5. For example, the termination mechanisms activate for `F-855-h`, `Eco-10-h`, `Eco-11-h`, and `Cyclic-8-h`, so F5B and F5B+ both terminate at lower degree than F5. With little to no penalty, F5B+ terminates first, but F5B terminates *well after* F5 in spite of the lower degree! Even in `Katsura-9-h` and `Katsura-10-h`, where $d_{\mathrm{maxGB}} = d_B < d_{\mathrm{F}} = d_{\mathrm{FR}} = d_{\mathrm{F5}}$, the termination mechanism of F5B+ incurs almost no penalty, so its timings are equivalent to those of F5, whereas F5B is slower. In other examples, such as `Cyclic-7-h` and (4,5,12), F5 and (therefore) F5B+ terminate at or a little after the degree(s) predicted by d_{F} and d_{FR}, but *before* reaching the maximal degree computed by d_B.

We finish this section with a conjecture about the degree bound needed in F5:

Conjecture 6.5.23. *The* F5 *Algorithm can terminate once the maximal degree of all SB–critical pairs not detected by any of* F5*'s criteria is exceeded.*

Note that this conjecture is not a corollary of Theorem 6.5.19. The conjecture implies to drop the check with Buchberger's 2nd Criterion in an implementation of F5B+. Proving this conjecture could give a way lower degree bound and improve timings of F5 a lot.

Let us close with the following remark:

Remark 6.5.24. The changes that need to be done to get F5+ or F5B+ from F5 can be applied to F5C or F5E without any modification. Thus algorithms like F5C+ and F5EB+ are clear from a theoretical point of view. We abandon to give extensive pseudo codes due to the clarity of how to achieve these variants.

Examples	F5	F5B	F5B+	F5/F5B	F5/F5B+	$d_{\max}$	d_{F5}	d_{SB}	d_{B}	d_F	d_{FR}
Katsura-9-h	39.951	53.973	40.231	0.74	0.99	13	16	21	13	16	16
Katsura-10-h	1,145.473	1,407.927	1,136.437	0.80	1.00	15	18	26	15	18	18
F-855-h	9,831.814	11,364.473	9,793.178	0.86	1.00	14	18	20	17	17	16
Eco-10-h	47.266	57.975	46.671	0.82	1.01	15	20	23	17	17	17
Eco-11-h	1,117.139	1,368.448	1,072.472	0.82	1.04	17	23	26	19	19	19
Cyclic-7-h	6.243	9.182	6.217	0.67	1.00	19	23	28	24	23	21
Cyclic-8-h	3,791.548	4,897.631	3,772.668	0.77	1.00	29	34	41	33	32	30
(4,6,8)	195.455	204.887	195.691	0.95	1.00	22	36	42	34	34	34
(5,4,8)	45.103	46.930	45.123	0.96	1.00	20	22	35	23	20	20
(6,4,8)	46.180	46.880	46.247	0.99	1.00	20	20	34	22	20	20
(7,4,8)	0.827	0.780	0.830	1.06	1.00	14	19	27	14	17	15
(8,3,8)	122.972	126.816	123.000	0.97	1.00	22	37	35	26	31	29
(4,5,12)	4.498	5.680	4.590	0.79	0.98	29	33	37	42	32	30
(6,5,12)	12.071	21.150	12.060	0.57	1.00	50	54	73	55	54	50
(8,4,12)	46.122	47.613	47.750	0.97	0.97	27	35	44	30	34	29
(12,4,12)	14.413	14.897	14.360	0.97	1.00	42	55	60	43	53	43
(4,3,16)	1.439	1.403	1.450	1.03	0.99	15	15	23	18	15	15
(6,3,16)	36.300	37.136	36.300	0.98	1.00	10	14	23	15	14	13
(8,3,16)	467.560	471.737	467.530	0.99	1.00	12	16	21	13	15	13
(12,3,16)	210.327	206.441	210.311	1.02	1.00	21	25	34	20	24	23
(4,3,20)	1.512	1.680	1.500	0.90	1.01	16	22	24	22	21	21
(6,4,20)	1,142.433	1,327.540	1,144.370	0.86	1.00	27	37	39	29	35	31
(8,4,20)	8.242	8.230	8.251	1.00	1.00	35	40	48	36	40	37
(12,3,20)	0.650	0.693	0.650	0.94	1.00	22	26	34	27	26	23
(16,3,20)	2.054	2.060	2.050	1.00	1.00	26	26	41	27	26	26

Table 6.7: Timings (in seconds) & degrees of F5, F5B, and F5B+

7 Generalizing signature–based algorithms

In this chapter we give some new approaches for using the ideas of signature–based algorithms in a more general setting. The chapter is a mixture of various topics in the area of standard basis computations. It should be understood and read in three different views:

(1) This chapter presents results of current research. All of them are not published anywhere else and unique to this publication.

(2) Due to the first point, some of the topics covered are still in the process of being investigated and further developed, not only by the author, but also lots of other people in the computer algebra community. In some sections we can only present first results and give some discussion on future research.

(3) Other sections contain completey new results which are proven there in complete, but which still lack implementation. This is due to different reasons, e.g. the complexity of the implementation which needs more time, or the problem of not avail-

able features in SINGULAR , like a thread–safe memory manager, whose implementation must be done first.

In Section 7.1 we show how to generalize all signature–based algorithms to be capable of inhomogeneous input, giving an in–depth discussion on selection strategies for critical pairs and problems with the sig–safe reduction. Efficiently computing the ideal quotient $\langle f_1, \ldots, f_k \rangle : f_{k+1}$ is a property which is known for G2V. In Section 7.2 we do not only explain how this is done, but show that all signature–based algorithms are capable of this. Thereby we give a complete new proof of this feature, improving the computation of ideal quotients even more. Following this we explain how to generalize the notion of a signature which turns out to have various applications for standard basis computations, for example in non–incremental signature–based algorithms (see Section 7.4) or parallelization of those (see Section 7.5). We finish this chapter with a new theoretical result, which uses signature–based algorithms for the computation of standard bases for corresponding first modules of syzygies in Section 7.6.

All in all this chapter gives a nice insight in what can be expected from the signature–based world in the near future, not only in terms of optimizations and improvements, but above all speaking about generalizing the algorithms to a wider field of systems they are usable on.

7.1 Signature–based algorithms and inhomogeneous input

In [63] Faugère restricted F5 to work on homogeneous ideals in $\mathcal{P}$ only. Clearly, whenever one wants to compute a standard basis for an inhomogeneous ideal the ideas of Section 2.2 can be used:

(1) Homogenize the inhomogeneous ideal I w.r.t. some new variable.

(2) Compute the standard basis G^{h} for this homogeneous ideal I^{h}.

(3) Cut down G^{h} to a standard basis G for I.

The problem of this approach is that computing a standard basis for I^{h} can be much harder than the computations in the inhomogeneous case.[1] Thus it is desired to compute standard bases for inhomogeneous input with signature–based algorithms, too.

In Chapter 5 we have not restricted our discussion to the homogeneous case. All algorithms presented in that chapter can be used in the inhomogeneous setting, too. This we have also seen by the results of example computations given in Section 5.6. The great fact is that also F5 can compute standard bases for inhomogeneous input! After a discussion on

[1] In some cases it is even not possible to compute a standard basis in the homogenized setting.

how F5 must be changed to achieve this property we give a short summary on the general problems of signature–based algorithms and inhomogeneous input. This is a field of active research these days.

What is really needed for the correctness of any signature–based algorithm? The question is pretty easy, one must compute all critical pairs by increasing signatures, otherwise criteria checks may corrupt data, and wrong pairs are marked to be useless. So whereas this is ensured in SigStd and all its variants due to the fact we always take the next element out of the pair set P with lowest possible signature, we must be more careful in F5: A closer look to Algorithm IncF5 (and its optimized variants) shows that F5 presorts a bunch of elements of P in a second pair set P' which consists of all critical pairs of minimal possible degree, which are not detected to be useless by the F5 Criterion. In P' then the element of lowest possible signature is chosen. Two questions arise from this investigation:

(1) Does F5 compute new elements by increasing signature throughout the algorithm's working?

(2) If the answer to the first question is positive, does this also hold using inhomogeneous input?

To answer the first question we need to find a connection between the degree of a labeled polynomial (i.e. the degree of the polynomial part of it) and its signature. For this we assume homogeneous polynomial data as input of F5, and $<$ to be a well–order on $\mathcal{P}$. In Section 2.2 we have already seen the following nice property of homogeneous polynomials when constructing s–vectors:

Let f and g be two homogeneous polynomials in $\mathcal{P}$. Computing corresponding multiples u and v such that $u\,\mathrm{lt}(f) = v\,\mathrm{lt}(g)$ we can construct their s–vector $uf - vg$. It clearly holds that $\deg(u\,\mathrm{lt}(f)) = \deg(v\,\mathrm{lt}(g))$. As both f and g are homogeneous, uf and vg are homogeneous, too. Thus for all terms $t \in \mathrm{supp}(f)$ it holds that

$$\deg(ut) = \deg(u\,\mathrm{lt}(f)) = \deg(u) + \deg(f).$$

A similar statement holds for the terms in $\mathrm{supp}(g)$.

With this in mind let us see how the signature and the degree of a labeled polynomial, with homogeneous polynomial part, computed in F5 are related to each other. For this let us assume the ith call of IncF5, i.e. there exists a module morphism

$$\begin{array}{rccc} \pi: & \mathcal{P}^i & \to & \langle f_1, \ldots, f_i \rangle, \\ & e_k & \mapsto & f_k \end{array}$$

for all $1 \leq k \leq i$. Let us have a closer look at the labeled polynomials computed during the actual call of IncF5.

▷ In the beginning a first current index labeled polynomial is initialized, $g_s = (e_i, f_i)$. In this situation we know that

$$\text{sig-deg}(g_s) = \deg(g_s).$$

Note that this even holds if $\mathrm{poly}(g_s)$ is inhomogeneous.

▷ The first critical pairs are generated by g_s and elements g_k of lower index in G. Let u and v be the corresponding multipliers such that $u\,\mathrm{lt}(g_s) = v\,\mathrm{lt}(g_k)$. Since g_s and g_k are both homogeneous the degree of the critical pair (ug_s, vg_k) and $ug_s - vg_k$ is the same. Moreover, reductions by homogeneous elements do not change the degree of the s–vector. So whenever SigRedF5 returns a labeled polynomial r with $\mathrm{poly}(r) \neq 0$ $\deg(r) = \deg(ug_s - vg_k) = \deg(ug_s)$. On the other hand, we know that $\mathrm{sig}(r) = ue_i$; in other words

$$\text{sig-deg}(r) = \deg\left(u\pi(e_i)\right) = \deg(ug_s) = \deg(r).$$

Thus all labeled polynomials f constructed in this way fulfill $\text{sig-deg}(f) = \deg(f)$.

▷ Labeled polynomials derived from critical pairs (uf, vg) generated by elements f and g of current index i are left to be investigated. By the above discussion we can assume that

$$\text{sig-deg}(f) = \deg(f) \quad \text{and} \quad \text{sig-deg}(g) = \deg(g).$$

W.l.o.g. let $u\,\mathrm{sig}(f) > v\,\mathrm{sig}(g)$. As $\deg(r) = \deg(uf - vg) = \deg(uf)$ for the corresponding reduced labeled polynomial r after SigRedF5 (assuming that $\mathrm{poly}(r) \neq 0$, the other case is trivial), we see that also in this situation equality of the polynomial degree and the signature degree holds:

$$\text{sig-deg}(r) = \text{sig-deg}(uf) = \deg(u) + \text{sig-deg}(f) = \deg(u) + \deg(f) = \deg(r).$$

We see that, assuming homogeneous input of a signature–based standard basis algorithms, it is useless to presort the pair set P by increasing degree of the critical pairs and later on sort the part P' of pairs of minimal degree by the signature: The signature of any s–vector corresponding to a critical pair in P' is smaller than the signature of an s–vector generated out of a pair from P.

It follows that we can remove this presorting in IncF5 without changing any computational step of F5 at all! This means that IncF5 can use exactly the same `while` loop as IncSigCrit and differences between F5 and SigStd vanish more and more. We waive stating the updated pseudo code of IncF5 implementing this change since it is trivial.

Next we need to look at the above discussed degree connections, this time under the assumption that the underlying polynomial data is *not homogeneous*. In this situation the connection between sig-deg and deg becomes more complicated:

▷ Clearly, for the initial labeled polynomial of the ith iteration step of F5 nothing changes: $g_s = (e_i, f_i)$ with

$$\text{sig-deg}(g_s) = \deg(f_i) = \deg(g_s).$$

▷ For critical pairs (ug_s, vg_k) generated by g_s and lower index element $g_k \in G$ it still holds that the degree of the critical pair, $\deg\left(\tau(g_s, g_k)\right)$, is equal to $\text{sig-deg}(ug_s)$. However, computing the s–vector and reducing it even further the degree can drop.

So the reduced labeled polynomial r in the end only fulfills the much weaker inequality

$$\deg(r) \leq \text{sig-deg}(r) = \text{sig-deg}(ug_s) = \deg(ug_s) = \deg\left(\tau(g_s, g_k)\right).$$

So from this point on for all current index labeled polynomials g, besides the initial one g_s, the degree of the polynomial part of g can be smaller than the degree of its signature.

▷ This leads to the problem that at the moment we generate critical pairs (uf, vg) of labeled polynomials computed during the current iteration step, again assuming $u\,\mathrm{sig}(f) > v\,\mathrm{sig}(g)$,

$$\deg\left(\tau(f, g)\right) < \text{sig-deg}(uf)$$

is possible. It is even way more likely than having an equality of those degrees. From this point onwards it is clear that there is for any current index labeled polynomial h no dependency between $\deg(h)$ and sig-deg(h) besides $\deg(h) \leq$ sig-deg(h).

From this we see that whereas it is still safe to compute by increasing signature in SigStd and its variants, F5's attempt to presort by the degree of the critical pairs can cause problems. Think about the following quite likely situation (e.g. in `Eco-11` such a situation happens hundreds of times):

Let (uf, vg) and $(u'f', v'g')$ be two critical pairs in P, again assuming that $u\,\mathrm{sig}(f) > v\,\mathrm{sig}(g)$ and $u'\,\mathrm{sig}(f') > v'\,\mathrm{sig}(g')$. Moreover, assume that $\deg(uf) < \deg(u'f')$. In F5 this means that once IncF5 has processed all critical pairs of degree $< \deg(uf)$, (uf, vg) is added to P', whereas $(u'f', v'g')$ stays in P and its further computation is postponed to a later point. In this constellation it is still possible that $u\,\mathrm{sig}(f) > u'\,\mathrm{sig}(f')$, but this would mean that an element of higher signature is computed *before* an element of lower signature. Doing computations by increasing signatures is of supreme importance in the signature–based world, all proofs of correctness and even termination of the different algorithms are based on this fact!

So it is not possible to ensure correctness and / or termination of a signature–based algorithm, given inhomogeneous input, using a degree dependent preselection of critical pairs. F5 is the only signature–based algorithm using this. We have seen that in the case of homogeneous input this preselection is useless and does not change anything w.r.t. to the order in which F5 handles its critical pairs. Thus one should always implement F5 without a degree preselection due to the fact that this method

(1) does not change any computational aspect of F5 in the homogeneous case, and

(2) enables F5 to compute standard bases of inhomogeneous ideals.

Due to the equality between the polynomial degree and the degree of a signature for homogeneous elements those algorithms are designed to handle standard basis computations in this setting very well, discarding lots of useless critical pairs, sorting them by degree (which is a good selection as we have already seen in Section 2.2), and having no real downsides by sig–safe reductions besides the general constraint of sig–safeness, but this is indispensable for the algorithms' correctness.

We have already seen, also in Section 2.2, that one of the best possible choices of critical pairs from the pair set is using the sugar degree (see Definition 2.2.1). The nice fact is that it coincides with the signature degree in signature–based algorithms.

Theorem 7.1.1. *The degree of the signature of a labeled polynomial f, computed in a signature–based standard basis algorithm, coincides to the sugar degree of the polynomial part of f, that is*

$$\text{sig-deg}(f) = \text{s-deg}\left(\text{poly}(f)\right).$$

Proof. Let f, g be two labeled polynomials computed in a signature–based standard basis algorithm interreducing intermediate standard bases. Moreover, let $f_i \in \mathcal{P}$ be the input of the next incremental step of the corresponding algorithm.

(1) For each such f_i it holds that the initial labeled polynomial $g_s = (e_i, f_i)$ fulfills that $\text{sig-deg}(g_s) = \deg(g_s)$.

(2) For any term $t \in \mathcal{P}$ it holds that $\text{sig-deg}(tf) = \deg(t) + \text{sig-deg}(f)$, if the corresponding element is not detected by any (NM) resp. (RW) related criterion.

(3) Let u and v be terms in $\mathcal{P}$ such that $u\,\text{lt}(f) = v\,\text{lt}(g)$.

   a) Assuming that $\text{index}(f) = \text{index}(g)$ it follows that the signature degree of the corresponding s–vector is given by

   $$\text{sig-deg}(uf - vg) = \max\left\{\text{sig-deg}(uf), \text{sig-deg}(vg)\right\}.$$

   W.l.o.g. let $\text{index}(f) > \text{index}(g)$. Then it holds that $\text{sig-deg}(f) \geq \deg(f)$ and $\text{sig-deg}(g) = \deg(g)$. Since $\deg(uf) = \deg(vg)$ it holds that $\text{sig-deg}(uf) \geq \text{sig-deg}(vg)$. It follows that

   $$\text{sig-deg}(uf - vg) = \text{sig-deg}(uf).$$

These are just the properties of the definition of the sugar degree given in Definition 2.2.1. □

This means that any signature–based algorithm, which depends on computing its data by increasing signatures, computes new elements for the standard basis w.r.t. the sugar degree. Thus by default a really good selection strategy is taken in these algorithms.

This discovery seems to be incompatible with the experimental results in Section 5.6. There we have seen that the algorithms have problems computing standard bases of inhomogeneous ideals, e.g. `Eco-11` cannot be computed, whereas `Eco-11-h` is not a problem at all. In Buchberger–like algorithms the problem is just the other way around! Homogenizing ideals and trying to compute a corresponding standard basis can be a much harder problem than the computations in the inhomogeneous setting. So the question arises where exactly the problems of signature–based standard basis algorithms lie w.r.t. inhomogeneous input?

▷ The selection strategy is efficient as we have seen in Theorem 7.1.1.

- ▷ Also the criteria detecting useless critical pairs work quite great in the inhomogeneous setting, discarding a lot more elements than the Gebauer–Möller implementation in most of the examples.
- ▷ So the only situation where problems can occur is the reduction process. In there we can ignore the reducers of lower index, since they are handled without any difference as in a Buchberger–like algorithm. So the reductions with current index labeled polynomials seem to be left as a potential source of trouble.

Let us investigate this case a bit more carefully: Due to the fact that we lose the connection

$$\deg(f) = \text{sig-deg}(f)$$

for all labeled polynomials f computed during an iteration step when switching from homogeneous to inhomogeneous input, forcing the reduction to be sig–safe can have really bad impact on the algorithms behaviour. The problem is that in the given setting reductions are not only sig–unsafe because the signatures have the same degree, but differ. Now it is even possible that a multiplied reducer has a signature of higher degree than the element to be reduced! Thus a lot more reductions do not take place. This again means that a bunch of new critical pairs are generated and tested and computed. But this time the signatures of these critical pairs need not have the same degree. Let us give an example: Assume a labeled polynomial f to be reduced by another labeled polynomial $g \in G$, i.e. there exists a term $u \in \mathcal{P}$ such that $\operatorname{lt}(f) = u \operatorname{lt}(g)$. The reduction itself is not allowed as $u \operatorname{sig}(g) > \operatorname{sig}(f)$. So in the following a new critical pair (ug, f) with signature $u \operatorname{sig}(g)$ is generated and later on computed. The problem is the "later on": Whereas (ug, f) has the same signature degree as f in the homogeneous setting, assuming polynomials to be inhomogeneous it is possible that $\text{sig-deg}(ug) > \text{sig-deg}(f)$. This means that the corresponding data needed from the reduction step of f and ug cannot be used in the algorithm at the time it really is needed. This triggers other reductions that would be helpful to take place at an earlier point of the algorithm to be delayed. All in all, correctness is still ensured, but the overhead that is computed due to all these not allowed and postponed reduction steps has a clear penalty on the performance of signature–based standard basis algorithms.

Using the same setting as in sections 5.6 and 6.4 we compare the corresponding implementations of AP and F5E, where we adjusted the selection strategy of critical pairs in F5E as discussed above to ensure correcntess and termination of the computations for inhomogeneous input. We compare F5E to AP, as AP is the fastest algorithm of the ones presented in Chapter 5 considering inhomogeneous input data.

There are two important observations:

(1) First of all, F5E allocates much less memory than AP. In most examples less critical pairs are considered in F5E than in AP.

(2) On the other hand, AP can compute `Eco-11`, whereas F5E do not terminate on the computer we use for the example sets in this thesis.

The performance differs quite a lot, sometimes F5E is way more efficient than AP, for example for `Cyclic-8`. For the `Eco-n` examples the picture is just the other way around, F5E is always slower than AP.

Test case	Time (sec)	Memory (MB)	Zero reds	Crit. pairs	Red. steps	$\#(G)$
Cyclic-7	6.500	17.531	36	978	83,880	758
Cyclic-8	5,418.410	489.005	244	5,770	3,403,874	3,402
Eco-8	0.390	3.526	0	404	24,887	187
Eco-9	14.970	32.079	0	918	24,7434	373
Eco-10	734.830	242.203	0	2,035	2,384,889	725
Eco-11	–.–	–.–	-	-	-	-
F-633	0.000	0.035	0	54	290	60
F-744	9.540	25.092	0	818	179,100	87
F-855	101.520	149.352	0	2,704	835,718	148
Katsura-8	0.050	1.500	0	120	1,634	105
Katsura-9	0.490	6.000	0	247	5,371	202
Katsura-10	5.890	23.518	0	502	18,343	399
Katsura-11	70.100	92.098	0	1,013	63,194	784

Table 7.1: Computation for inhomogeneous input using F5E

Test case	Time (sec)	Memory (MB)	Zero reds	Crit. pairs	Red. steps	$\#(G)$
Cyclic-7	5.950	637.573	36	914	93,742	658
Cyclic-8	14,078.000	60,444.643	244	20,086	49,444,223	2,611
Eco-8	0.080	12.026	0	398	10,161	187
Eco-9	1.630	90.595	0	954	83,911	373
Eco-10	67.410	666.817	0	2,337	869,101	725
Eco-11-h	4,521.250	26,206.020	502	14,994	7,787,226	1,968
Eco-11	7,692.740	5,345.710	0	228,450	9,623,810	1,455
F-633	0.000	0.535	0	54	319	56
F-744	42.700	133.606	0	899	249,228	87
F-855	182.390	1,228.474	0	3,309	1,749,296	148
Katsura-8	0.050	8.000	0	120	1,626	105
Katsura-9	0.440	41.000	0	247	5,309	202
Katsura-10	5.110	212.546	0	502	17,868	399
Katsura-11	66.590	1,133.735	0	1,013	60,965	784

Table 7.2: Computation for inhomogeneous input using AP

So the only practical conclusion we can get out of this comparison is that there is not *the one* signature–based standard basis algorithm for inhomogeneous computations. Again, more research needs to be done to get more insight in the inner structures and impacts of inhomogeneous data on the algorithms.

To find optimizations or solutions to this problem is a point of the author's current research. We hope to get some more insight in the behaviour of the algorithms in the inhomogeneous case. Right now it is, to the knowledge of the author, an open problem.

Remark 7.1.2.

(1) Clearly, removing the preselection of critical pairs of minimal degree in F5 implies that one needs to change SigRedF5, too, due to the creation of new critical pairs corresponding to sig–unsafe reductions there. It is obvious how to implement this change, thus we do not illustrate this with pseudo code.

(2) Let us also give some note on local monomial orders on $\mathcal{P}$: As local monomial orders are to be taken into account only in the inhomogeneous setting it is clear that before we can even think about how to handle such a setting we need to understand and to improve the inhomogeneous situation w.r.t. global orders first. There is hope that by getting more knowledge of the sig–safe reduction steps in the global inhomogeneous situation counterparts of Mora's normal form algorithm (see Algorithm 3) can be achieved, too.

(3) Note that we know by the above disussion that the resulting data of F5E's computations for the homogeneous examples do not change at all. The order in which the critical pairs are chosen is exactly the same, independent of a pre–selection by degree.

7.2 Computing the ideal quotient

As already mentioned in Section 5.5 signature–based algorithms can be easily modified to compute not only a standard basis for an ideal $I = \langle f_1, \ldots, f_r \rangle$, but also the ideal quotients

$$\langle f_1, \ldots, f_k \rangle : f_{k+1}.$$

This is a generalization of standard basis computations which was first noted for G2V in [76], but can be applied to any signature–based algorithm presented in this thesis. We show how to achieve the ideal quotient from SigStd, applying the generalizations to AP, G2V, and F5 is straightforward. The statements and proofs of this sections are presented for the first time ever in such a generality.

Convention. In the following we denote $I_k = \langle f_1, \ldots, f_k \rangle$ for $k \in \{1, \ldots, r-1\}$.

The main idea for constructing generators for ideal quotients as side products of incremental standard basis computations can be found in the exact sequence given by

$$0 \longrightarrow R/\left(I_k : f_{k+1}\right) \xrightarrow{\phi} R/I_k \xrightarrow{\psi} R/\langle f_1, \ldots, f_{k+1}\rangle \longrightarrow 0.$$

There ϕ is just a multiplication by f_{k+1}, which is injective, and ψ is the canonical homomorphism between R/I_k and $R/\langle f_1, \ldots, f_{k+1}\rangle$. It clearly holds that $\phi\left(R/\left(I_k : f_{k+1}\right)\right) = \ker(\psi)$.

Any signature–based algorithm presented in this thesis with the dynamically updating (NM) criterion, i.e. an NonMin?–like implementation, where a new element $\operatorname{slt}(g)$ is added to the set S of all leading terms of elements of I_k whenever $\operatorname{poly}(g)$ has been reduced to zero in SigRed, can compute a basis for $I_k : f_{k+1}$. In fact, we have the following.

Proposition 7.2.1. *In the setting presented in Chapter 5 S is a basis for the ideal $L(I_k : f_{k+1})$ at the end of each iteration step, i.e. whenever IncSigCrit returns to SigStd.*

Proof. Let us assume that the current index in IncSigCrit is s, i.e. $\pi(e_s) = f_{k+1}$. In other words, $f_{k+1} = p_s$ and $I_k = \langle p_1, \ldots, p_{s-1}\rangle$.

(1) The initial elements of S are already in $L(I_k)$, thus they are clearly in $L(I_k : p_s)$. Any element f computed during IncSigCrit such that SigRed reduces $\operatorname{poly}(f)$ to zero fulfills the following property:

$$\operatorname{label}(f) = \sum_{i=1}^{s} q_i e_i.$$

Since $\operatorname{poly}(f) = \pi\big(\operatorname{label}(f)\big) = 0$ we have that

$$\begin{aligned} \operatorname{poly}(f) &= \sum_{i=1}^{s} q_i p_i = 0 \\ \Rightarrow q_s p_s &= \underbrace{\sum_{i=1}^{s-1} q_i p_i}_{\in I_k}. \end{aligned}$$

This means that $q_s \in I_k : f_{k+1}$, and moreover, $\operatorname{lt}(g_s) \in L(I_k : f_{k+1})$. Thus $S \subset L(I_k : f_{k+1})$.

(2) Assume that there exists an element $g \in I_k : f_{k+1}$, but $\operatorname{lt}(g) \notin \langle S\rangle$ when IncSigCrit stops. This would mean that $\operatorname{lt}(g) = \operatorname{slt}(h)$ of some labeled polynomial h which has not reduced to zero in SigRed w.r.t. G. Then four situations are possible:

a) Either there is no reducer of $\operatorname{poly}(h)$ in $\operatorname{poly}(G)$, neither sig–safe nor sig–unsafe. This means that we get a representation

$$\Rightarrow q_s p_s = \underbrace{\sum_{i=1}^{s-1} q_i p_i}_{\in I_k} + \underbrace{\operatorname{poly}(h)}_{\notin I_k}.$$

But then it would follow that $q_s = g \notin I_k : f_{k+1}$, which is a contradiction to our assumption.

b) It is possible that a sig–unsafe reduction has not taken place, which could have lead to a zero reduction. Then in the next round of critical pair creation h and the corresponding sig–unsafe reducer build a new critical pair. This process of creating new critical pairs (if sig–unsafe reductions take place) goes on until we reach the zero reduction $\mathrm{poly}(h')$. Then g is not in $I_k : f_{k+1}$ by the same argument as in Situation (2)a.

c) The critical pair corresponding to the s–vector h, which would reduce to zero, is detected by NonMin?. But then there exists an element $s \in S$ such that $s \mid \mathrm{slt}(h)$.

d) The critical pair corresponding to the s–vector h, which would reduce to zero, is detected by Rewrite?. Then there either exists a syzygy $l \in \mathcal{P}^s$ such that $\mathrm{slt}(l) = \mathrm{lt}(g)$, i.e. there exists some element $t \in S$ with $t \mid \mathrm{slt}(l)$ due to a previously computed zero reduction. Or there exists another s–vector not reducing to zero, whose leading term of its signature divides $\mathrm{lt}(g)$. But then the corresponding critical pair h has a standard representation w.r.t. an ideal including $p_1, \ldots, p_{s-1}$ and at least $\mathrm{poly}(h)$, whereas $\mathrm{poly}(h) \notin I_k$. Thus also $g \notin I_k : f_{k+1}$.

□

Moreover, we can generalize the labels of labeled polynomials we have restricted ourselves to in practice (see Section 4.3): In theory we always assume a labeled polynomial $f = (l, p)$ such that $\pi(l) = p$. Here we can require that for any labeled polynomial f we store the part of highest index of the corresponding label in l.

Definition 7.2.2.

(1) Assume IncSigCrit with input values $G_{i-1} = \{p_1, \ldots, p_{s-1}\}$, $f_i = p_s$. Let $l \in \mathcal{P}^s$ be a complete label of a labeled polynomial $f = (s, p)$ (whereas we think of l as the label computed during IncSigCrit considering Algorithm 34 for sig–safe reductions); i.e. $\pi(l) = p$. We define the map

$$\begin{array}{rccl} \phi : & \mathcal{P}^s & \longrightarrow & \mathcal{P}^s / \langle e_1, \ldots, e_{s-1} \rangle, \\ & \sum_{i=1}^s p_i e_i & \longmapsto & p_s e_s, \end{array}$$

where the $p_i \in \mathcal{P}$ and the e_i are the canonical generators of $\mathcal{P}^s$. If $u = \phi(l)$, then we call $f = (u, p)$ a *curr–index labeled polynomial*. By definition, a curr–index labeled polynomial $g = (v, q)$ with $\mathrm{index}(g) < s$ has $v = 0$.

(2) For a curr–index labeled polynomial $f = (u, p)$ we can define the *polynomial part of its label u* by $\mathrm{pp}(f) := \mathrm{pp}(u)$ where

$$\begin{array}{rccl} \mathrm{pp} : & \mathcal{P}^s & \longrightarrow & \mathcal{P}, \\ & u = p_s e_s & \longmapsto & p_s, \end{array}$$

Example 7.2.3. Let us reconsider Example 4.3.4. There we compute a slim labeled polynomial $g_4 = (ye_3, -xz^3 + yz^3)$. Considering the whole label during the sig–safe reduction step, i.e. using Algorithm 34, we end up with the full labeled polynomial $g_4 = \big((y+z)e_3 - z^2 e_2 + (yz + z^2)e_1, -xz^3 + yz^3\big)$. In this setting $\phi : \mathcal{P}^3 \to \mathcal{P}^3 / \langle e_1, e_2 \rangle$. Thus we receive a curr–index labeled polynomial $g_4 = \left((y+z)e_3, -xz^2 + yz^3\right)$. Moreover, $\mathrm{pp}(g_4) = y + z$.

In this sense we need to reconsider our initial definition of SigRed given in Algorithm 34: Instead of only reducing the polynomial part, we also need to reduce the label of the curr–index labeled polynomials. This means, whenever we have an element f, and we find a reducer g such that $t = \frac{\mathrm{lt}(f)}{\mathrm{lt}(g)}$ and $\mathrm{sig}(tg) < \mathrm{sig}(f)$, then we need to compute

$$\begin{aligned}\mathrm{poly}(f) &= \mathrm{poly}(f) - t\,\mathrm{poly}(g), \text{ and}\\ \mathrm{label}(f) &= \mathrm{label}(f) - t\,\mathrm{label}(g).\end{aligned}$$

Similar to the above approach computing the leading ideal of $I_k : f_{k+1}$ we would like to have at the end of each iteration step that the elements of S generate $I_k : f_{k+1}$. We illustrate the needed changes in Algorithm 61:

Algorithm 60 SigStd including ideal quotients(SigStdQ)

Input: $F = \{p_1, \ldots, p_r\}$ a finite subset of $\mathcal{P}$
Output: G a standard basis for $\langle F \rangle$ w.r.t. $<$, L a list of ideal quotients

1: $L \leftarrow [\,]$
2: $G_1 \leftarrow \{p_1\}$
3: **for** $(i = 2, \ldots, r)$ **do**
4: $\quad p_i \leftarrow \mathrm{Reduce}(p_i, G_{i-1})$
5: $\quad$ **if** $(p_i \neq 0)$ **then**
6: $\quad\quad (G_i, B_i) \leftarrow \mathrm{IncSigQ}(p_i, G_{i-1})$
7: $\quad\quad L \leftarrow \mathrm{Concat}(L, B_i)$
8: $\quad$ **else**
9: $\quad\quad G_i \leftarrow G_{i-1}$
10: $G \leftarrow G_r$
11: **return** (G, L)

It is important to store not only the leading terms of the elements from G_k, but the whole polynomials in S (see Line 7). The elements in G_k are clearly in I_k and thus also by construction in $I_k : f_{k+1}$.
Whenever a zero reduction of a labeled polynomial $f = (l, p)$ happens, we have seen in our previous discussions (see the comparison of F5 and G2V regarding the usage of the non–minimal signature criterion) that it is advantageous to add $\mathrm{siglt}(f)$ to S. In our special setting here, we need not only to add the term, but the whole polynomial part of the signature, i.e. $\mathrm{pp}(f)$. This has changed, w.r.t. Algorithm 36 in Line 23.
In the end, IncSigQ returns not only the standard basis B of I_k as IncSigCrit, but also S, which stores generators for $I_k : f_{k+1}$.

This leads to a small change in SigStd, namely taking care of this second return value of IncSigQ. In Algorithm 60 we present the generalized variant of SigStd to store the ideal quotients, too. SigStdQ returns not only the standard basis G, but also a list of ideal quotients $I_k : f_{k+1}$ for $k = \{1, \ldots, r-1\}$.

Algorithm 61 IncSigCrit with curr–index labeled polynomials (IncSigQ)

Input: f_i a polynomial, $G_{i-1} = \{p_1, \ldots, p_{s-1}\}$ a standard basis for $\langle f_1, \ldots, f_{i-1}\rangle$
Output: B a standard basis for $\langle f_1, \ldots, f_i\rangle$ w.r.t. $<$

$B \leftarrow \varnothing, G \leftarrow \varnothing, P \leftarrow \varnothing$
$S \leftarrow \varnothing$
$p_s \leftarrow f_i$
$t \leftarrow s$
for $(k = 1, \ldots, s-1)$ **do**
  $g_k \leftarrow (0, p_k)$
  $S \leftarrow S \cup \{p_k\}$
$g_s \leftarrow (e_s, p_s)$
$G \leftarrow \{g_1, \ldots, g_s\}$
for $(k = 1, \ldots, s-1)$ **do**
  $u \leftarrow \operatorname{lc}(g_k) \frac{\tau(g_s, g_k)}{\operatorname{lm}(g_s)}$
  $v \leftarrow \operatorname{lc}(g_s) \frac{\tau(g_s, g_k)}{\operatorname{lm}(g_k)}$
  if $\big($! NonMin? (ug_s, vg_k, S) and ! Rewrite? $(ug_s, vg_k, G, P)\big)$ **then**
    $P \leftarrow P \cup \{(ug_s, vg_k)\}$
while $(P \neq \varnothing)$ **do**
  Choose (uf, vg) from P with $\max_{<} \{u \operatorname{sig}(f), v \operatorname{sig}(g)\}$ minimal w.r.t. $<$.
  if $\big($! NonMin?(uf, vg, S) and ! Rewrite?$(uf, vg, G, P)\big)$ **then**
    $P \leftarrow P \setminus \{(uf, vg)\}$
    $l \leftarrow u \operatorname{label}(f) - v \operatorname{label}(g)$
    $r \leftarrow \big(l, u \operatorname{poly}(f) - v \operatorname{poly}(g)\big)$
    $r \leftarrow$ SigRed(r, G)
    if $(\operatorname{poly}(r) = 0)$ **then**
      $S \leftarrow S \cup \{\operatorname{pp}(r)\}$
    else if $\big(\operatorname{poly}(r) \neq 0$ and r not sig–redundant w.r.t. $G\big)$ **then**
      for $(k = 1, \ldots, t)$ **do**
        $u \leftarrow \operatorname{lc}(g_k) \frac{\tau(r, g_k)}{\operatorname{lm}(r)}$
        $v \leftarrow \operatorname{lc}(r) \frac{\tau(r, g_k)}{\operatorname{lm}(g_k)}$
        if $\big(u \operatorname{siglm}(r) \neq v \operatorname{siglm}(g_k)\big)$ **then**
          if $\big($! NonMin? (ur, vg_k, S) and ! Rewrite? $(ur, vg_k, G, P)\big)$ **then**
            $P \leftarrow P \cup \{(ur, vg_k)\}$
      $t \leftarrow t + 1$
      $g_t \leftarrow r$
      $G \leftarrow G \cup \{g_t\}$
$B \leftarrow \operatorname{poly}(G)$
return (B, S)

It is left to give a statement of our approach:

Corollary 7.2.4. *Let $F = \{f_1, \ldots, f_r\}$, a finite subset in $\mathcal{P}$, be the input of* SigStdQ*, $I = \langle F\rangle$ an ideal. Then the algorithm returns a standard basis for I w.r.t. the underlying order on $\mathcal{P}$*

and a list of generators of the quotient ideals $I_k : f_{k+1}$ for $1 \le k \le r-1$.

Proof. The statement about the algorithmic behaviour of SigStdQ and its returning of a standard basis for the input are clear by our previous discussions. The statement about the list of ideal quotients can be restricted to proving that IncSigQ stores the generators of $I_k : f_{k+1}$ in S. But this is just clear by looking at the pseudo code of IncSigQ and restating the proof of Proposition 7.2.1 without restricting to the leading ideal. □

Remark 7.2.5.

(1) Note that exchanging sparse labeled polynomials by curr–index labeled polynomials has a huge impact on the performance. Not only that the algorithm needs much more memory, the number of computations increases due to the fact that we have to adjust the labels whenever we reduce with an element of current index in SigRed.

(2) Even more, considering NonMin? and Rewrite? the criteria checks get even harder, as we first need to get the leading terms of the labels, which consumes some computational time, too.

7.3 Generalizing signatures

In this section we give the idea of generalizing the signature. This is mainly based on some remark given in [126]. There complete syzygies are used to compute standard bases, which has a bad impact on the performance of their algorithm, since it needs to take care of a lot more data. They suggest to keep only some terms of the syzygies stored and use them to detected useless critical pairs. We have already seen that signature–based algorithms are just a very special implementation of the syzygy idea, using only the leading terms of the corresponding module elements, the signatures. In this setting the idea of generalizing the signatures means to not only take the leading term of a module element into account, but also some more terms. With this one can interreduce some signatures, which could lead to more rules detecting more useless critical pairs. Other ideas consider more flexible ways of reducing sig–safe, which could also be very helpful in the inhomogeneous setting as we have seen in Section 7.1.

Let us start generalizing the definition of a signature given in Definition 4.1.3.

Definition 7.3.1. Let $F = \{f_1, \dots, f_k\}$ be a finite subset in $\mathcal{P}$, $I = \langle F \rangle$ be a finitiely generated ideal in $\mathcal{P}$, and let $e_1, \dots, e_k$ be the canonical generators of $\mathcal{P}^k$ such that

$$\begin{array}{rccl} \pi : & \mathcal{P}^k & \longrightarrow & I \\ & e_i & \longmapsto & f_i \text{ for all } 1 \le i \le k \end{array}$$

is a surjective module homomorphism. Let $<$ be a well–order on $\mathcal{P}^k$, and let $g \in I$, $h \in \mathcal{P}^k$.

(1) We define the *signature of length j of h* recursively by

$$\begin{aligned} \operatorname{sig}(h,1) &:= \operatorname{sig}(h),\\ \operatorname{sig}(h,j) &:= \operatorname{sig}(h,j-1) + \operatorname{lt}_{<}\big(h - \operatorname{sig}(h,j-1)\big). \end{aligned}$$

for $2 \leq j \leq \#\big(\operatorname{supp}(h)\big)$.

(2) The *(minimal) signature of length j of g* is denoted

$$\operatorname{sig}(g,j) := \operatorname{sig}\big(\min_{<} \operatorname{labels}(g), j\big).$$

(3) Moreover, let $r = (l,p) \in \mathcal{P}^k \times \mathcal{P}$ be a labeled polynomial. The *signature of length j of r* is given by

$$\operatorname{sig}(r,j) := \operatorname{sig}(l,j).$$

Convention. In the following we mostly speak of the signature of an element without explicitly noting the length of the signature. The reader may always think of the corresponding signature of length 1 in these situations. Whenever a signature of some length greater than 1 is supposed, we explicitly state the length.

Remark 7.3.2. Note that Definition 4.1.9 of a standard representation of labeled polynomials also makes sense if we are interested in signatures of level $j > 1$: By definition, $\operatorname{sig}(r)$ consists of the leading term of $\operatorname{label}(r)$ w.r.t. $<$. All $j-1$ summands added when constructing $\operatorname{sig}(r,j)$ are smaller than $\operatorname{sig}(r)$. Thus the condition on $\operatorname{sig}(r)$ is enough to define a standard representation of a labeled polynomial w.r.t. some given set G.

Let us give some facts about what needs to be updated in a signature–based standard basis algorithm in order to use generalized signatures of a given length > 1. For this assume that we are using signatures of length $j > 1$ in the following. New implementations must be done whenever a current index reduction of a labeled polynomial takes place. This happens at two points of the incremental step of the algorithm:

(1) Whenever an s–vector is generated out of a critical pair it is no longer sufficient to search for the maximum of the leading terms of the generalized signatures, but one needs to compare all j terms of the corresponding signatures and construct a new signature of length j for the s–vector out of the $2j$ terms in question.

(2) A very similar situation happens whenever a sig–safe reduction $f - ug$ takes place: This time we have already checked that $\operatorname{lt}(\operatorname{sig}(f,j)) > u \operatorname{lt}(\operatorname{sig}(g,j))$, but still we need to compare all other $j-1$ terms of $\operatorname{sig}(f,j)$ with the j terms of $\operatorname{sig}(g,j)$ and recompute $\operatorname{sig}(f,j)$ possibly.

Remark 7.3.3.

(1) The second situation from above is completely new for signature–based algorithms: Sig–safe reductions are defined exactly the way such that one only needs to compute the reduction step on the polynomial part. In this generalized setting computations with the signatures need to be done, although the reduction is sig–safe! This leads to a huge computational overhead.

(2) Note that actions like criteria checking does not change at all, since they are based on the leading terms of the generalized signatures only. From the computational point of view a bit more overhead is generated due to the fact that one first needs to get the leading term out of the generalized signature before checking them in the respective implementation of (NM) and (RW).

(3) Be cautious, the memory consumption can increase quite a lot generalizing signatures. Thinking about thousands of critical pairs that need to be stored, most of them are useless, but they are possibly detected to be so with the respective implementation of (RW) first and thus are stored for the time being.

Having seen the amount of overhead generalizing the signatures introduces, let us discuss what are the benefits we can get. There are two advantages over a usual signature–based standard basis algorithm:

(1) Having j terms stored in the signatures, but using only the leading term of them in (NM) and (RW) would be quite foolish. The idea is to regularly check if one can interreduce the set of signatures and get a new term, the leading term of the reduced signature, for example as a new rule in F5. This is something one cannot do with usual signatures.

(2) Thinking about the sig–safe reduction process, it is now possible to perform a reduction $f - ug$ where $\operatorname{lt}(\operatorname{sig}(f,j)) = u \operatorname{lt}(\operatorname{sig}(g,j))$. In this situation only the leading term cancels out, but there are (hopefully) enough terms of the signature $\operatorname{sig}(f,j) - u\operatorname{sig}(g,j)$ left to construct a new signature of $f-ug$. Thus a sig–unsafe reduction can take place always besides the quite unusual situation where $\operatorname{sig}(f,j) - u\operatorname{sig}(g,j) = 0$. This could have a positive impact on computations in the inhomogeneous setting.

Although these benefits seem to be quite desirable it is clearly questionable how the ratio of drawbacks and advantages when generalizing signatures. Up to now no working algorithm including the ideas mentiond in this section is known, which is to the greatest part due to the complexity of its implementation. The author is working on such an implementation, but it is too early to present faithful experimental results. Still it is an area of the signature–based world that seems to be promising giving some nice new results improving the computation of standard bases.

7.4 Non–incremental signature–based standard basis algorithms

One of the biggest drawbacks of all signature–based algorithms presented until now is their dependency on incremental computations. If one wants to compute a standard basis for an ideal $I = \langle f_1, \ldots, f_r \rangle \subset \mathcal{P}$ in the signature–based world we compute the standard basis G_1 for $\langle f_1\rangle$, then G_2 for $\langle f_1, f_2\rangle$, and so on until we reach G_r, a standard basis for I. As long as the number r of ideal generators is not too big, e.g. in complete intersections, this is not a problem at all. Assuming r to be quite big (compared to the number of variables in the polynomial ring the ideal is defined in) this tends to be a problem: When computing a standard basis for $\langle f_1, \ldots, f_k\rangle$ for $1 < k < r$ information stored in $f_{k+1}, \ldots, f_r$ cannot be used as it is done in a non–incremental standard basis algorithm.

Some approach in the direction of non–incremental signature–based algorithms is already done:

(1) Arri and Perry presented a generalized variant of F5 in [7], which can be used in an incremental fashion as well as in a non–incremental one. We have presented the incremental version of this algorithm denoted AP in Section 5.3.

(2) Recently Gao, Volny, and Wang unveiled a generalization of the G2V Algorithm (see Section 5.5), called GVW ([77]).

The author is preparing an implementation of a signature–based algorithm in Singular to be also working in a non–incremental way. This is not only a non–trivial approach, but also the theory of this area is not really elaborated until now. Thus besides the aspects of implementation a lot of research in this field of non–incremental signature–based computations must be done.

In this section we straiten ourselves to a presentation of the general idea, the benefits and drawbacks such an attempt can have, and how heuristics play an important role to achieve a dynamically, auto–adjusting signature–based algorithm that can be used on a wide class of inputs without introducing penalties in performance or memory usage.

The first thing to do is to review why all the signature–based standard basis algorithms presented so far are tied to an incremental framework. In the prelude of Chapter 5 we determined the following restriction for all our considerations: The monomial order on the signatures is set to be $\prec_{\mathrm{i}}$. Let us review its definition:

$$\begin{aligned} m_i e_i \prec_{\mathrm{i}} m_j e_j :\Longleftrightarrow\ & i < j \text{ or,} \\ & i = j \text{ and } m_i < m_j \end{aligned}$$

for monomials $m_i, m_j \in \mathcal{P}$. Due to the fact that we have at the same time defined a connection between the finitely generated, free module $\mathcal{P}^r$ with canonical generators e_i and the ideal $I = \langle f_1, \ldots, f_r\rangle$, given by

$$\begin{array}{rccl} \pi: & \mathcal{P}^r & \longrightarrow & I \\ & e_i & \longmapsto & f_i \ \text{ for all } 1 \le i \le r \end{array}$$

this causes an incrementally working algorithm:

(1) On the one hand, any s–vector of a critical pair (uf, vg) with $\mathrm{index}(f) > \mathrm{index}(g)$ has the signature $u\,\mathrm{sig}(f)$.

(2) On the other hand, critical pairs are handled by increasing signature.

These two properties of signature–based algorithms evoke the matter of fact that, entering IncSig, for example, with two new initial elements $g_s = (e_i, f_i)$ and $g_t = (e_j, f_j)$ with $j > i$ no element which is generated by g_t is taken into account before all possible critical pairs generated by g_s and lower index elements are processed. Thus the algorithm would first compute a standard basis for $\langle f_1, \ldots, f_i \rangle$ before any impact of g_t on the computation takes place. Note that this does not only comprise critical pairs generated by g_t, but g_t in general: No reduction in SigRed with g_t can take place since they are all sig–unsafe. To the contrary, those sig–unsafe reductions generate new critical pairs whose signature has index j, again an element clogging the pair set P and whose computation is postponed to the point when a standard basis for $\langle f_1, \ldots, f_i \rangle$ is already computed.

Thus it is useless to start with all ideal generators at once as they are used in the computation only one thing at a time. Still more, it introduces lots of disadvantages:

▷ The algorithm computes and stores critical pairs which are useless at that point of the computations.

▷ Due to this lots of useless comparisons getting the element of smallest possible signature out of the pair set P have to be done.

▷ Even more, one must isolate g_t such that it is not used as a reducer, since this would end up with sig–unsafe reductions, generating even more, to be postponed critical pairs.

So what can be done to enable the usage of the information stored in g_t at the same time as we compute with g_s? Clearly, we cannot change the order in which critical pairs are handled, those still need to be processed by increasing signatures. Thus the only weak point left is the monomial order on the signatures. Instead of preferring the postion over term order we could use $<_{\mathrm{m}}$ or the Schreyer order $<_{\mathrm{lm}}$. The only real requirement on the chosen monomial order on the signatures can be defined by the following lemma:

Lemma 7.4.1. *Let $<$ be the monomial order on the set of all signatures. If $<$ is a well–order correctness of all signature–based algorithms presented in this thesis remains.*

Proof. This is clear since in all such proofs we only assumed that $<_{\mathrm{i}}$ is a well–order and that the critical pairs are processed by increasing signature. As long as these properties are still valid for the chosen order $<$, no proof is corrupted. □

In [7] Arri and Perry give their algorithm in a non–incremental fashion. As we have already discussed in Section 5.3, AP is, when using $<_{\mathrm{i}}$ as monomial order on the signatures, nothing else but F5 with an eased variant of F5's Rewritten Criterion. Sadly, the authors of the paper do not provide an implementation of their algorithm, the only publicly available implementation of AP is done by the author of this thesis, which is restricted to the incremental structure.

In [77] Gao, Volny, and Wang present their algorithm GVW, a generalized implementation of G2V which can handle different monomial orders on the set of signatures. Besides the module monomial orders defined in this thesis they give two more:

Definition 7.4.2. Let $m_i e_i, m_j e_j$ be two monomials in $\mathcal{P}^r$, $\langle f_1, \ldots, f_r \rangle$ and π as defined above.

(1)

$$\begin{aligned} m_i e_i <_{g1} m_j e_j :\Longleftrightarrow\ & \deg(m_i f_i) > \deg(m_j f_j) \text{ or,} \\ & \deg(m_i f_i) = \deg(m_j f_j) \text{ and } m_i e_i <_{m} m_j e_j. \end{aligned}$$

(2)

$$\begin{aligned} m_i e_i <_{g2} m_j e_j :\Longleftrightarrow\ & \mathrm{lm}(m_i f_i) > \mathrm{lm}(m_j f_j) \text{ or,} \\ & \mathrm{lm}(m_i f_i) = \mathrm{lm}(m_j f_j) \text{ and } m_i e_i <_{i} m_j e_j. \end{aligned}$$

Their experimental results show that in a range of examples the choice of $<_{g2}$ is way better than choosing $<_i$ as default in the incremental setting. Besides way faster timings a lot more useless critical pairs are detected in GVW using $<_{g2}$ compared to those discarded by G2V.

Remark 7.4.3.

(1) Note that $<_{g2}$ is nothing else but the Schreyer order $<_{lm}$ defined in Example 3.3.4: Checking $m_i e_i <_i m_j e_j$, whenever $\deg(m_i f_i) = \deg(m_j f_j)$ holds, just reduces to $i < j$.

(2) Two difficulties in their approach comparing the impact of the different order on the set of signatures might be pointed out here:

   a) Firstly, using $<_{g1}$ GVW is way slower than G2V. The problem is that they give only 9 examples, which are quite standard. Thus there is no complete picture of how different monomial orders influence the computations.

   b) Secondly, the source code of their implementation is not available to the public. It is not clear from their description in [77] if both, GVW and G2V are based on the same C++–implementation they mention, i.e. differing only by dropping in different monomial orders $<$. If those algorithms have two completely different implementations a comparison cannot be made accurately at all.

It follows that a lot of research has to be done in this area of the signature–based world. On the one hand, the benefits of well–chosen monomial orders on the signatures are quite impressive. On the other hand, these tests are not prestigious at all. Lots of different systems need to be tested, especially randomized ones and those coming from non–complete intersections.

Questions like the following need to be answered in the near future:

▷ Does it make sense to dynamically adjust $<$ and $<$ together as an generalization of the ideas presented in Sections 2.7 and 2.8?

▷ How can one optimize the behaviour of signature–based standard basis algorithms for inhomogeneous input by changing $<$?

▷ Using modern multicore processors in computers, one could start multiple computations for the same input with different orders on the signatures and take the first that finishes.

▷ Can one give heuristics for predefining optimal monomial orders $<$ for wide classes of systems?

▷ Do different orders $<$ have an influence on the "quality" of the resulting standard basis[2] like $<_{\mathrm{lp}}$ and $<_{\mathrm{dp}}$?

With this discussion the reader may get a feeling for the importance of research in this area. Quite a lot of new results and improvements can be expected.

7.5 Parallelization of signature–based algorithms

With modern multicore and multiprocessor computers available these days the question of parallelization of signature–based algorithms comes up quite naturally. There are two different ways of parallelizing:

(1) Speaking in terms of modular computations as presented in Section 2.9 parallelization of signature–based algorithms can be achieved as for any other standard basis algorithm: Computations in a polynomial ring over a ground field of characteristic 0 can be modularized to several calculations over different ground fields of finite characteristic, each being a prime which is lucky w.r.t. the respective setting. All we need to ensure, to make use of the ideas of Section 2.9, is the computation of the reduced standard bases. Thus we need to interreduce also after the last iteration step, which can easily be done and is not interfering with the signature–based computations at all. Getting a modular F5 is straightforward, even from the implementational point of view, thus we leave out any further discussion on this and refer to Section 2.9 for more details on problems and pitfalls of this approach.

Another idea of distributing computations on several parallel computations can be found in the incremental structure of the signature–based algorithms:[3]

a) Once a standard basis for $\langle f_1, \ldots, f_i \rangle$ is computed one could start several calls of the next incremental step with different initial input elements, say one computes a standard basis for $\langle f_1, \ldots, f_i, f_j \rangle$, the other for $\langle f_1 \ldots, f_i, f_k \rangle$, and so on. In the end we choose the one which finishes first and do this step recursively. This approach is still in the process of being implemented and should be available soon.

[2] This is meant in terms of further applications of the computed standard basis.
[3] Note that this idea can be applied to any incremental algorithm in general.

b) An even more sophisticated approach would be to completely divide the computations and merge them back together step by step. Let us illustrate this: Assume we want to compute a standard basis for $I = \langle f_1, \ldots, f_r \rangle \subset \mathcal{P}$.

   i. Then one could compute the ceiling $k = \lceil \frac{r}{2} \rceil$ and start the computation of standard bases for $\langle f_1, \ldots, f_k \rangle$ and $\langle f_{k+1}, \ldots, f_r \rangle$. Depending on the number of available processors resp. CPU cores this step can be done recursively.

   ii. So we end up with a couple of standard bases $G_1, \ldots, G_m$ where m denotes number of CPU cores available. Next we need to merge them together, i.e. we can, in parallel compute standard bases of $G_{1,2} = G_1 \cup G_2, \ldots, G_{m-1,m} = G_{m-1} \cup G_m$.

   iii. Again we do this step recursively and end up with a standard basis G for $\langle f_1, \ldots, f_r \rangle$.

   Clearly, this idea can be easily combined with Approach (1)a. The main problem of this attempt is also its fundament: the incremental structure. The standard bases G_i tend to have lots of elements, thus lots of incremental steps must be done merging the bases together. This has very bad effects considering timings of the overall algorithm as we have already discussed in more detail in Section 7.4. An idea would be to use at the point all G_is are computed a non–incremental signature–based algorithm. Sadly, due to the fact of lacking a good implementation we have not been able to test such situations until now.

c) Furthermore, note that all these ideas can also be combined with using different monomial orders on the set of signatures in each segmented computation.

At the latest from this point on, it becomes clear that parallelizing in this vein is not at all trivial. The main problem lies in the vast number of possibilities and combinations. A good heuristic of when to use which approach can only be achieved by a basis for test cases covering a wide range of different examples. Moreover, the basic algorithms must be implemented in a comparable way as otherwise the results lack validity.

(2) Another way of parallelizing signature–based computations has not been covered in this thesis until now. Instead of modularizing the computations using different processes on a computer, one could also parallelize code in a subtler way using different threads in a single process. So we are not talking about parallelizing a whole standard basis computation, but we want to parallelize only parts of it. Possible parts are:

   a) Critical pair generation,

   b) Criteria checks,

   c) Reduction process.

   Note that nearly all of this can be done in a Buchberger–like algorithm, too. Due to the fact that Singular is not thread–safe right now, we are not able to implement any of these ideas in the near future.

Let us finish this section with some rather general notes on parallelization of standard basis algorithms.

Remark 7.5.1.

(1) Parallelization of multiplication and division of polynomials is quite tricky and needs lots of implementational tricks. Maple has done some steps in this direction, but it took them more than one year to implement it ([128, 129]). Moreover, different philosophies on how to achieve the best result for sparse and dense polynomials are swirring around and no real winner of this competition has been found until now. Regarding the afford and time needed to implement this in a computer algebra system such a step should be deliberated.

(2) A more canonical approach on parallelizing the reduction process is to perform an F4–like Gaussian Elimination. The matrix operations can be parallelized easily, even using the huge amount of shaders on modern graphic cards. Faugère and Lachartre have obtained some nice results in [69].

(3) Note that all ideas given under (2) really need to be implemented in a thread–safe environment. The steps we want to parallelize are computed in a short amount of time, but happen quite a lot and do (mostly) not depend on each other. Using different processes as in the situation of the other presented ideas is not possible since the communication between the processes may take longer than the computational step itself. Thus a thread–safe memory management of Singular is the next step before we can start and implement our ideas on parallelizing signature–based algorithms.

7.6 Computing syzygies with generalized signature–based algorithms

With this section the wheel comes full circle in some sense. In Chapter 3 we started with a discussion on syzygies and their computation, ending with SyzStd (see Section 3.3), an algorithm which uses information from syzygies to compute a standard basis for an ideal. On the one hand, using information stored in the syzygies is quite good for detecting useless critical pairs in the algorithm. On the other hand, the general problem of this attempt lies in the fact that storing, computing, and comparing all the data stored in the syzygies slows down the standard basis computation, which is the main task.

In Chapter 4 we started improving the above mentioned idea by Möller, Mora, and Traverso storing only those parts of the corresponding module elements which are relevant for the detection of useless critical pairs, the leading terms. Those are denoted signatures, the basic concept all signature–based standard basis algorithms have in common.

In Section 7.3 we have seen that one can generalize the signatures to consist of more than the leading terms of the corresponding module elements. Now we take this generalization to a maximum, storing the whole syzygy resp. label of labeled polynomial in the algorithm. In contrast to Section 3.3 our intention this time is not to use those generalized signatures in a special way for the computation of a standard basis. We still use our signature–based standard basis algorithms with their respective implementations of (NM) and (RW). What we try to compute is a standard basis for the corresponding first module of syzygies.

So we have used ideas from syzygies and optimized their usage for standard basis computations in what we call signature–based algorithms. Now we use these optimized algorithms to improve the computation of syzygies.

In [10] Ars and Hashemi have already presented an algorithm to compute syzygies using a signature–based algorithm. Rather imprecise they called their publication *Computing Syzygies by Faugère's* F5 *Algorithm*, which is not completely correct: Instead of using F5 as presented in Section 6.1 they simplify F5 by not performing any checks with the Rewritten Criterion. Here we illustrate how to compute syzygies with a way improved version of this simplified F5 using not only principal syzygies for detecting useless elements.

Remark 7.6.1. Note that the results we present here can be applied to any signature–based algorithm presented in this thesis, even to F5E, the most aggressive and fastest of all such algorithms. This is a huge improvement compared to the work of Ars and Hashemi and is not published anywhere else before.

As in [10] we use the ideas of [155] to compute a basis for the first module of syzygies. Thus we first need to present an algorithm based on F5, which computes syzygies, too. This has to be in the vein of Algorithm 29 from Section 3.2, a variant of Std which also stores syzygies discovered during the standard basis computation.

Convention. In this section we again assume $F = \{f_1, \ldots, f_r\}$, a finite subset in $\mathcal{P}$, as input for our algorithm. We want to compute $\operatorname{Syz}(I)$ for $I = \langle f_1, \ldots, f_r \rangle$. $<$ always denotes a well–order on $\mathcal{P}$, whereas $\prec$ means the order $\prec_i$ on $\mathcal{P}^r$.

What is the main idea of computing $\operatorname{Syz}(I)$ using F5? On the one hand, F5 stores all leading terms of the principal syzygies in S for its checks of the F5 Criterion. Thus we just need to read this information off. On the other hand, we get the non–principal syzygies by reducing s–vectors to zero during F5's incremental standard basis computations. We present and explain the pseudo code of the algorithms in the following, again highlighting changes w.r.t. the corresponding code of the basic F5 Algorithm.

In our previous considerations we have been interested in carrying as few as possible data in our algorithms. We have found out that the leading term of the label of a polynomial is enough to detect useless critical pairs in the signature–based world. For this we introduced the notion of slim labeled polynomials in Definition 4.3.1. We agreed on the fact that we always assume labeled polynomials to be slim in the pseudo codes given in chapters 5 and 6. We need to invert this attempt: Whenever we reduce an element to zero in F5 we need to know the complete label, not only the corresponding signature! Thus we use the idea of generalizing the signatures already mentioned in Section 7.3 and always store the complete label of a labeled polynomial during the algorithm's working.

Definition 7.6.2. Let $r = (l, p)$ be a labeled polynomial. r is called *complete* iff $l \in \operatorname{labels}(p)$.

Looking at the code of Algorithm 62 we see that a new data structure is stored, namely the set of elements in $\mathcal{P}^r$, called T. In Line 14 F5Syz returns not only the standard basis B

for I, but also T, which is a standard basis for $\mathrm{Syz}(I)$. As already mentioned above we can distinguish in T principal and non–principal syzygies:

(1) Those which are added to T in Line 8 are the non–principal ones coming from the incremental computations of F5, which are now done in a variant of IncF5 called IncF5Syz and described in Algorithm 63.

(2) In the end we can read off the principal syzygies of $\mathrm{Syz}(I)$ by just computing all possible combinations of $f_j e_i - f_i e_j$, $j < i$ (Line 13).

Note that we are no longer allowed to reduce the initial f_i s before entering IncF5Syz. This is due to the fact that whereas it is clear that those reduction steps would be sig–safe, this time they have an impact on the labels of the corresponding elements. Note that these reductions take place in SigRedF5Syz later on, so there is no problem concerning the correctness of the standard basis computation of the algorithm.

Algorithm 62 The F5Syz Algorithm(F5Syz)

Input: $F = \{f_1, \ldots, f_r\}$ a finite subset of $\mathcal{P}$
Output: B a standard basis for $\langle F \rangle$ w.r.t. $<$, T a standard basis for $\mathrm{Syz}(F)$

1: $G_1 \leftarrow \{(e_1, f_1)\}$
2: S = empty list
3: R = empty list
4: $T = \varnothing$
5: **for** $(i = 2, \ldots, r)$ **do**
6:     **if** $(f_i \neq 0)$ **then**
7:         $G_i, S, R, T' \leftarrow$ IncF5Syz(f_i, G_{i-1}, S, R)
8:         $T \leftarrow T \cup T'$
9:     **else**
10:         $G_i \leftarrow G_{i-1}$
11: **for** $(i = 2, \ldots, r)$ **do**
12:     **for** $(j = 1, \ldots, i-1)$ **do**
13:         $T \leftarrow T \cup \{f_j e_i - f_i e_j\}$
14: $B \leftarrow \mathrm{poly}(G_r)$
15: **return** (B, T)

What is left is to see how the non–principal syzygies are computed in IncF5Syz. Clearly, using complete labeled polynomials we get the non–principal part of $\mathrm{Syz}(I)$ as a side effect, by just taking the label of the polynomial that is reduced to zero in SigRedF5Syz (see Line 33 of Algorithm 63). However, we obtain this data only since we need to perform more computations: we need to keep the label correct at every single reduction step! This leads to one fundamental change compared to F5: We need to adjust the label even when reducing with lower index elements! Thus we cannot swap those reductions out of SigRedF5Syz as we have done this for SigRedF5 (see Line 19). Moreover, generating the s–vector out of a not detected to be useless critical pair we also need to really compute the label and not only take the maximum of the two signatures as done in IncF5 (see Line 17).

The main differences to the usual F5 implementation lie in the following facts:

(1) IncF5Syz does not use the Rewritten Criterion at all, but it uses some of its data by adding corresponding criteria to S when a labeled polynomial reduces to zero (see Line 34).

(2) Due to this NonMinF5? is used again in Line 15 to check the critical pairs before generating s–vectors out of them; the list of criteria could have been updated because of an intermediate zero reduction.

Exactly this idea of actively using detected zero reductions makes our approach a huge optimization of the attempt Ars and Hashemi give in [10].

This means that the true syzygy computation takes place in Algorithm 64: In contrast to SigRedF5 all reducers of G are taken into account here (see Line 1 of SigRedF5Syz). Whenever a sig–safe reduction is allowed, we cannot prereduce the reducer $u\,\mathrm{poly}(g)$ with elements of lower index as done in SigRed, but we need to carry out all reductions step by step (Line 9). At the same time we reduce the polynomials, we need to adjust the corresponding labels, too.

(1) In Line 10 we update the value of the current label computing $\mathrm{label}(f) - u\,\mathrm{label}(g)$. This must be done for any sig–safe reduction step.

(2) If a sig–unsafe reduction is detected, then a new critical pair corresponding to this not processed reduction is added to the pair set P'. At this point we need to have the labeled polynomial $(\mathrm{label}(f), p)$ as second generator, as $\mathrm{label}(f)$ is the current label fitting to the polynomial data p at that exact moment of computations.

In Theorem 7.6.4 we show that making these changes in the code of F5 the resulting algorithm F5Syz computes besides a standard basis for the input ideal also a standard basis for the corresponding first module of syzygies.

Remark 7.6.3.

(1) Clearly, the idea of actively using zero reductions goes back to G2V and the results of Section 5.1, especially Corollary 5.1.4. We have already integrated this idea in F5E.

(2) Note that performing a sig–safe reduction step one can reduce the polynomial part and adjust the corresponding label quite easily in parallel on a multicore computer. Those computations only share the multiplier u, all other data is completely independent of each other during these steps.

Theorem 7.6.4. *Let $F = \{f_1, \ldots, f_r\}$ be the input of* F5Syz, *$I = \langle f_1, \ldots, f_r\rangle$. If the algorithm terminates, then it returns*

(1) a standard basis B for I w.r.t. $<$, and

(2) a standard basis T for $\mathrm{Syz}(F)$ *w.r.t. $\prec$.*

Proof. Part (1) is clear since we do not change the criteria checks and polynomial reductions compared to F5.

So we need to show (2): It is clear that $T \subset \mathrm{Syz}(F)$. Assume there exists some $s \in \mathrm{Syz}(F)\backslash T$. Then s must come from a, in F5Syz not processed, zero reduction of a critical pair (uf, vg). This means that (uf, vg) has been detected by the F5 Criterion. Assume that the actual computation is the ith call of IncF5Syz, i.e. the current index of labeled polynomials is i. Thus either a principal syzygy (corresponding to elements in the lists $S[1], \ldots, S[i-1]$ set in previous iteration steps of F5Syz) or a non–principal syzygy (corresponding to elements in $S[i-1]$ added to the list in the current iteration step whenever a zero reduction appeared) s' exists whose leading term divides the one of s.

Thus $L(T) = L(\mathrm{Syz}(F))$, T is a standard basis for $\mathrm{Syz}(F)$ w.r.t. $<$. □

Let us close this section with a remark on the problems the Rewritten Criterion could cause when computing $\mathrm{Syz}(F)$.

Remark 7.6.5. We have seen in Lemma 6.1.8 that whenever a labeled polynomial uf is detected by the Rewritten Criterion, then there exists a representation

$$u\,\mathrm{poly}(f) = t\,\mathrm{poly}(h) + \sum_{g_j \in G, g_j \neq h} \delta_j\,\mathrm{poly}(g_j)$$

such that

(1) $h \in G$ or $\mathrm{poly}(h) = 0$,

(2) for all g_j with $t_j \neq 0$ $t_j\,\mathrm{siglm}(g_j) < u\,\mathrm{siglm}(f)$, and

(3) $\mathrm{lm}(u)\,\mathrm{siglm}(f) = \mathrm{lm}(t)\,\mathrm{siglm}(h)$.

What happens if $\mathrm{poly}(h) \neq 0$? In this situation our proof of Theorem 7.6.4 does not work any more, we know that (uf, vg) is useless for the computation of the standard basis for I, but possibly the syzygy is needed for $\mathrm{Syz}(F)$. Since h has not reduced to zero, we do not know if the corresponding syzygy is necessary for the set T being a basis for $\mathrm{Syz}(F)$.

Thus, using only those rules coming from zero reductions by adding the corresponding signature leading terms to S ensures the correctness of F5Syz.

Algorithm 63 Incremental F5 step computing syzygies(IncF5Syz)

Input: f_i a polynomial, $G_{i-1} = \{g_1, \ldots, g_{s-1}\}$ a set of labeled polynomials such that $\operatorname{poly}(G_{i-1})$ is a standard basis for $\langle f_1, \ldots, f_{i-1}\rangle$, S a list of lists of terms in $\mathcal{P}$, R a list of $(i-1)$ lists of terms in $\mathcal{P}$

Output: G a set of labeled polynomials such that $\operatorname{poly}(G)$ is a standard basis for $\langle f_1, \ldots, f_i\rangle$, S a list of i lists of terms in $\mathcal{P}$, R a list of i lists of terms in $\mathcal{P}$, T a set of elements in $\mathcal{P}^r$

$B \leftarrow \varnothing, G \leftarrow \varnothing, P \leftarrow \varnothing, P' \leftarrow \varnothing, R[i] \leftarrow$ empty list, $S[i] \leftarrow$ empty list, $T \leftarrow \varnothing$
$t \leftarrow s$
$g_s \leftarrow (e_i, f_i)$
$S[i] \leftarrow \operatorname{addF5Crit}(\operatorname{lt}(g_s), S[i])$
$G \leftarrow \{g_1, \ldots, g_s\}$
for $(k = 1, \ldots, s-1)$ **do**
  $u \leftarrow \operatorname{lc}(g_k)\frac{\tau(g_s,g_k)}{\operatorname{lm}(g_s)}$
  $v \leftarrow \operatorname{lc}(g_s)\frac{\tau(g_s,g_k)}{\operatorname{lm}(g_k)}$
  if $\left(!\textsc{NonMinF5?}(ug_s, S) \text{ and } !\textsc{NonMinF5?}(vg_k, S)\right)$ **then**
    $P \leftarrow P \cup \{(ug_s, vg_k)\}$
while $(P \neq \varnothing)$ **do**
  $P' \leftarrow \textsc{Select}(P)$ (critical pairs of minimal degree)
  while $(P' \neq \varnothing)$ **do**
    Choose (uf, vg) from P' with $\max_{\prec}\{u \operatorname{sig}(f), v \operatorname{sig}(g)\}$ minimal w.r.t. $\prec$.
    if $(!\textsc{NonMinF5?}(u, f, R) \text{ and } !\textsc{NonMinF5?}(v, g, R))$ **then**
      $P' \leftarrow P' \setminus \{(uf, vg)\}$
      $l \leftarrow u \operatorname{label}(f) - v \operatorname{label}(g)$
      $r \leftarrow (l, u \operatorname{poly}(f) - v \operatorname{poly}(g))$
      $(r, P') \leftarrow \textsc{SigRedF5Syz}(r, G, S, R, s, P')$
      if $(\operatorname{poly}(r) \neq 0$ and r not sig–redundant w.r.t. $G)$ **then**
        $S[i] \leftarrow \operatorname{addF5Crit}(\operatorname{lt}(r), S[i])$
        for $(k = 1, \ldots, t)$ **do**
          if $(\operatorname{lm}(g_k) \nmid \operatorname{lm}(r))$ **then**
            $u \leftarrow \operatorname{lc}(g_k)\frac{\tau(r,g_k)}{\operatorname{lm}(r)}$
            $v \leftarrow \operatorname{lc}(r)\frac{\tau(r,g_k)}{\operatorname{lm}(g_k)}$
            if $(\operatorname{lm}(u) \operatorname{siglm}(r) \neq \operatorname{lm}(v) \operatorname{siglm}(g_k))$ **then**
              if $\left(!\textsc{NonMinF5?}(ur, S) \text{ and } !\textsc{NonMinF5?}(vg_k, S)\right)$ **then**
                $P \leftarrow P \cup \{(ur, vg_k)\}$
        $t \leftarrow t + 1$
        $g_t \leftarrow r$
        $G \leftarrow G \cup \{g_t\}$
      else
        $T \leftarrow T \cup \{\operatorname{label}(r)\}$
        $S[i-1] \leftarrow \operatorname{addF5Crit}(\operatorname{siglt}(r), S[i-1])$
return (G, S, R, T)

Algorithm 64 F5Syz's semi–complete sig–safe reduction algorithm (SigRedF5Syz)

Input: f a labeled polynomial, $G = \{g_1, \ldots, g_t\}$ a finite set of labeled polynomials, S a list of lists of terms in $\mathcal{P}$, R a list of lists of terms in $\mathcal{P}$, s the index of the first labeled polynomial of current index, P' a set of critical pairs

Output: h a labeled polynomial sig–safe reduced w.r.t. G, P' a set of critical pairs

$D \leftarrow G$
$l \leftarrow \mathrm{siglm}(f)$
$p \leftarrow \mathrm{poly}(f)$
while $\left(p \neq 0 \text{ and } D_p \leftarrow \left\{g \in D \mid \mathrm{lm}\left(\mathrm{poly}(g)\right) \mid \mathrm{lm}(p)\right\} \neq \varnothing\right)$ **do**
  Choose any $g \in D_p$.
  $u \leftarrow \frac{\mathrm{lt}(p)}{\mathrm{lt}(\mathrm{poly}(g))}$
  if $(!\,$NonMinF5?(ug, S) and $!\,$RewriteF5?$(u, g, R))$ **then**
    if $\left(\mathrm{lm}(u)\,\mathrm{siglm}(g) < l\right)$ **then**
      $p \leftarrow p - u\,\mathrm{poly}(g)$
      $\mathrm{label}(f) \leftarrow \mathrm{label}(f) - u\,\mathrm{label}(g)$
    else if $\left(\mathrm{lm}(u)\,\mathrm{siglm}(g) > l\right)$ **then**
      $P' \leftarrow P' \cup \{(ug, (\mathrm{label}(f), p))\}$
$h \leftarrow \left(\mathrm{label}(f), p\right)$
return (h, P')

A Examples

In the following we give a complete list of all examples used in this thesis. The examples are sorted by their names in increasing order. The code is given in the Singular language and is the exact data used for the computations done.

Note that "`-h`" at the ending of an example's name indicates that the corresponding ideal is homogeneous.

Cyclic-7

Polynomial ring in 7 variables: x(0), x(1), x(2), x(3), x(4), x(5), x(6)

$$
\begin{aligned}
\mathtt{i[1]} &= \mathtt{x(0)} + \mathtt{x(1)} + \mathtt{x(2)} + \mathtt{x(3)} + \mathtt{x(4)} + \mathtt{x(5)} + \mathtt{x(6)} \\
\mathtt{i[2]} &= \mathtt{x(0)}\cdot\mathtt{x(1)} + \mathtt{x(1)}\cdot\mathtt{x(2)} + \mathtt{x(2)}\cdot\mathtt{x(3)} + \mathtt{x(3)}\cdot\mathtt{x(4)} + \mathtt{x(4)}\cdot\mathtt{x(5)}+ \\
&\quad \mathtt{x(0)}\cdot\mathtt{x(6)} + \mathtt{x(5)}\cdot\mathtt{x(6)} \\
\mathtt{i[3]} &= \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(2)} + \mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(3)} + \mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)}+ \\
&\quad \mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)} + \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(6)} + \mathtt{x(0)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)}+ \\
&\quad \mathtt{x(4)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} \\
\mathtt{i[4]} &= \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(3)} + \mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)} + \mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)}+ \\
&\quad \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(6)} + \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} + \mathtt{x(0)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)}+ \\
&\quad \mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} \\
\mathtt{i[5]} &= \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)} + \mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)}+ \\
&\quad \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(6)} + \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)}+ \\
&\quad \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} + \mathtt{x(0)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)}+ \\
&\quad \mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} \\
\mathtt{i[6]} &= \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)} + \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(6)}+ \\
&\quad \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} + \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)}+ \\
&\quad \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} + \mathtt{x(0)}\cdot\mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)}+ \\
&\quad \mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} \\
\mathtt{i[7]} &= \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} - 1
\end{aligned}
$$

Cyclic-8

Polynomial ring in 8 variables: $\mathtt{x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7)}$

$$
\begin{aligned}
\mathtt{i[1]} &= \mathtt{x(0)+x(1)+x(2)+x(3)+x(4)+x(5)+x(6)+x(7)}\\
\mathtt{i[2]} &= \mathtt{x(0)\cdot x(1)+x(1)\cdot x(2)+x(2)\cdot x(3)+x(3)\cdot x(4)+x(4)\cdot x(5)+}\\
&\quad \mathtt{x(5)\cdot x(6)+x(0)\cdot x(7)+x(6)\cdot x(7)}\\
\mathtt{i[3]} &= \mathtt{x(0)\cdot x(1)\cdot x(2)+x(1)\cdot x(2)\cdot x(3)+x(2)\cdot x(3)\cdot x(4)+}\\
&\quad \mathtt{x(3)\cdot x(4)\cdot x(5)+x(4)\cdot x(5)\cdot x(6)+x(0)\cdot x(1)\cdot x(7)+}\\
&\quad \mathtt{x(0)\cdot x(6)\cdot x(7)+x(5)\cdot x(6)\cdot x(7)}\\
\mathtt{i[4]} &= \mathtt{x(0)\cdot x(1)\cdot x(2)\cdot x(3)+x(1)\cdot x(2)\cdot x(3)\cdot x(4)+x(2)\cdot x(3)\cdot x(4)\cdot x(5)+}\\
&\quad \mathtt{x(3)\cdot x(4)\cdot x(5)\cdot x(6)+x(0)\cdot x(1)\cdot x(2)\cdot x(7)+x(0)\cdot x(1)\cdot x(6)\cdot x(7)+}\\
&\quad \mathtt{x(0)\cdot x(5)\cdot x(6)\cdot x(7)+x(4)\cdot x(5)\cdot x(6)\cdot x(7)}\\
\mathtt{i[5]} &= \mathtt{x(0)\cdot x(1)\cdot x(2)\cdot x(3)\cdot x(4)+x(1)\cdot x(2)\cdot x(3)\cdot x(4)\cdot x(5)+}\\
&\quad \mathtt{x(2)\cdot x(3)\cdot x(4)\cdot x(5)\cdot x(6)+x(0)\cdot x(1)\cdot x(2)\cdot x(3)\cdot x(7)+}\\
&\quad \mathtt{x(0)\cdot x(1)\cdot x(2)\cdot x(6)\cdot x(7)+x(0)\cdot x(1)\cdot x(5)\cdot x(6)\cdot x(7)+}\\
&\quad \mathtt{x(0)\cdot x(4)\cdot x(5)\cdot x(6)\cdot x(7)+x(3)\cdot x(4)\cdot x(5)\cdot x(6)\cdot x(7)}\\
\mathtt{i[6]} &= \mathtt{x(0)\cdot x(1)\cdot x(2)\cdot x(3)\cdot x(4)\cdot x(5)+x(1)\cdot x(2)\cdot x(3)\cdot x(4)\cdot x(5)\cdot x(6)+}\\
&\quad \mathtt{x(0)\cdot x(1)\cdot x(2)\cdot x(3)\cdot x(4)\cdot x(7)+x(0)\cdot x(1)\cdot x(2)\cdot x(3)\cdot x(6)\cdot x(7)+}\\
&\quad \mathtt{x(0)\cdot x(1)\cdot x(2)\cdot x(5)\cdot x(6)\cdot x(7)+x(0)\cdot x(1)\cdot x(4)\cdot x(5)\cdot x(6)\cdot x(7)+}\\
&\quad \mathtt{x(0)\cdot x(3)\cdot x(4)\cdot x(5)\cdot x(6)\cdot x(7)+}\\
&\quad \mathtt{x(2)\cdot x(3)\cdot x(4)\cdot x(5)\cdot x(6)\cdot x(7)}\\
\mathtt{i[7]} &= \mathtt{x(0)\cdot x(1)\cdot x(2)\cdot x(3)\cdot x(4)\cdot x(5)\cdot x(6)+x(0)\cdot x(1)\cdot x(2)\cdot x(3)\cdot x(4)\cdot x(5)\cdot x(7)+}\\
&\quad \mathtt{x(0)\cdot x(1)\cdot x(2)\cdot x(3)\cdot x(4)\cdot x(6)\cdot x(7)+x(0)\cdot x(1)\cdot x(2)\cdot x(3)\cdot x(5)\cdot x(6)\cdot x(7)+}\\
&\quad \mathtt{x(0)\cdot x(1)\cdot x(2)\cdot x(4)\cdot x(5)\cdot x(6)\cdot x(7)+x(0)\cdot x(1)\cdot x(3)\cdot x(4)\cdot x(5)\cdot x(6)\cdot x(7)+}\\
&\quad \mathtt{x(0)\cdot x(2)\cdot x(3)\cdot x(4)\cdot x(5)\cdot x(6)\cdot x(7)+}\\
&\quad \mathtt{x(1)\cdot x(2)\cdot x(3)\cdot x(4)\cdot x(5)\cdot x(6)\cdot x(7)}\\
\mathtt{i[8]} &= \mathtt{x(0)\cdot x(1)\cdot x(2)\cdot x(3)\cdot x(4)\cdot x(5)\cdot x(6)\cdot x(7)-1}
\end{aligned}
$$

Eco-8

Polynomial ring in 8 variables: $\mathtt{x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7)}$

$$
\begin{aligned}
\mathtt{i[1]} &= \mathtt{x(0)\cdot x(1)\cdot x(7) + x(1)\cdot x(2)\cdot x(7) + x(2)\cdot x(3)\cdot x(7) +} \\
&\quad \mathtt{x(3)\cdot x(4)\cdot x(7) + x(4)\cdot x(5)\cdot x(7) + x(5)\cdot x(6)\cdot x(7) +} \\
&\quad \mathtt{x(0)\cdot x(7) - 1} \\
\mathtt{i[2]} &= \mathtt{x(0)\cdot x(2)\cdot x(7) + x(1)\cdot x(3)\cdot x(7) + x(2)\cdot x(4)\cdot x(7) +} \\
&\quad \mathtt{x(3)\cdot x(5)\cdot x(7) + x(4)\cdot x(6)\cdot x(7) + x(1)\cdot x(7) - 2} \\
\mathtt{i[3]} &= \mathtt{x(0)\cdot x(3)\cdot x(7) + x(1)\cdot x(4)\cdot x(7) + x(2)\cdot x(5)\cdot x(7) +} \\
&\quad \mathtt{x(3)\cdot x(6)\cdot x(7) + x(2)\cdot x(7) - 3} \\
\mathtt{i[4]} &= \mathtt{x(0)\cdot x(4)\cdot x(7) + x(1)\cdot x(5)\cdot x(7) + x(2)\cdot x(6)\cdot x(7) +} \\
&\quad \mathtt{x(3)\cdot x(7) - 4} \\
\mathtt{i[5]} &= \mathtt{x(0)\cdot x(5)\cdot x(7) + x(1)\cdot x(6)\cdot x(7) + x(4)\cdot x(7) - 5} \\
\mathtt{i[6]} &= \mathtt{x(0)\cdot x(6)\cdot x(7) + x(5)\cdot x(7) - 6} \\
\mathtt{i[7]} &= \mathtt{x(6)\cdot x(7) - 7} \\
\mathtt{i[8]} &= \mathtt{x(0) + x(1) + x(2) + x(3) + x(4) + x(5) + x(6) + 1}
\end{aligned}
$$

Eco-8-h

Polynomial ring in 9 variables: $\mathtt{x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), h}$

$$
\begin{array}{lcl}
\mathtt{i[1]} & = & \mathtt{x(0)\cdot x(1)\cdot x(7) + x(1)\cdot x(2)\cdot x(7) + x(2)\cdot x(3)\cdot x(7) +} \\
 & & \mathtt{x(3)\cdot x(4)\cdot x(7) + x(4)\cdot x(5)\cdot x(7) + x(5)\cdot x(6)\cdot x(7) +} \\
 & & \mathtt{x(0)\cdot x(7)\cdot h - h\hat{\ }3} \\
\mathtt{i[2]} & = & \mathtt{x(0)\cdot x(2)\cdot x(7) + x(1)\cdot x(3)\cdot x(7) + x(2)\cdot x(4)\cdot x(7) +} \\
 & & \mathtt{x(3)\cdot x(5)\cdot x(7) + x(4)\cdot x(6)\cdot x(7) + x(1)\cdot x(7)\cdot h - 2\cdot h\hat{\ }3} \\
\mathtt{i[3]} & = & \mathtt{x(0)\cdot x(3)\cdot x(7) + x(1)\cdot x(4)\cdot x(7) + x(2)\cdot x(5)\cdot x(7) +} \\
 & & \mathtt{x(3)\cdot x(6)\cdot x(7) + x(2)\cdot x(7)\cdot h - 3\cdot h\hat{\ }3} \\
\mathtt{i[4]} & = & \mathtt{x(0)\cdot x(4)\cdot x(7) + x(1)\cdot x(5)\cdot x(7) + x(2)\cdot x(6)\cdot x(7) +} \\
 & & \mathtt{x(3)\cdot x(7)\cdot h - 4\cdot h\hat{\ }3} \\
\mathtt{i[5]} & = & \mathtt{x(0)\cdot x(5)\cdot x(7) + x(1)\cdot x(6)\cdot x(7) + x(4)\cdot x(7)\cdot h - 5\cdot h\hat{\ }3} \\
\mathtt{i[6]} & = & \mathtt{x(0)\cdot x(6)\cdot x(7) + x(5)\cdot x(7)\cdot h - 6\cdot h\hat{\ }3} \\
\mathtt{i[7]} & = & \mathtt{x(6)\cdot x(7) - 7\cdot h\hat{\ }2} \\
\mathtt{i[8]} & = & \mathtt{x(0) + x(1) + x(2) + x(3) + x(4) + x(5) + x(6) + h}
\end{array}
$$

Eco-9

Polynomial ring in 9 variables: $\mathtt{x(0)}, \mathtt{x(1)}, \mathtt{x(2)}, \mathtt{x(3)}, \mathtt{x(4)}, \mathtt{x(5)}, \mathtt{x(6)}, \mathtt{x(7)}, \mathtt{x(8)}$

$$
\begin{aligned}
\mathtt{i[1]} &= \mathtt{x(0)}\cdot\mathtt{x(1)}\cdot\mathtt{x(8)}+\mathtt{x(1)}\cdot\mathtt{x(2)}\cdot\mathtt{x(8)}+\mathtt{x(2)}\cdot\mathtt{x(3)}\cdot\mathtt{x(8)}+\\
&\quad \mathtt{x(3)}\cdot\mathtt{x(4)}\cdot\mathtt{x(8)}+\mathtt{x(4)}\cdot\mathtt{x(5)}\cdot\mathtt{x(8)}+\mathtt{x(5)}\cdot\mathtt{x(6)}\cdot\mathtt{x(8)}+\\
&\quad \mathtt{x(6)}\cdot\mathtt{x(7)}\cdot\mathtt{x(8)}+\mathtt{x(0)}\cdot\mathtt{x(8)}-1\\
\mathtt{i[2]} &= \mathtt{x(0)}\cdot\mathtt{x(2)}\cdot\mathtt{x(8)}+\mathtt{x(1)}\cdot\mathtt{x(3)}\cdot\mathtt{x(8)}+\mathtt{x(2)}\cdot\mathtt{x(4)}\cdot\mathtt{x(8)}+\\
&\quad \mathtt{x(3)}\cdot\mathtt{x(5)}\cdot\mathtt{x(8)}+\mathtt{x(4)}\cdot\mathtt{x(6)}\cdot\mathtt{x(8)}+\mathtt{x(5)}\cdot\mathtt{x(7)}\cdot\mathtt{x(8)}+\\
&\quad \mathtt{x(1)}\cdot\mathtt{x(8)}-2\\
\mathtt{i[3]} &= \mathtt{x(0)}\cdot\mathtt{x(3)}\cdot\mathtt{x(8)}+\mathtt{x(1)}\cdot\mathtt{x(4)}\cdot\mathtt{x(8)}+\mathtt{x(2)}\cdot\mathtt{x(5)}\cdot\mathtt{x(8)}+\\
&\quad \mathtt{x(3)}\cdot\mathtt{x(6)}\cdot\mathtt{x(8)}+\mathtt{x(4)}\cdot\mathtt{x(7)}\cdot\mathtt{x(8)}+\mathtt{x(2)}\cdot\mathtt{x(8)}-3\\
\mathtt{i[4]} &= \mathtt{x(0)}\cdot\mathtt{x(4)}\cdot\mathtt{x(8)}+\mathtt{x(1)}\cdot\mathtt{x(5)}\cdot\mathtt{x(8)}+\mathtt{x(2)}\cdot\mathtt{x(6)}\cdot\mathtt{x(8)}+\\
&\quad \mathtt{x(3)}\cdot\mathtt{x(7)}\cdot\mathtt{x(8)}+\mathtt{x(3)}\cdot\mathtt{x(8)}-4\\
\mathtt{i[5]} &= \mathtt{x(0)}\cdot\mathtt{x(5)}\cdot\mathtt{x(8)}+\mathtt{x(1)}\cdot\mathtt{x(6)}\cdot\mathtt{x(8)}+\mathtt{x(2)}\cdot\mathtt{x(7)}\cdot\mathtt{x(8)}+\\
&\quad \mathtt{x(4)}\cdot\mathtt{x(8)}-5\\
\mathtt{i[6]} &= \mathtt{x(0)}\cdot\mathtt{x(6)}\cdot\mathtt{x(8)}+\mathtt{x(1)}\cdot\mathtt{x(7)}\cdot\mathtt{x(8)}+\mathtt{x(5)}\cdot\mathtt{x(8)}-6\\
\mathtt{i[7]} &= \mathtt{x(0)}\cdot\mathtt{x(7)}\cdot\mathtt{x(8)}+\mathtt{x(6)}\cdot\mathtt{x(8)}-7\\
\mathtt{i[8]} &= \mathtt{x(7)}\cdot\mathtt{x(8)}-8\\
\mathtt{i[9]} &= \mathtt{x(0)}+\mathtt{x(1)}+\mathtt{x(2)}+\mathtt{x(3)}+\mathtt{x(4)}+\mathtt{x(5)}+\mathtt{x(6)}+\mathtt{x(7)}+1
\end{aligned}
$$

Eco-9-h

Polynomial ring in 10 variables: $\texttt{x(0)}, \texttt{x(1)}, \texttt{x(2)}, \texttt{x(3)}, \texttt{x(4)}, \texttt{x(5)}, \texttt{x(6)}, \texttt{x(7)}, \texttt{x(8)}, \texttt{h}$

$$
\begin{aligned}
\texttt{i[1]} &= \texttt{x(0)}\cdot\texttt{x(1)}\cdot\texttt{x(8)}+\texttt{x(1)}\cdot\texttt{x(2)}\cdot\texttt{x(8)}+\texttt{x(2)}\cdot\texttt{x(3)}\cdot\texttt{x(8)}+\\
&\quad \texttt{x(3)}\cdot\texttt{x(4)}\cdot\texttt{x(8)}+\texttt{x(4)}\cdot\texttt{x(5)}\cdot\texttt{x(8)}+\texttt{x(5)}\cdot\texttt{x(6)}\cdot\texttt{x(8)}+\\
&\quad \texttt{x(6)}\cdot\texttt{x(7)}\cdot\texttt{x(8)}+\texttt{x(0)}\cdot\texttt{x(8)}\cdot\texttt{h}-\texttt{h\^{}3}\\
\texttt{i[2]} &= \texttt{x(0)}\cdot\texttt{x(2)}\cdot\texttt{x(8)}+\texttt{x(1)}\cdot\texttt{x(3)}\cdot\texttt{x(8)}+\texttt{x(2)}\cdot\texttt{x(4)}\cdot\texttt{x(8)}+\\
&\quad \texttt{x(3)}\cdot\texttt{x(5)}\cdot\texttt{x(8)}+\texttt{x(4)}\cdot\texttt{x(6)}\cdot\texttt{x(8)}+\texttt{x(5)}\cdot\texttt{x(7)}\cdot\texttt{x(8)}+\\
&\quad \texttt{x(1)}\cdot\texttt{x(8)}\cdot\texttt{h}-\texttt{2}\cdot\texttt{h\^{}3}\\
\texttt{i[3]} &= \texttt{x(0)}\cdot\texttt{x(3)}\cdot\texttt{x(8)}+\texttt{x(1)}\cdot\texttt{x(4)}\cdot\texttt{x(8)}+\texttt{x(2)}\cdot\texttt{x(5)}\cdot\texttt{x(8)}+\\
&\quad \texttt{x(3)}\cdot\texttt{x(6)}\cdot\texttt{x(8)}+\texttt{x(4)}\cdot\texttt{x(7)}\cdot\texttt{x(8)}+\texttt{x(2)}\cdot\texttt{x(8)}\cdot\texttt{h}-\texttt{3}\cdot\texttt{h\^{}3}\\
\texttt{i[4]} &= \texttt{x(0)}\cdot\texttt{x(4)}\cdot\texttt{x(8)}+\texttt{x(1)}\cdot\texttt{x(5)}\cdot\texttt{x(8)}+\texttt{x(2)}\cdot\texttt{x(6)}\cdot\texttt{x(8)}+\\
&\quad \texttt{x(3)}\cdot\texttt{x(7)}\cdot\texttt{x(8)}+\texttt{x(3)}\cdot\texttt{x(8)}\cdot\texttt{h}-\texttt{4}\cdot\texttt{h\^{}3}\\
\texttt{i[5]} &= \texttt{x(0)}\cdot\texttt{x(5)}\cdot\texttt{x(8)}+\texttt{x(1)}\cdot\texttt{x(6)}\cdot\texttt{x(8)}+\texttt{x(2)}\cdot\texttt{x(7)}\cdot\texttt{x(8)}+\\
&\quad \texttt{x(4)}\cdot\texttt{x(8)}\cdot\texttt{h}-\texttt{5}\cdot\texttt{h\^{}3}\\
\texttt{i[6]} &= \texttt{x(0)}\cdot\texttt{x(6)}\cdot\texttt{x(8)}+\texttt{x(1)}\cdot\texttt{x(7)}\cdot\texttt{x(8)}+\texttt{x(5)}\cdot\texttt{x(8)}\cdot\texttt{h}-\texttt{6}\cdot\texttt{h\^{}3}\\
\texttt{i[7]} &= \texttt{x(0)}\cdot\texttt{x(7)}\cdot\texttt{x(8)}+\texttt{x(6)}\cdot\texttt{x(8)}\cdot\texttt{h}-\texttt{7}\cdot\texttt{h\^{}3}\\
\texttt{i[8]} &= \texttt{x(7)}\cdot\texttt{x(8)}-\texttt{8}\cdot\texttt{h\^{}2}\\
\texttt{i[9]} &= \texttt{x(0)}+\texttt{x(1)}+\texttt{x(2)}+\texttt{x(3)}+\texttt{x(4)}+\texttt{x(5)}+\texttt{x(6)}+\texttt{x(7)}+\texttt{h}
\end{aligned}
$$

Eco-10

Polynomial ring in 10 variables: x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), x(9)

$$
\begin{aligned}
\mathtt{i[1]} &= x(0)\cdot x(1)\cdot x(9) + x(1)\cdot x(2)\cdot x(9) + x(2)\cdot x(3)\cdot x(9) +\\
&\quad x(3)\cdot x(4)\cdot x(9) + x(4)\cdot x(5)\cdot x(9) + x(5)\cdot x(6)\cdot x(9) +\\
&\quad x(6)\cdot x(7)\cdot x(9) + x(7)\cdot x(8)\cdot x(9) + x(0)\cdot x(9) - 1\\
\mathtt{i[2]} &= x(0)\cdot x(2)\cdot x(9) + x(1)\cdot x(3)\cdot x(9) + x(2)\cdot x(4)\cdot x(9) +\\
&\quad x(3)\cdot x(5)\cdot x(9) + x(4)\cdot x(6)\cdot x(9) + x(5)\cdot x(7)\cdot x(9) +\\
&\quad x(6)\cdot x(8)\cdot x(9) + x(1)\cdot x(9) - 2\\
\mathtt{i[3]} &= x(0)\cdot x(3)\cdot x(9) + x(1)\cdot x(4)\cdot x(9) + x(2)\cdot x(5)\cdot x(9) +\\
&\quad x(3)\cdot x(6)\cdot x(9) + x(4)\cdot x(7)\cdot x(9) + x(5)\cdot x(8)\cdot x(9) +\\
&\quad x(2)\cdot x(9) - 3\\
\mathtt{i[4]} &= x(0)\cdot x(4)\cdot x(9) + x(1)\cdot x(5)\cdot x(9) + x(2)\cdot x(6)\cdot x(9) +\\
&\quad x(3)\cdot x(7)\cdot x(9) + x(4)\cdot x(8)\cdot x(9) + x(3)\cdot x(9) - 4\\
\mathtt{i[5]} &= x(0)\cdot x(5)\cdot x(9) + x(1)\cdot x(6)\cdot x(9) + x(2)\cdot x(7)\cdot x(9) +\\
&\quad x(3)\cdot x(8)\cdot x(9) + x(4)\cdot x(9) - 5\\
\mathtt{i[6]} &= x(0)\cdot x(6)\cdot x(9) + x(1)\cdot x(7)\cdot x(9) + x(2)\cdot x(8)\cdot x(9) +\\
&\quad x(5)\cdot x(9) - 6\\
\mathtt{i[7]} &= x(0)\cdot x(7)\cdot x(9) + x(1)\cdot x(8)\cdot x(9) + x(6)\cdot x(9) - 7\\
\mathtt{i[8]} &= x(0)\cdot x(8)\cdot x(9) + x(7)\cdot x(9) - 8\\
\mathtt{i[9]} &= x(8)\cdot x(9) - 9\\
\mathtt{i[10]} &= x(0) + x(1) + x(2) + x(3) + x(4) + x(5) + x(6) + x(7) + x(8) + 1
\end{aligned}
$$

Eco-10-h

Polynomial ring in 11 variables: $\texttt{x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), x(9), h}$

$$
\begin{aligned}
\texttt{i[1]} &= \texttt{x(0)} \cdot \texttt{x(1)} \cdot \texttt{x(9)} + \texttt{x(1)} \cdot \texttt{x(2)} \cdot \texttt{x(9)} + \texttt{x(2)} \cdot \texttt{x(3)} \cdot \texttt{x(9)} + \\
&\quad \texttt{x(3)} \cdot \texttt{x(4)} \cdot \texttt{x(9)} + \texttt{x(4)} \cdot \texttt{x(5)} \cdot \texttt{x(9)} + \texttt{x(5)} \cdot \texttt{x(6)} \cdot \texttt{x(9)} + \\
&\quad \texttt{x(6)} \cdot \texttt{x(7)} \cdot \texttt{x(9)} + \texttt{x(7)} \cdot \texttt{x(8)} \cdot \texttt{x(9)} + \texttt{x(0)} \cdot \texttt{x(9)} \cdot \texttt{h} - \texttt{h\^{}3} \\
\texttt{i[2]} &= \texttt{x(0)} \cdot \texttt{x(2)} \cdot \texttt{x(9)} + \texttt{x(1)} \cdot \texttt{x(3)} \cdot \texttt{x(9)} + \texttt{x(2)} \cdot \texttt{x(4)} \cdot \texttt{x(9)} + \\
&\quad \texttt{x(3)} \cdot \texttt{x(5)} \cdot \texttt{x(9)} + \texttt{x(4)} \cdot \texttt{x(6)} \cdot \texttt{x(9)} + \texttt{x(5)} \cdot \texttt{x(7)} \cdot \texttt{x(9)} + \\
&\quad \texttt{x(6)} \cdot \texttt{x(8)} \cdot \texttt{x(9)} + \texttt{x(1)} \cdot \texttt{x(9)} \cdot \texttt{h} - \texttt{2} \cdot \texttt{h\^{}3} \\
\texttt{i[3]} &= \texttt{x(0)} \cdot \texttt{x(3)} \cdot \texttt{x(9)} + \texttt{x(1)} \cdot \texttt{x(4)} \cdot \texttt{x(9)} + \texttt{x(2)} \cdot \texttt{x(5)} \cdot \texttt{x(9)} + \\
&\quad \texttt{x(3)} \cdot \texttt{x(6)} \cdot \texttt{x(9)} + \texttt{x(4)} \cdot \texttt{x(7)} \cdot \texttt{x(9)} + \texttt{x(5)} \cdot \texttt{x(8)} \cdot \texttt{x(9)} + \\
&\quad \texttt{x(2)} \cdot \texttt{x(9)} \cdot \texttt{h} - \texttt{3} \cdot \texttt{h\^{}3} \\
\texttt{i[4]} &= \texttt{x(0)} \cdot \texttt{x(4)} \cdot \texttt{x(9)} + \texttt{x(1)} \cdot \texttt{x(5)} \cdot \texttt{x(9)} + \texttt{x(2)} \cdot \texttt{x(6)} \cdot \texttt{x(9)} + \\
&\quad \texttt{x(3)} \cdot \texttt{x(7)} \cdot \texttt{x(9)} + \texttt{x(4)} \cdot \texttt{x(8)} \cdot \texttt{x(9)} + \texttt{x(3)} \cdot \texttt{x(9)} \cdot \texttt{h} - \texttt{4} \cdot \texttt{h\^{}3} \\
\texttt{i[5]} &= \texttt{x(0)} \cdot \texttt{x(5)} \cdot \texttt{x(9)} + \texttt{x(1)} \cdot \texttt{x(6)} \cdot \texttt{x(9)} + \texttt{x(2)} \cdot \texttt{x(7)} \cdot \texttt{x(9)} + \\
&\quad \texttt{x(3)} \cdot \texttt{x(8)} \cdot \texttt{x(9)} + \texttt{x(4)} \cdot \texttt{x(9)} \cdot \texttt{h} - \texttt{5} \cdot \texttt{h\^{}3} \\
\texttt{i[6]} &= \texttt{x(0)} \cdot \texttt{x(6)} \cdot \texttt{x(9)} + \texttt{x(1)} \cdot \texttt{x(7)} \cdot \texttt{x(9)} + \texttt{x(2)} \cdot \texttt{x(8)} \cdot \texttt{x(9)} + \\
&\quad \texttt{x(5)} \cdot \texttt{x(9)} \cdot \texttt{h} - \texttt{6} \cdot \texttt{h\^{}3} \\
\texttt{i[7]} &= \texttt{x(0)} \cdot \texttt{x(7)} \cdot \texttt{x(9)} + \texttt{x(1)} \cdot \texttt{x(8)} \cdot \texttt{x(9)} + \texttt{x(6)} \cdot \texttt{x(9)} \cdot \texttt{h} - \texttt{7} \cdot \texttt{h\^{}3} \\
\texttt{i[8]} &= \texttt{x(0)} \cdot \texttt{x(8)} \cdot \texttt{x(9)} + \texttt{x(7)} \cdot \texttt{x(9)} \cdot \texttt{h} - \texttt{8} \cdot \texttt{h\^{}3} \\
\texttt{i[9]} &= \texttt{x(8)} \cdot \texttt{x(9)} - \texttt{9} \cdot \texttt{h\^{}2} \\
\texttt{i[10]} &= \texttt{x(0)} + \texttt{x(1)} + \texttt{x(2)} + \texttt{x(3)} + \texttt{x(4)} + \texttt{x(5)} + \texttt{x(6)} + \texttt{x(7)} + \texttt{x(8)} + \texttt{h}
\end{aligned}
$$

Eco-11

Polynomial ring in 11 variables: $\mathtt{x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), x(9), x(10)}$

$$
\begin{array}{lcl}
\mathtt{i[1]} & = & \mathtt{x(0)\cdot x(1)\cdot x(10) + x(1)\cdot x(2)\cdot x(10) + x(2)\cdot x(3)\cdot x(10) +} \\
 & & \mathtt{x(3)\cdot x(4)\cdot x(10) + x(4)\cdot x(5)\cdot x(10) + x(5)\cdot x(6)\cdot x(10) +} \\
 & & \mathtt{x(6)\cdot x(7)\cdot x(10) + x(7)\cdot x(8)\cdot x(10) + x(8)\cdot x(9)\cdot x(10) +} \\
 & & \mathtt{x(0)\cdot x(10) - 1} \\
\mathtt{i[2]} & = & \mathtt{x(0)\cdot x(2)\cdot x(10) + x(1)\cdot x(3)\cdot x(10) + x(2)\cdot x(4)\cdot x(10) +} \\
 & & \mathtt{x(3)\cdot x(5)\cdot x(10) + x(4)\cdot x(6)\cdot x(10) + x(5)\cdot x(7)\cdot x(10) +} \\
 & & \mathtt{x(6)\cdot x(8)\cdot x(10) + x(7)\cdot x(9)\cdot x(10) + x(1)\cdot x(10) - 2} \\
\mathtt{i[3]} & = & \mathtt{x(0)\cdot x(3)\cdot x(10) + x(1)\cdot x(4)\cdot x(10) + x(2)\cdot x(5)\cdot x(10) +} \\
 & & \mathtt{x(3)\cdot x(6)\cdot x(10) + x(4)\cdot x(7)\cdot x(10) + x(5)\cdot x(8)\cdot x(10) +} \\
 & & \mathtt{x(6)\cdot x(9)\cdot x(10) + x(2)\cdot x(10) - 3} \\
\mathtt{i[4]} & = & \mathtt{x(0)\cdot x(4)\cdot x(10) + x(1)\cdot x(5)\cdot x(10) + x(2)\cdot x(6)\cdot x(10) +} \\
 & & \mathtt{x(3)\cdot x(7)\cdot x(10) + x(4)\cdot x(8)\cdot x(10) + x(5)\cdot x(9)\cdot x(10) +} \\
 & & \mathtt{x(3)\cdot x(10) - 4} \\
\mathtt{i[5]} & = & \mathtt{x(0)\cdot x(5)\cdot x(10) + x(1)\cdot x(6)\cdot x(10) + x(2)\cdot x(7)\cdot x(10) +} \\
 & & \mathtt{x(3)\cdot x(8)\cdot x(10) + x(4)\cdot x(9)\cdot x(10) + x(4)\cdot x(10) - 5} \\
\mathtt{i[6]} & = & \mathtt{x(0)\cdot x(6)\cdot x(10) + x(1)\cdot x(7)\cdot x(10) + x(2)\cdot x(8)\cdot x(10) +} \\
 & & \mathtt{x(3)\cdot x(9)\cdot x(10) + x(5)\cdot x(10) - 6} \\
\mathtt{i[7]} & = & \mathtt{x(0)\cdot x(7)\cdot x(10) + x(1)\cdot x(8)\cdot x(10) + x(2)\cdot x(9)\cdot x(10) +} \\
 & & \mathtt{x(6)\cdot x(10) - 7} \\
\mathtt{i[8]} & = & \mathtt{x(0)\cdot x(8)\cdot x(10) + x(1)\cdot x(9)\cdot x(10) + x(7)\cdot x(10) - 8} \\
\mathtt{i[9]} & = & \mathtt{x(0)\cdot x(9)\cdot x(10) + x(8)\cdot x(10) - 9} \\
\mathtt{i[10]} & = & \mathtt{x(9)\cdot x(10) - 10} \\
\mathtt{i[11]} & = & \mathtt{x(0) + x(1) + x(2) + x(3) + x(4) + x(5) + x(6) + x(7) + x(8) +} \\
 & & \mathtt{x(9) + 1}
\end{array}
$$

Eco-11-h

Polynomial ring in 12 variables: x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), x(9), x(10), h

$$
\begin{aligned}
\texttt{i[1]} &= x(0)\cdot x(1)\cdot x(10) + x(1)\cdot x(2)\cdot x(10) + x(2)\cdot x(3)\cdot x(10) + \\
&\quad x(3)\cdot x(4)\cdot x(10) + x(4)\cdot x(5)\cdot x(10) + x(5)\cdot x(6)\cdot x(10) + \\
&\quad x(6)\cdot x(7)\cdot x(10) + x(7)\cdot x(8)\cdot x(10) + x(8)\cdot x(9)\cdot x(10) + \\
&\quad x(0)\cdot x(10)\cdot h - h\hat{}3 \\
\texttt{i[2]} &= x(0)\cdot x(2)\cdot x(10) + x(1)\cdot x(3)\cdot x(10) + x(2)\cdot x(4)\cdot x(10) + \\
&\quad x(3)\cdot x(5)\cdot x(10) + x(4)\cdot x(6)\cdot x(10) + x(5)\cdot x(7)\cdot x(10) + \\
&\quad x(6)\cdot x(8)\cdot x(10) + x(7)\cdot x(9)\cdot x(10) + x(1)\cdot x(10)\cdot h - 2\cdot h\hat{}3 \\
\texttt{i[3]} &= x(0)\cdot x(3)\cdot x(10) + x(1)\cdot x(4)\cdot x(10) + x(2)\cdot x(5)\cdot x(10) + \\
&\quad x(3)\cdot x(6)\cdot x(10) + x(4)\cdot x(7)\cdot x(10) + x(5)\cdot x(8)\cdot x(10) + \\
&\quad x(6)\cdot x(9)\cdot x(10) + x(2)\cdot x(10)\cdot h - 3\cdot h\hat{}3 \\
\texttt{i[4]} &= x(0)\cdot x(4)\cdot x(10) + x(1)\cdot x(5)\cdot x(10) + x(2)\cdot x(6)\cdot x(10) + \\
&\quad x(3)\cdot x(7)\cdot x(10) + x(4)\cdot x(8)\cdot x(10) + x(5)\cdot x(9)\cdot x(10) + \\
&\quad x(3)\cdot x(10)\cdot h - 4\cdot h\hat{}3 \\
\texttt{i[5]} &= x(0)\cdot x(5)\cdot x(10) + x(1)\cdot x(6)\cdot x(10) + x(2)\cdot x(7)\cdot x(10) + \\
&\quad x(3)\cdot x(8)\cdot x(10) + x(4)\cdot x(9)\cdot x(10) + x(4)\cdot x(10)\cdot h - 5\cdot h\hat{}3 \\
\texttt{i[6]} &= x(0)\cdot x(6)\cdot x(10) + x(1)\cdot x(7)\cdot x(10) + x(2)\cdot x(8)\cdot x(10) + \\
&\quad x(3)\cdot x(9)\cdot x(10) + x(5)\cdot x(10)\cdot h - 6\cdot h\hat{}3 \\
\texttt{i[7]} &= x(0)\cdot x(7)\cdot x(10) + x(1)\cdot x(8)\cdot x(10) + x(2)\cdot x(9)\cdot x(10) + \\
&\quad x(6)\cdot x(10)\cdot h - 7\cdot h\hat{}3 \\
\texttt{i[8]} &= x(0)\cdot x(8)\cdot x(10) + x(1)\cdot x(9)\cdot x(10) + x(7)\cdot x(10)\cdot h - 8\cdot h\hat{}3 \\
\texttt{i[9]} &= x(0)\cdot x(9)\cdot x(10) + x(8)\cdot x(10)\cdot h - 9\cdot h\hat{}3 \\
\texttt{i[10]} &= x(9)\cdot x(10) - 10\cdot h\hat{}2 \\
\texttt{i[11]} &= x(0) + x(1) + x(2) + x(3) + x(4) + x(5) + x(6) + x(7) + x(8) + \\
&\quad x(9) + h
\end{aligned}
$$

F-633

Polynomial ring in 10 variables: U6, U5, U4, U3, U2, u6, u5, u4, u3, u2

$$
\begin{aligned}
\mathtt{i[1]} &= 2\cdot u6 + 2\cdot u5 + 2\cdot u4 + 2\cdot u3 + 2\cdot u2 + 1\\
\mathtt{i[2]} &= 2\cdot U6 + 2\cdot U5 + 2\cdot U4 + 2\cdot U3 + 2\cdot U2 + 1\\
\mathtt{i[3]} &= 4\cdot U5\cdot u6 + 4\cdot U4\cdot u6 + 4\cdot U3\cdot u6 + 4\cdot U2\cdot u6 - 4\cdot U6\cdot u5 + 4\cdot U4\cdot u5 +\\
&\quad 4\cdot U3\cdot u5 + 4\cdot U2\cdot u5 - 4\cdot U6\cdot u4 - 4\cdot U5\cdot u4 + 4\cdot U3\cdot u4 + 4\cdot U2\cdot u4 -\\
&\quad 4\cdot U6\cdot u3 - 4\cdot U5\cdot u3 - 4\cdot U4\cdot u3 + 4\cdot U2\cdot u3 - 4\cdot U6\cdot u2 - 4\cdot U5\cdot u2 -\\
&\quad 4\cdot U4\cdot u2 - 4\cdot U3\cdot u2 + 2\cdot u6 + 2\cdot u5 + 2\cdot u4 + 2\cdot u3 + 2\cdot u2 + 1\\
\mathtt{i[4]} &= -4\cdot U5\cdot u6 - 4\cdot U4\cdot u6 - 4\cdot U3\cdot u6 - 4\cdot U2\cdot u6 + 4\cdot U6\cdot u5 -\\
&\quad 4\cdot U4\cdot u5 - 4\cdot U3\cdot u5 - 4\cdot U2\cdot u5 + 4\cdot U6\cdot u4 + 4\cdot U5\cdot u4 - 4\cdot U3\cdot u4 -\\
&\quad 4\cdot U2\cdot u4 + 4\cdot U6\cdot u3 + 4\cdot U5\cdot u3 + 4\cdot U4\cdot u3 - 4\cdot U2\cdot u3 + 4\cdot U6\cdot u2 +\\
&\quad 4\cdot U5\cdot u2 + 4\cdot U4\cdot u2 + 4\cdot U3\cdot u2 + 2\cdot U6 + 2\cdot U5 + 2\cdot U4 + 2\cdot U3 + 2\cdot U2 + 1\\
\mathtt{i[5]} &= U2\cdot u2 - 1\\
\mathtt{i[6]} &= U3\cdot u3 - 1\\
\mathtt{i[7]} &= U4\cdot u4 - 1\\
\mathtt{i[8]} &= U5\cdot u5 - 1\\
\mathtt{i[9]} &= U6\cdot u6 - 1
\end{aligned}
$$

F-633-h

Polynomial ring in 11 variables: $\mathtt{U6, U5, U4, U3, U2, u6, u5, u4, u3, u2, h}$

$$
\begin{aligned}
\mathtt{i[1]} &= 2\cdot \mathtt{u6}+2\cdot \mathtt{u5}+2\cdot \mathtt{u4}+2\cdot \mathtt{u3}+2\cdot \mathtt{u2}+\mathtt{h}\\
\mathtt{i[2]} &= 2\cdot \mathtt{U6}+2\cdot \mathtt{U5}+2\cdot \mathtt{U4}+2\cdot \mathtt{U3}+2\cdot \mathtt{U2}+\mathtt{h}\\
\mathtt{i[3]} &= 4\cdot \mathtt{U5}\cdot \mathtt{u6}+4\cdot \mathtt{U4}\cdot \mathtt{u6}+4\cdot \mathtt{U3}\cdot \mathtt{u6}+4\cdot \mathtt{U2}\cdot \mathtt{u6}-4\cdot \mathtt{U6}\cdot \mathtt{u5}+4\cdot \mathtt{U4}\cdot \mathtt{u5}+\\
&\quad 4\cdot \mathtt{U3}\cdot \mathtt{u5}+4\cdot \mathtt{U2}\cdot \mathtt{u5}-4\cdot \mathtt{U6}\cdot \mathtt{u4}-4\cdot \mathtt{U5}\cdot \mathtt{u4}+4\cdot \mathtt{U3}\cdot \mathtt{u4}+4\cdot \mathtt{U2}\cdot \mathtt{u4}-\\
&\quad 4\cdot \mathtt{U6}\cdot \mathtt{u3}-4\cdot \mathtt{U5}\cdot \mathtt{u3}-4\cdot \mathtt{U4}\cdot \mathtt{u3}+4\cdot \mathtt{U2}\cdot \mathtt{u3}-4\cdot \mathtt{U6}\cdot \mathtt{u2}-4\cdot \mathtt{U5}\cdot \mathtt{u2}-\\
&\quad 4\cdot \mathtt{U4}\cdot \mathtt{u2}-4\cdot \mathtt{U3}\cdot \mathtt{u2}+2\cdot \mathtt{u6}\cdot \mathtt{h}+2\cdot \mathtt{u5}\cdot \mathtt{h}+2\cdot \mathtt{u4}\cdot \mathtt{h}+2\cdot \mathtt{u3}\cdot \mathtt{h}+\\
&\quad 2\cdot \mathtt{u2}\cdot \mathtt{h}+\mathtt{h\hat{}2}\\
\mathtt{i[4]} &= -4\cdot \mathtt{U5}\cdot \mathtt{u6}-4\cdot \mathtt{U4}\cdot \mathtt{u6}-4\cdot \mathtt{U3}\cdot \mathtt{u6}-4\cdot \mathtt{U2}\cdot \mathtt{u6}+4\cdot \mathtt{U6}\cdot \mathtt{u5}-\\
&\quad 4\cdot \mathtt{U4}\cdot \mathtt{u5}-4\cdot \mathtt{U3}\cdot \mathtt{u5}-4\cdot \mathtt{U2}\cdot \mathtt{u5}+4\cdot \mathtt{U6}\cdot \mathtt{u4}+4\cdot \mathtt{U5}\cdot \mathtt{u4}-4\cdot \mathtt{U3}\cdot \mathtt{u4}-\\
&\quad 4\cdot \mathtt{U2}\cdot \mathtt{u4}+4\cdot \mathtt{U6}\cdot \mathtt{u3}+4\cdot \mathtt{U5}\cdot \mathtt{u3}+4\cdot \mathtt{U4}\cdot \mathtt{u3}-4\cdot \mathtt{U2}\cdot \mathtt{u3}+4\cdot \mathtt{U6}\cdot \mathtt{u2}+\\
&\quad 4\cdot \mathtt{U5}\cdot \mathtt{u2}+4\cdot \mathtt{U4}\cdot \mathtt{u2}+4\cdot \mathtt{U3}\cdot \mathtt{u2}+2\cdot \mathtt{U6}\cdot \mathtt{h}+2\cdot \mathtt{U5}\cdot \mathtt{h}+2\cdot \mathtt{U4}\cdot \mathtt{h}+\\
&\quad 2\cdot \mathtt{U3}\cdot \mathtt{h}+2\cdot \mathtt{U2}\cdot \mathtt{h}+\mathtt{h\hat{}2}\\
\mathtt{i[5]} &= \mathtt{U2}\cdot \mathtt{u2}-\mathtt{h\hat{}2}\\
\mathtt{i[6]} &= \mathtt{U3}\cdot \mathtt{u3}-\mathtt{h\hat{}2}\\
\mathtt{i[7]} &= \mathtt{U4}\cdot \mathtt{u4}-\mathtt{h\hat{}2}\\
\mathtt{i[8]} &= \mathtt{U5}\cdot \mathtt{u5}-\mathtt{h\hat{}2}\\
\mathtt{i[9]} &= \mathtt{U6}\cdot \mathtt{u6}-\mathtt{h\hat{}2}
\end{aligned}
$$

F-744

Polynomial ring in 12 variables: U7, U6, U5, U4, U3, U2, u7, u6, u5, u4, u3, u2

$$
\begin{aligned}
\mathtt{i[1]} &= 2\cdot u7 + 2\cdot u6 + 2\cdot u5 + 2\cdot u4 + 2\cdot u3 + 2\cdot u2 + 1\\
\mathtt{i[2]} &= 2\cdot U7 + 2\cdot U6 + 2\cdot U5 + 2\cdot U4 + 2\cdot U3 + 2\cdot U2 + 1\\
\mathtt{i[3]} &= 8\cdot U6\cdot u7 + 8\cdot U5\cdot u7 + 8\cdot U4\cdot u7 + 8\cdot U3\cdot u7 + 8\cdot U2\cdot u7 + 8\cdot U6\cdot u6+\\
&\quad 8\cdot U5\cdot u6 + 8\cdot U4\cdot u6 + 8\cdot U3\cdot u6 + 8\cdot U2\cdot u6 + 8\cdot U5\cdot u5 + 8\cdot U4\cdot u5+\\
&\quad 8\cdot U3\cdot u5 + 8\cdot U2\cdot u5 + 8\cdot U4\cdot u4 + 8\cdot U3\cdot u4 + 8\cdot U2\cdot u4 + 8\cdot U3\cdot u3+\\
&\quad 8\cdot U2\cdot u3 + 8\cdot U2\cdot u2 - 17\\
\mathtt{i[4]} &= 8\cdot U7\cdot u6 + 8\cdot U6\cdot u6 + 8\cdot U7\cdot u5 + 8\cdot U6\cdot u5 + 8\cdot U5\cdot u5 + 8\cdot U7\cdot u4+\\
&\quad 8\cdot U6\cdot u4 + 8\cdot U5\cdot u4 + 8\cdot U4\cdot u4 + 8\cdot U7\cdot u3 + 8\cdot U6\cdot u3 + 8\cdot U5\cdot u3+\\
&\quad 8\cdot U4\cdot u3 + 8\cdot U3\cdot u3 + 8\cdot U7\cdot u2 + 8\cdot U6\cdot u2 + 8\cdot U5\cdot u2 + 8\cdot U4\cdot u2+\\
&\quad 8\cdot U3\cdot u2 + 8\cdot U2\cdot u2 - 17\\
\mathtt{i[5]} &= 16\cdot U5\cdot U3\cdot u4 + 16\cdot U5\cdot U2\cdot u4 + 16\cdot U5\cdot U2\cdot u3 + 16\cdot U4\cdot U2\cdot u3+\\
&\quad 8\cdot U5\cdot u4 + 8\cdot U5\cdot u3 + 8\cdot U4\cdot u3 + 8\cdot U5\cdot u2 + 8\cdot U4\cdot u2 + 8\cdot U3\cdot u2+\\
&\quad 18\cdot U5 + 18\cdot U4 + 18\cdot U3 + 18\cdot U2 + 11\\
\mathtt{i[6]} &= 16\cdot U4\cdot u5\cdot u3 + 16\cdot U4\cdot u5\cdot u2 + 16\cdot U3\cdot u5\cdot u2 + 16\cdot U3\cdot u4\cdot u2+\\
&\quad 8\cdot U4\cdot u5 + 8\cdot U3\cdot u5 + 8\cdot U2\cdot u5 + 8\cdot U3\cdot u4 + 8\cdot U2\cdot u4 + 8\cdot U2\cdot u3+\\
&\quad 18\cdot u5 + 18\cdot u4 + 18\cdot u3 + 18\cdot u2 + 11\\
\mathtt{i[7]} &= U2\cdot u2 - 1\\
\mathtt{i[8]} &= U3\cdot u3 - 1\\
\mathtt{i[9]} &= U4\cdot u4 - 1\\
\mathtt{i[10]} &= U5\cdot u5 - 1\\
\mathtt{i[11]} &= U6\cdot u6 - 1\\
\mathtt{i[12]} &= U7\cdot u7 - 1
\end{aligned}
$$

F-744-h

Polynomial ring in 13 variables: U7, U6, U5, U4, U3, U2, u7, u6, u5, u4, u3, u2, h

$$
\begin{aligned}
i[1] &= 2\cdot u7+2\cdot u6+2\cdot u5+2\cdot u4+2\cdot u3+2\cdot u2+h\\
i[2] &= 2\cdot U7+2\cdot U6+2\cdot U5+2\cdot U4+2\cdot U3+2\cdot U2+h\\
i[3] &= 8\cdot U6\cdot u7+8\cdot U5\cdot u7+8\cdot U4\cdot u7+8\cdot U3\cdot u7+8\cdot U2\cdot u7+8\cdot U6\cdot u6+\\
&\quad 8\cdot U5\cdot u6+8\cdot U4\cdot u6+8\cdot U3\cdot u6+8\cdot U2\cdot u6+8\cdot U5\cdot u5+8\cdot U4\cdot u5+\\
&\quad 8\cdot U3\cdot u5+8\cdot U2\cdot u5+8\cdot U4\cdot u4+8\cdot U3\cdot u4+8\cdot U2\cdot u4+8\cdot U3\cdot u3+\\
&\quad 8\cdot U2\cdot u3+8\cdot U2\cdot u2-17\cdot h\hat{}2\\
i[4] &= 8\cdot U7\cdot u6+8\cdot U6\cdot u6+8\cdot U7\cdot u5+8\cdot U6\cdot u5+8\cdot U5\cdot u5+8\cdot U7\cdot u4+\\
&\quad 8\cdot U6\cdot u4+8\cdot U5\cdot u4+8\cdot U4\cdot u4+8\cdot U7\cdot u3+8\cdot U6\cdot u3+8\cdot U5\cdot u3+\\
&\quad 8\cdot U4\cdot u3+8\cdot U3\cdot u3+8\cdot U7\cdot u2+8\cdot U6\cdot u2+8\cdot U5\cdot u2+8\cdot U4\cdot u2+\\
&\quad 8\cdot U3\cdot u2+8\cdot U2\cdot u2-17\cdot h\hat{}2\\
i[5] &= 16\cdot U5\cdot U3\cdot u4+16\cdot U5\cdot U2\cdot u4+16\cdot U5\cdot U2\cdot u3+16\cdot U4\cdot U2\cdot u3+\\
&\quad 8\cdot U5\cdot u4\cdot h+8\cdot U5\cdot u3\cdot h+8\cdot U4\cdot u3\cdot h+8\cdot U5\cdot u2\cdot h+8\cdot U4\cdot u2\cdot h+\\
&\quad 8\cdot U3\cdot u2\cdot h+18\cdot U5\cdot h\hat{}2+18\cdot U4\cdot h\hat{}2+18\cdot U3\cdot h\hat{}2+18\cdot U2\cdot h\hat{}2+\\
&\quad 11\cdot h\hat{}3\\
i[6] &= 16\cdot U4\cdot u5\cdot u3+16\cdot U4\cdot u5\cdot u2+16\cdot U3\cdot u5\cdot u2+16\cdot U3\cdot u4\cdot u2+\\
&\quad 8\cdot U4\cdot u5\cdot h+8\cdot U3\cdot u5\cdot h+8\cdot U2\cdot u5\cdot h+8\cdot U3\cdot u4\cdot h+8\cdot U2\cdot u4\cdot h+\\
&\quad 8\cdot U2\cdot u3\cdot h+18\cdot u5\cdot h\hat{}2+18\cdot u4\cdot h\hat{}2+18\cdot u3\cdot h\hat{}2+18\cdot u2\cdot h\hat{}2+\\
&\quad 11\cdot h\hat{}3\\
i[7] &= U2\cdot u2-h\hat{}2\\
i[8] &= U3\cdot u3-h\hat{}2\\
i[9] &= U4\cdot u4-h\hat{}2\\
i[10] &= U5\cdot u5-h\hat{}2\\
i[11] &= U6\cdot u6-h\hat{}2\\
i[12] &= U7\cdot u7-h\hat{}2
\end{aligned}
$$

F-855

Polynomial ring in 14 variables: U8, U7, U6, U5, U4, U3, U2, u8, u7, u6, u5, u4, u3, u2

$$
\begin{aligned}
\mathtt{i[1]} &= 2\cdot u8+2\cdot u7+2\cdot u6+2\cdot u5+2\cdot u4+2\cdot u3+2\cdot u2+1\\
\mathtt{i[2]} &= 2\cdot U8+2\cdot U7+2\cdot U6+2\cdot U5+2\cdot U4+2\cdot U3+2\cdot U2+1\\
\mathtt{i[3]} &= U2\cdot u2-1\\
\mathtt{i[4]} &= U3\cdot u3-1\\
\mathtt{i[5]} &= U4\cdot u4-1\\
\mathtt{i[6]} &= U5\cdot u5-1\\
\mathtt{i[7]} &= U6\cdot u6-1\\
\mathtt{i[8]} &= U7\cdot u7-1\\
\mathtt{i[9]} &= U8\cdot u8-1\\
\mathtt{i[10]} &= -4\cdot U7\cdot u8-4\cdot U6\cdot u8-4\cdot U5\cdot u8-4\cdot U4\cdot u8-4\cdot U3\cdot u8-\\
&\quad 4\cdot U2\cdot u8+4\cdot U8\cdot u7-4\cdot U6\cdot u7-4\cdot U5\cdot u7-4\cdot U4\cdot u7-4\cdot U3\cdot u7-\\
&\quad 4\cdot U2\cdot u7+4\cdot U8\cdot u6+4\cdot U7\cdot u6-4\cdot U5\cdot u6-4\cdot U4\cdot u6-4\cdot U3\cdot u6-\\
&\quad 4\cdot U2\cdot u6+4\cdot U8\cdot u5+4\cdot U7\cdot u5+4\cdot U6\cdot u5-4\cdot U4\cdot u5-4\cdot U3\cdot u5-\\
&\quad 4\cdot U2\cdot u5+4\cdot U8\cdot u4+4\cdot U7\cdot u4+4\cdot U6\cdot u4+4\cdot U5\cdot u4-4\cdot U3\cdot u4-\\
&\quad 4\cdot U2\cdot u4+4\cdot U8\cdot u3+4\cdot U7\cdot u3+4\cdot U6\cdot u3+4\cdot U5\cdot u3+4\cdot U4\cdot u3-\\
&\quad 4\cdot U2\cdot u3+4\cdot U8\cdot u2+4\cdot U7\cdot u2+4\cdot U6\cdot u2+4\cdot U5\cdot u2+4\cdot U4\cdot u2+\\
&\quad 4\cdot U3\cdot u2+2\cdot U8+2\cdot U7+2\cdot U6+2\cdot U5+2\cdot U4+2\cdot U3+2\cdot U2+1\\
\mathtt{i[11]} &= 4\cdot U7\cdot u8+4\cdot U6\cdot u8+4\cdot U5\cdot u8+4\cdot U4\cdot u8+4\cdot U3\cdot u8+4\cdot U2\cdot u8-\\
&\quad 4\cdot U8\cdot u7+4\cdot U6\cdot u7+4\cdot U5\cdot u7+4\cdot U4\cdot u7+4\cdot U3\cdot u7+4\cdot U2\cdot u7-\\
&\quad 4\cdot U8\cdot u6-4\cdot U7\cdot u6+4\cdot U5\cdot u6+4\cdot U4\cdot u6+4\cdot U3\cdot u6+4\cdot U2\cdot u6-\\
&\quad 4\cdot U8\cdot u5-4\cdot U7\cdot u5-4\cdot U6\cdot u5+4\cdot U4\cdot u5+4\cdot U3\cdot u5+4\cdot U2\cdot u5-\\
&\quad 4\cdot U8\cdot u4-4\cdot U7\cdot u4-4\cdot U6\cdot u4-4\cdot U5\cdot u4+4\cdot U3\cdot u4+4\cdot U2\cdot u4-\\
&\quad 4\cdot U8\cdot u3-4\cdot U7\cdot u3-4\cdot U6\cdot u3-4\cdot U5\cdot u3-4\cdot U4\cdot u3+4\cdot U2\cdot u3-\\
&\quad 4\cdot U8\cdot u2-4\cdot U7\cdot u2-4\cdot U6\cdot u2-4\cdot U5\cdot u2-4\cdot U4\cdot u2-4\cdot U3\cdot u2+\\
&\quad 2\cdot u8+2\cdot u7+2\cdot u6+2\cdot u5+2\cdot u4+2\cdot u3+2\cdot u2+1\\
\mathtt{i[12]} &= 16\cdot U6\cdot U4\cdot u5+16\cdot U6\cdot U3\cdot u5+16\cdot U6\cdot U2\cdot u5+16\cdot U6\cdot U3\cdot u4+\\
&\quad 16\cdot U5\cdot U3\cdot u4+16\cdot U6\cdot U2\cdot u4+16\cdot U5\cdot U2\cdot u4+16\cdot U6\cdot U2\cdot u3+\\
&\quad 16\cdot U5\cdot U2\cdot u3+16\cdot U4\cdot U2\cdot u3+8\cdot U6\cdot u5+8\cdot U6\cdot u4+8\cdot U5\cdot u4+\\
&\quad 8\cdot U6\cdot u3+8\cdot U5\cdot u3+8\cdot U4\cdot u3+8\cdot U6\cdot u2+8\cdot U5\cdot u2+8\cdot U4\cdot u2+\\
&\quad 8\cdot U3\cdot u2+26\cdot U6+26\cdot U5+26\cdot U4+26\cdot U3+26\cdot U2+15\\
\mathtt{i[13]} &= 16\cdot U5\cdot u6\cdot u4+16\cdot U5\cdot u6\cdot u3+16\cdot U4\cdot u6\cdot u3+16\cdot U4\cdot u5\cdot u3+\\
&\quad 16\cdot U5\cdot u6\cdot u2+16\cdot U4\cdot u6\cdot u2+16\cdot U3\cdot u6\cdot u2+16\cdot U4\cdot u5\cdot u2+\\
&\quad 16\cdot U3\cdot u5\cdot u2+16\cdot U3\cdot u4\cdot u2+8\cdot U5\cdot u6+8\cdot U4\cdot u6+8\cdot U3\cdot u6+\\
&\quad 8\cdot U2\cdot u6+8\cdot U4\cdot u5+8\cdot U3\cdot u5+8\cdot U2\cdot u5+8\cdot U3\cdot u4+8\cdot U2\cdot u4+\\
&\quad 8\cdot U2\cdot u3+26\cdot u6+26\cdot u5+26\cdot u4+26\cdot u3+26\cdot u2+15
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{i}[14] \quad = \quad & -2\cdot U7\cdot U5\cdot u8\cdot u6 - 2\cdot U7\cdot U4\cdot u8\cdot u6 - 2\cdot U7\cdot U3\cdot u8\cdot u6- \\
& 2\cdot U7\cdot U2\cdot u8\cdot u6 - 2\cdot U7\cdot U4\cdot u8\cdot u5 - 2\cdot U6\cdot U4\cdot u8\cdot u5- \\
& 2\cdot U7\cdot U3\cdot u8\cdot u5 - 2\cdot U6\cdot U3\cdot u8\cdot u5 - 2\cdot U7\cdot U2\cdot u8\cdot u5- \\
& 2\cdot U6\cdot U2\cdot u8\cdot u5 + 2\cdot U8\cdot U6\cdot u7\cdot u5 - 2\cdot U6\cdot U4\cdot u7\cdot u5- \\
& 2\cdot U6\cdot U3\cdot u7\cdot u5 - 2\cdot U6\cdot U2\cdot u7\cdot u5 - 2\cdot U7\cdot U3\cdot u8\cdot u4- \\
& 2\cdot U6\cdot U3\cdot u8\cdot u4 - 2\cdot U5\cdot U3\cdot u8\cdot u4 - 2\cdot U7\cdot U2\cdot u8\cdot u4- \\
& 2\cdot U6\cdot U2\cdot u8\cdot u4 - 2\cdot U5\cdot U2\cdot u8\cdot u4 + 2\cdot U8\cdot U6\cdot u7\cdot u4+ \\
& 2\cdot U8\cdot U5\cdot u7\cdot u4 - 2\cdot U6\cdot U3\cdot u7\cdot u4 - 2\cdot U5\cdot U3\cdot u7\cdot u4- \\
& 2\cdot U6\cdot U2\cdot u7\cdot u4 - 2\cdot U5\cdot U2\cdot u7\cdot u4 + 2\cdot U8\cdot U5\cdot u6\cdot u4+ \\
& 2\cdot U7\cdot U5\cdot u6\cdot u4 - 2\cdot U5\cdot U3\cdot u6\cdot u4 - 2\cdot U5\cdot U2\cdot u6\cdot u4- \\
& 2\cdot U7\cdot U2\cdot u8\cdot u3 - 2\cdot U6\cdot U2\cdot u8\cdot u3 - 2\cdot U5\cdot U2\cdot u8\cdot u3- \\
& 2\cdot U4\cdot U2\cdot u8\cdot u3 + 2\cdot U8\cdot U6\cdot u7\cdot u3 + 2\cdot U8\cdot U5\cdot u7\cdot u3+ \\
& 2\cdot U8\cdot U4\cdot u7\cdot u3 - 2\cdot U6\cdot U2\cdot u7\cdot u3 - 2\cdot U5\cdot U2\cdot u7\cdot u3- \\
& 2\cdot U4\cdot U2\cdot u7\cdot u3 + 2\cdot U8\cdot U5\cdot u6\cdot u3 + 2\cdot U7\cdot U5\cdot u6\cdot u3+ \\
& 2\cdot U8\cdot U4\cdot u6\cdot u3 + 2\cdot U7\cdot U4\cdot u6\cdot u3 - 2\cdot U5\cdot U2\cdot u6\cdot u3- \\
& 2\cdot U4\cdot U2\cdot u6\cdot u3 + 2\cdot U8\cdot U4\cdot u5\cdot u3 + 2\cdot U7\cdot U4\cdot u5\cdot u3+ \\
& 2\cdot U6\cdot U4\cdot u5\cdot u3 - 2\cdot U4\cdot U2\cdot u5\cdot u3 + 2\cdot U8\cdot U6\cdot u7\cdot u2+ \\
& 2\cdot U8\cdot U5\cdot u7\cdot u2 + 2\cdot U8\cdot U4\cdot u7\cdot u2 + 2\cdot U8\cdot U3\cdot u7\cdot u2+ \\
& 2\cdot U8\cdot U5\cdot u6\cdot u2 + 2\cdot U7\cdot U5\cdot u6\cdot u2 + 2\cdot U8\cdot U4\cdot u6\cdot u2+ \\
& 2\cdot U7\cdot U4\cdot u6\cdot u2 + 2\cdot U8\cdot U3\cdot u6\cdot u2 + 2\cdot U7\cdot U3\cdot u6\cdot u2+ \\
& 2\cdot U8\cdot U4\cdot u5\cdot u2 + 2\cdot U7\cdot U4\cdot u5\cdot u2 + 2\cdot U6\cdot U4\cdot u5\cdot u2+ \\
& 2\cdot U8\cdot U3\cdot u5\cdot u2 + 2\cdot U7\cdot U3\cdot u5\cdot u2 + 2\cdot U6\cdot U3\cdot u5\cdot u2+ \\
& 2\cdot U8\cdot U3\cdot u4\cdot u2 + 2\cdot U7\cdot U3\cdot u4\cdot u2 + 2\cdot U6\cdot U3\cdot u4\cdot u2+ \\
& 2\cdot U5\cdot U3\cdot u4\cdot u2 + U8\cdot U6\cdot u7 + U8\cdot U5\cdot u7 + U8\cdot U4\cdot u7+ \\
& U8\cdot U3\cdot u7 + U8\cdot U2\cdot u7 + U8\cdot U5\cdot u6 + U7\cdot U5\cdot u6 + U8\cdot U4\cdot u6+ \\
& U7\cdot U4\cdot u6 + U8\cdot U3\cdot u6 + U7\cdot U3\cdot u6 + U8\cdot U2\cdot u6 + U7\cdot U2\cdot u6- \\
& U7\cdot u8\cdot u6 + U8\cdot U4\cdot u5 + U7\cdot U4\cdot u5 + U6\cdot U4\cdot u5 + U8\cdot U3\cdot u5+ \\
& U7\cdot U3\cdot u5 + U6\cdot U3\cdot u5 + U8\cdot U2\cdot u5 + U7\cdot U2\cdot u5 + U6\cdot U2\cdot u5- \\
& U7\cdot u8\cdot u5 - U6\cdot u8\cdot u5 - U6\cdot u7\cdot u5 + U8\cdot U3\cdot u4 + U7\cdot U3\cdot u4+ \\
& U6\cdot U3\cdot u4 + U5\cdot U3\cdot u4 + U8\cdot U2\cdot u4 + U7\cdot U2\cdot u4 + U6\cdot U2\cdot u4+ \\
& U5\cdot U2\cdot u4 - U7\cdot u8\cdot u4 - U6\cdot u8\cdot u4 - U5\cdot u8\cdot u4 - U6\cdot u7\cdot u4- \\
& U5\cdot u7\cdot u4 - U5\cdot u6\cdot u4 + U8\cdot U2\cdot u3 + U7\cdot U2\cdot u3 + U6\cdot U2\cdot u3+ \\
& U5\cdot U2\cdot u3 + U4\cdot U2\cdot u3 - U7\cdot u8\cdot u3 - U6\cdot u8\cdot u3 - U5\cdot u8\cdot u3- \\
& U4\cdot u8\cdot u3 - U6\cdot u7\cdot u3 - U5\cdot u7\cdot u3 - U4\cdot u7\cdot u3 - U5\cdot u6\cdot u3- \\
& U4\cdot u6\cdot u3 - U4\cdot u5\cdot u3 - U7\cdot u8\cdot u2 - U6\cdot u8\cdot u2 - U5\cdot u8\cdot u2- \\
& U4\cdot u8\cdot u2 - U3\cdot u8\cdot u2 - U6\cdot u7\cdot u2 - U5\cdot u7\cdot u2 - U4\cdot u7\cdot u2- \\
& U3\cdot u7\cdot u2 - U5\cdot u6\cdot u2 - U4\cdot u6\cdot u2 - U3\cdot u6\cdot u2 - U4\cdot u5\cdot u2- \\
& U3\cdot u5\cdot u2 - U3\cdot u4\cdot u2
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{i[15]} \quad = \quad & 2 \cdot U7 \cdot U5 \cdot u8 \cdot u6 + 2 \cdot U7 \cdot U4 \cdot u8 \cdot u6 + 2 \cdot U7 \cdot U3 \cdot u8 \cdot u6 + \\
& 2 \cdot U7 \cdot U2 \cdot u8 \cdot u6 + 2 \cdot U7 \cdot U4 \cdot u8 \cdot u5 + 2 \cdot U6 \cdot U4 \cdot u8 \cdot u5 + \\
& 2 \cdot U7 \cdot U3 \cdot u8 \cdot u5 + 2 \cdot U6 \cdot U3 \cdot u8 \cdot u5 + 2 \cdot U7 \cdot U2 \cdot u8 \cdot u5 + \\
& 2 \cdot U6 \cdot U2 \cdot u8 \cdot u5 - 2 \cdot U8 \cdot U6 \cdot u7 \cdot u5 + 2 \cdot U6 \cdot U4 \cdot u7 \cdot u5 + \\
& 2 \cdot U6 \cdot U3 \cdot u7 \cdot u5 + 2 \cdot U6 \cdot U2 \cdot u7 \cdot u5 + 2 \cdot U7 \cdot U3 \cdot u8 \cdot u4 + \\
& 2 \cdot U6 \cdot U3 \cdot u8 \cdot u4 + 2 \cdot U5 \cdot U3 \cdot u8 \cdot u4 + 2 \cdot U7 \cdot U2 \cdot u8 \cdot u4 + \\
& 2 \cdot U6 \cdot U2 \cdot u8 \cdot u4 + 2 \cdot U5 \cdot U2 \cdot u8 \cdot u4 - 2 \cdot U8 \cdot U6 \cdot u7 \cdot u4 - \\
& 2 \cdot U8 \cdot U5 \cdot u7 \cdot u4 + 2 \cdot U6 \cdot U3 \cdot u7 \cdot u4 + 2 \cdot U5 \cdot U3 \cdot u7 \cdot u4 + \\
& 2 \cdot U6 \cdot U2 \cdot u7 \cdot u4 + 2 \cdot U5 \cdot U2 \cdot u7 \cdot u4 - 2 \cdot U8 \cdot U5 \cdot u6 \cdot u4 - \\
& 2 \cdot U7 \cdot U5 \cdot u6 \cdot u4 + 2 \cdot U5 \cdot U3 \cdot u6 \cdot u4 + 2 \cdot U5 \cdot U2 \cdot u6 \cdot u4 + \\
& 2 \cdot U7 \cdot U2 \cdot u8 \cdot u3 + 2 \cdot U6 \cdot U2 \cdot u8 \cdot u3 + 2 \cdot U5 \cdot U2 \cdot u8 \cdot u3 + \\
& 2 \cdot U4 \cdot U2 \cdot u8 \cdot u3 - 2 \cdot U8 \cdot U6 \cdot u7 \cdot u3 - 2 \cdot U8 \cdot U5 \cdot u7 \cdot u3 - \\
& 2 \cdot U8 \cdot U4 \cdot u7 \cdot u3 + 2 \cdot U6 \cdot U2 \cdot u7 \cdot u3 + 2 \cdot U5 \cdot U2 \cdot u7 \cdot u3 + \\
& 2 \cdot U4 \cdot U2 \cdot u7 \cdot u3 - 2 \cdot U8 \cdot U5 \cdot u6 \cdot u3 - 2 \cdot U7 \cdot U5 \cdot u6 \cdot u3 - \\
& 2 \cdot U8 \cdot U4 \cdot u6 \cdot u3 - 2 \cdot U7 \cdot U4 \cdot u6 \cdot u3 + 2 \cdot U5 \cdot U2 \cdot u6 \cdot u3 + \\
& 2 \cdot U4 \cdot U2 \cdot u6 \cdot u3 - 2 \cdot U8 \cdot U4 \cdot u5 \cdot u3 - 2 \cdot U7 \cdot U4 \cdot u5 \cdot u3 - \\
& 2 \cdot U6 \cdot U4 \cdot u5 \cdot u3 + 2 \cdot U4 \cdot U2 \cdot u5 \cdot u3 - 2 \cdot U8 \cdot U6 \cdot u7 \cdot u2 - \\
& 2 \cdot U8 \cdot U5 \cdot u7 \cdot u2 - 2 \cdot U8 \cdot U4 \cdot u7 \cdot u2 - 2 \cdot U8 \cdot U3 \cdot u7 \cdot u2 - \\
& 2 \cdot U8 \cdot U5 \cdot u6 \cdot u2 - 2 \cdot U7 \cdot U5 \cdot u6 \cdot u2 - 2 \cdot U8 \cdot U4 \cdot u6 \cdot u2 - \\
& 2 \cdot U7 \cdot U4 \cdot u6 \cdot u2 - 2 \cdot U8 \cdot U3 \cdot u6 \cdot u2 - 2 \cdot U7 \cdot U3 \cdot u6 \cdot u2 - \\
& 2 \cdot U8 \cdot U4 \cdot u5 \cdot u2 - 2 \cdot U7 \cdot U4 \cdot u5 \cdot u2 - 2 \cdot U6 \cdot U4 \cdot u5 \cdot u2 - \\
& 2 \cdot U8 \cdot U3 \cdot u5 \cdot u2 - 2 \cdot U7 \cdot U3 \cdot u5 \cdot u2 - 2 \cdot U6 \cdot U3 \cdot u5 \cdot u2 - \\
& 2 \cdot U8 \cdot U3 \cdot u4 \cdot u2 - 2 \cdot U7 \cdot U3 \cdot u4 \cdot u2 - 2 \cdot U6 \cdot U3 \cdot u4 \cdot u2 - \\
& 2 \cdot U5 \cdot U3 \cdot u4 \cdot u2 - U8 \cdot U6 \cdot u7 - U8 \cdot U5 \cdot u7 - U8 \cdot U4 \cdot u7 - \\
& U8 \cdot U3 \cdot u7 - U8 \cdot U2 \cdot u7 - U8 \cdot U5 \cdot u6 - U7 \cdot U5 \cdot u6 - U8 \cdot U4 \cdot u6 - \\
& U7 \cdot U4 \cdot u6 - U8 \cdot U3 \cdot u6 - U7 \cdot U3 \cdot u6 - U8 \cdot U2 \cdot u6 - U7 \cdot U2 \cdot u6 + \\
& U7 \cdot u8 \cdot u6 - U8 \cdot U4 \cdot u5 - U7 \cdot U4 \cdot u5 - U6 \cdot U4 \cdot u5 - U8 \cdot U3 \cdot u5 - \\
& U7 \cdot U3 \cdot u5 - U6 \cdot U3 \cdot u5 - U8 \cdot U2 \cdot u5 - U7 \cdot U2 \cdot u5 - U6 \cdot U2 \cdot u5 + \\
& U7 \cdot u8 \cdot u5 + U6 \cdot u8 \cdot u5 + U6 \cdot u7 \cdot u5 - U8 \cdot U3 \cdot u4 - U7 \cdot U3 \cdot u4 - \\
& U6 \cdot U3 \cdot u4 - U5 \cdot U3 \cdot u4 - U8 \cdot U2 \cdot u4 - U7 \cdot U2 \cdot u4 - U6 \cdot U2 \cdot u4 - \\
& U5 \cdot U2 \cdot u4 + U7 \cdot u8 \cdot u4 + U6 \cdot u8 \cdot u4 + U5 \cdot u8 \cdot u4 + U6 \cdot u7 \cdot u4 + \\
& U5 \cdot u7 \cdot u4 + U5 \cdot u6 \cdot u4 - U8 \cdot U2 \cdot u3 - U7 \cdot U2 \cdot u3 - U6 \cdot U2 \cdot u3 - \\
& U5 \cdot U2 \cdot u3 - U4 \cdot U2 \cdot u3 + U7 \cdot u8 \cdot u3 + U6 \cdot u8 \cdot u3 + U5 \cdot u8 \cdot u3 + \\
& U4 \cdot u8 \cdot u3 + U6 \cdot u7 \cdot u3 + U5 \cdot u7 \cdot u3 + U4 \cdot u7 \cdot u3 + U5 \cdot u6 \cdot u3 + \\
& U4 \cdot u6 \cdot u3 + U4 \cdot u5 \cdot u3 + U7 \cdot u8 \cdot u2 + U6 \cdot u8 \cdot u2 + U5 \cdot u8 \cdot u2 + \\
& U4 \cdot u8 \cdot u2 + U3 \cdot u8 \cdot u2 + U6 \cdot u7 \cdot u2 + U5 \cdot u7 \cdot u2 + U4 \cdot u7 \cdot u2 + \\
& U3 \cdot u7 \cdot u2 + U5 \cdot u6 \cdot u2 + U4 \cdot u6 \cdot u2 + U3 \cdot u6 \cdot u2 + U4 \cdot u5 \cdot u2 + \\
& U3 \cdot u5 \cdot u2 + U3 \cdot u4 \cdot u2
\end{aligned}
$$

F-855-h

Polynomial ring in 15 variables: U8, U7, U6, U5, U4, U3, U2, u8, u7, u6, u5, u4, u3, u2, h

```
i[1]  = 2·u8 + 2·u7 + 2·u6 + 2·u5 + 2·u4 + 2·u3 + 2·u2 + h
i[2]  = 2·U8 + 2·U7 + 2·U6 + 2·U5 + 2·U4 + 2·U3 + 2·U2 + h
i[3]  = U2·u2 − hˆ2
i[4]  = U3·u3 − hˆ2
i[5]  = U4·u4 − hˆ2
i[6]  = U5·u5 − hˆ2
i[7]  = U6·u6 − hˆ2
i[8]  = U7·u7 − hˆ2
i[9]  = U8·u8 − hˆ2
i[10] = −4·U7·u8 − 4·U6·u8 − 4·U5·u8 − 4·U4·u8 − 4·U3·u8−
        4·U2·u8 + 4·U8·u7 − 4·U6·u7 − 4·U5·u7 − 4·U4·u7 − 4·U3·u7−
        4·U2·u7 + 4·U8·u6 + 4·U7·u6 − 4·U5·u6 − 4·U4·u6 − 4·U3·u6−
        4·U2·u6 + 4·U8·u5 + 4·U7·u5 + 4·U6·u5 − 4·U4·u5 − 4·U3·u5−
        4·U2·u5 + 4·U8·u4 + 4·U7·u4 + 4·U6·u4 + 4·U5·u4 − 4·U3·u4−
        4·U2·u4 + 4·U8·u3 + 4·U7·u3 + 4·U6·u3 + 4·U5·u3 + 4·U4·u3−
        4·U2·u3 + 4·U8·u2 + 4·U7·u2 + 4·U6·u2 + 4·U5·u2 + 4·U4·u2+
        4·U3·u2 + 2·U8·h + 2·U7·h + 2·U6·h + 2·U5·h + 2·U4·h+
        2·U3·h + 2·U2·h + hˆ2
i[11] = 4·U7·u8 + 4·U6·u8 + 4·U5·u8 + 4·U4·u8 + 4·U3·u8 + 4·U2·u8−
        4·U8·u7 + 4·U6·u7 + 4·U5·u7 + 4·U4·u7 + 4·U3·u7 + 4·U2·u7−
        4·U8·u6 − 4·U7·u6 + 4·U5·u6 + 4·U4·u6 + 4·U3·u6 + 4·U2·u6−
        4·U8·u5 − 4·U7·u5 − 4·U6·u5 + 4·U4·u5 + 4·U3·u5 + 4·U2·u5−
        4·U8·u4 − 4·U7·u4 − 4·U6·u4 − 4·U5·u4 + 4·U3·u4 + 4·U2·u4−
        4·U8·u3 − 4·U7·u3 − 4·U6·u3 − 4·U5·u3 − 4·U4·u3 + 4·U2·u3−
        4·U8·u2 − 4·U7·u2 − 4·U6·u2 − 4·U5·u2 − 4·U4·u2 − 4·U3·u2+
        2·u8·h + 2·u7·h + 2·u6·h + 2·u5·h + 2·u4·h + 2·u3·h+
        2·u2·h + hˆ2
i[12] = 16·U6·U4·u5 + 16·U6·U3·u5 + 16·U6·U2·u5 + 16·U6·U3·u4+
        16·U5·U3·u4 + 16·U6·U2·u4 + 16·U5·U2·u4 + 16·U6·U2·u3+
        16·U5·U2·u3 + 16·U4·U2·u3 + 8·U6·u5·h + 8·U6·u4·h+
        8·U5·u4·h + 8·U6·u3·h + 8·U5·u3·h + 8·U4·u3·h + 8·U6·u2·h+
        8·U5·u2·h + 8·U4·u2·h + 8·U3·u2·h + 26·U6·hˆ2 + 26·U5·hˆ2+
        26·U4·hˆ2 + 26·U3·hˆ2 + 26·U2·hˆ2 + 15·hˆ3
i[13] = 16·U5·u6·u4 + 16·U5·u6·u3 + 16·U4·u6·u3 + 16·U4·u5·u3+
        16·U5·u6·u2 + 16·U4·u6·u2 + 16·U3·u6·u2 + 16·U4·u5·u2+
        16·U3·u5·u2 + 16·U3·u4·u2 + 8·U5·u6·h + 8·U4·u6·h+
        8·U3·u6·h + 8·U2·u6·h + 8·U4·u5·h + 8·U3·u5·h + 8·U2·u5·h+
        8·U3·u4·h + 8·U2·u4·h + 8·U2·u3·h + 26·u6·hˆ2 + 26·u5·hˆ2+
        26·u4·hˆ2 + 26·u3·hˆ2 + 26·u2·hˆ2 + 15·hˆ3
```

$$
\begin{aligned}
\mathtt{i[14]} \quad = \quad & -2 \cdot \mathtt{U7} \cdot \mathtt{U5} \cdot \mathtt{u8} \cdot \mathtt{u6} - 2 \cdot \mathtt{U7} \cdot \mathtt{U4} \cdot \mathtt{u8} \cdot \mathtt{u6} - 2 \cdot \mathtt{U7} \cdot \mathtt{U3} \cdot \mathtt{u8} \cdot \mathtt{u6} - \\
& 2 \cdot \mathtt{U7} \cdot \mathtt{U2} \cdot \mathtt{u8} \cdot \mathtt{u6} - 2 \cdot \mathtt{U7} \cdot \mathtt{U4} \cdot \mathtt{u8} \cdot \mathtt{u5} - 2 \cdot \mathtt{U6} \cdot \mathtt{U4} \cdot \mathtt{u8} \cdot \mathtt{u5} - \\
& 2 \cdot \mathtt{U7} \cdot \mathtt{U3} \cdot \mathtt{u8} \cdot \mathtt{u5} - 2 \cdot \mathtt{U6} \cdot \mathtt{U3} \cdot \mathtt{u8} \cdot \mathtt{u5} - 2 \cdot \mathtt{U7} \cdot \mathtt{U2} \cdot \mathtt{u8} \cdot \mathtt{u5} - \\
& 2 \cdot \mathtt{U6} \cdot \mathtt{U2} \cdot \mathtt{u8} \cdot \mathtt{u5} + 2 \cdot \mathtt{U8} \cdot \mathtt{U6} \cdot \mathtt{u7} \cdot \mathtt{u5} - 2 \cdot \mathtt{U6} \cdot \mathtt{U4} \cdot \mathtt{u7} \cdot \mathtt{u5} - \\
& 2 \cdot \mathtt{U6} \cdot \mathtt{U3} \cdot \mathtt{u7} \cdot \mathtt{u5} - 2 \cdot \mathtt{U6} \cdot \mathtt{U2} \cdot \mathtt{u7} \cdot \mathtt{u5} - 2 \cdot \mathtt{U7} \cdot \mathtt{U3} \cdot \mathtt{u8} \cdot \mathtt{u4} - \\
& 2 \cdot \mathtt{U6} \cdot \mathtt{U3} \cdot \mathtt{u8} \cdot \mathtt{u4} - 2 \cdot \mathtt{U5} \cdot \mathtt{U3} \cdot \mathtt{u8} \cdot \mathtt{u4} - 2 \cdot \mathtt{U7} \cdot \mathtt{U2} \cdot \mathtt{u8} \cdot \mathtt{u4} - \\
& 2 \cdot \mathtt{U6} \cdot \mathtt{U2} \cdot \mathtt{u8} \cdot \mathtt{u4} - 2 \cdot \mathtt{U5} \cdot \mathtt{U2} \cdot \mathtt{u8} \cdot \mathtt{u4} + 2 \cdot \mathtt{U8} \cdot \mathtt{U6} \cdot \mathtt{u7} \cdot \mathtt{u4} + \\
& 2 \cdot \mathtt{U8} \cdot \mathtt{U5} \cdot \mathtt{u7} \cdot \mathtt{u4} - 2 \cdot \mathtt{U6} \cdot \mathtt{U3} \cdot \mathtt{u7} \cdot \mathtt{u4} - 2 \cdot \mathtt{U5} \cdot \mathtt{U3} \cdot \mathtt{u7} \cdot \mathtt{u4} - \\
& 2 \cdot \mathtt{U6} \cdot \mathtt{U2} \cdot \mathtt{u7} \cdot \mathtt{u4} - 2 \cdot \mathtt{U5} \cdot \mathtt{U2} \cdot \mathtt{u7} \cdot \mathtt{u4} + 2 \cdot \mathtt{U8} \cdot \mathtt{U5} \cdot \mathtt{u6} \cdot \mathtt{u4} + \\
& 2 \cdot \mathtt{U7} \cdot \mathtt{U5} \cdot \mathtt{u6} \cdot \mathtt{u4} - 2 \cdot \mathtt{U5} \cdot \mathtt{U3} \cdot \mathtt{u6} \cdot \mathtt{u4} - 2 \cdot \mathtt{U5} \cdot \mathtt{U2} \cdot \mathtt{u6} \cdot \mathtt{u4} - \\
& 2 \cdot \mathtt{U7} \cdot \mathtt{U2} \cdot \mathtt{u8} \cdot \mathtt{u3} - 2 \cdot \mathtt{U6} \cdot \mathtt{U2} \cdot \mathtt{u8} \cdot \mathtt{u3} - 2 \cdot \mathtt{U5} \cdot \mathtt{U2} \cdot \mathtt{u8} \cdot \mathtt{u3} - \\
& 2 \cdot \mathtt{U4} \cdot \mathtt{U2} \cdot \mathtt{u8} \cdot \mathtt{u3} + 2 \cdot \mathtt{U8} \cdot \mathtt{U6} \cdot \mathtt{u7} \cdot \mathtt{u3} + 2 \cdot \mathtt{U8} \cdot \mathtt{U5} \cdot \mathtt{u7} \cdot \mathtt{u3} + \\
& 2 \cdot \mathtt{U8} \cdot \mathtt{U4} \cdot \mathtt{u7} \cdot \mathtt{u3} - 2 \cdot \mathtt{U6} \cdot \mathtt{U2} \cdot \mathtt{u7} \cdot \mathtt{u3} - 2 \cdot \mathtt{U5} \cdot \mathtt{U2} \cdot \mathtt{u7} \cdot \mathtt{u3} - \\
& 2 \cdot \mathtt{U4} \cdot \mathtt{U2} \cdot \mathtt{u7} \cdot \mathtt{u3} + 2 \cdot \mathtt{U8} \cdot \mathtt{U5} \cdot \mathtt{u6} \cdot \mathtt{u3} + 2 \cdot \mathtt{U7} \cdot \mathtt{U5} \cdot \mathtt{u6} \cdot \mathtt{u3} + \\
& 2 \cdot \mathtt{U8} \cdot \mathtt{U4} \cdot \mathtt{u6} \cdot \mathtt{u3} + 2 \cdot \mathtt{U7} \cdot \mathtt{U4} \cdot \mathtt{u6} \cdot \mathtt{u3} - 2 \cdot \mathtt{U5} \cdot \mathtt{U2} \cdot \mathtt{u6} \cdot \mathtt{u3} - \\
& 2 \cdot \mathtt{U4} \cdot \mathtt{U2} \cdot \mathtt{u6} \cdot \mathtt{u3} + 2 \cdot \mathtt{U8} \cdot \mathtt{U4} \cdot \mathtt{u5} \cdot \mathtt{u3} + 2 \cdot \mathtt{U7} \cdot \mathtt{U4} \cdot \mathtt{u5} \cdot \mathtt{u3} + \\
& 2 \cdot \mathtt{U6} \cdot \mathtt{U4} \cdot \mathtt{u5} \cdot \mathtt{u3} - 2 \cdot \mathtt{U4} \cdot \mathtt{U2} \cdot \mathtt{u5} \cdot \mathtt{u3} + 2 \cdot \mathtt{U8} \cdot \mathtt{U6} \cdot \mathtt{u7} \cdot \mathtt{u2} + \\
& 2 \cdot \mathtt{U8} \cdot \mathtt{U5} \cdot \mathtt{u7} \cdot \mathtt{u2} + 2 \cdot \mathtt{U8} \cdot \mathtt{U4} \cdot \mathtt{u7} \cdot \mathtt{u2} + 2 \cdot \mathtt{U8} \cdot \mathtt{U3} \cdot \mathtt{u7} \cdot \mathtt{u2} + \\
& 2 \cdot \mathtt{U8} \cdot \mathtt{U5} \cdot \mathtt{u6} \cdot \mathtt{u2} + 2 \cdot \mathtt{U7} \cdot \mathtt{U5} \cdot \mathtt{u6} \cdot \mathtt{u2} + 2 \cdot \mathtt{U8} \cdot \mathtt{U4} \cdot \mathtt{u6} \cdot \mathtt{u2} + \\
& 2 \cdot \mathtt{U7} \cdot \mathtt{U4} \cdot \mathtt{u6} \cdot \mathtt{u2} + 2 \cdot \mathtt{U8} \cdot \mathtt{U3} \cdot \mathtt{u6} \cdot \mathtt{u2} + 2 \cdot \mathtt{U7} \cdot \mathtt{U3} \cdot \mathtt{u6} \cdot \mathtt{u2} + \\
& 2 \cdot \mathtt{U8} \cdot \mathtt{U4} \cdot \mathtt{u5} \cdot \mathtt{u2} + 2 \cdot \mathtt{U7} \cdot \mathtt{U4} \cdot \mathtt{u5} \cdot \mathtt{u2} + 2 \cdot \mathtt{U6} \cdot \mathtt{U4} \cdot \mathtt{u5} \cdot \mathtt{u2} + \\
& 2 \cdot \mathtt{U8} \cdot \mathtt{U3} \cdot \mathtt{u5} \cdot \mathtt{u2} + 2 \cdot \mathtt{U7} \cdot \mathtt{U3} \cdot \mathtt{u5} \cdot \mathtt{u2} + 2 \cdot \mathtt{U6} \cdot \mathtt{U3} \cdot \mathtt{u5} \cdot \mathtt{u2} + \\
& 2 \cdot \mathtt{U8} \cdot \mathtt{U3} \cdot \mathtt{u4} \cdot \mathtt{u2} + 2 \cdot \mathtt{U7} \cdot \mathtt{U3} \cdot \mathtt{u4} \cdot \mathtt{u2} + 2 \cdot \mathtt{U6} \cdot \mathtt{U3} \cdot \mathtt{u4} \cdot \mathtt{u2} + \\
& 2 \cdot \mathtt{U5} \cdot \mathtt{U3} \cdot \mathtt{u4} \cdot \mathtt{u2} + \mathtt{U8} \cdot \mathtt{U6} \cdot \mathtt{u7} \cdot \mathtt{h} + \mathtt{U8} \cdot \mathtt{U5} \cdot \mathtt{u7} \cdot \mathtt{h} + \mathtt{U8} \cdot \mathtt{U4} \cdot \mathtt{u7} \cdot \mathtt{h} + \\
& \mathtt{U8} \cdot \mathtt{U3} \cdot \mathtt{u7} \cdot \mathtt{h} + \mathtt{U8} \cdot \mathtt{U2} \cdot \mathtt{u7} \cdot \mathtt{h} + \mathtt{U8} \cdot \mathtt{U5} \cdot \mathtt{u6} \cdot \mathtt{h} + \mathtt{U7} \cdot \mathtt{U5} \cdot \mathtt{u6} \cdot \mathtt{h} + \\
& \mathtt{U8} \cdot \mathtt{U4} \cdot \mathtt{u6} \cdot \mathtt{h} + \mathtt{U7} \cdot \mathtt{U4} \cdot \mathtt{u6} \cdot \mathtt{h} + \mathtt{U8} \cdot \mathtt{U3} \cdot \mathtt{u6} \cdot \mathtt{h} + \mathtt{U7} \cdot \mathtt{U3} \cdot \mathtt{u6} \cdot \mathtt{h} + \\
& \mathtt{U8} \cdot \mathtt{U2} \cdot \mathtt{u6} \cdot \mathtt{h} + \mathtt{U7} \cdot \mathtt{U2} \cdot \mathtt{u6} \cdot \mathtt{h} - \mathtt{U7} \cdot \mathtt{u8} \cdot \mathtt{u6} \cdot \mathtt{h} + \mathtt{U8} \cdot \mathtt{U4} \cdot \mathtt{u5} \cdot \mathtt{h} + \\
& \mathtt{U7} \cdot \mathtt{U4} \cdot \mathtt{u5} \cdot \mathtt{h} + \mathtt{U6} \cdot \mathtt{U4} \cdot \mathtt{u5} \cdot \mathtt{h} + \mathtt{U8} \cdot \mathtt{U3} \cdot \mathtt{u5} \cdot \mathtt{h} + \mathtt{U7} \cdot \mathtt{U3} \cdot \mathtt{u5} \cdot \mathtt{h} + \\
& \mathtt{U6} \cdot \mathtt{U3} \cdot \mathtt{u5} \cdot \mathtt{h} + \mathtt{U8} \cdot \mathtt{U2} \cdot \mathtt{u5} \cdot \mathtt{h} + \mathtt{U7} \cdot \mathtt{U2} \cdot \mathtt{u5} \cdot \mathtt{h} + \mathtt{U6} \cdot \mathtt{U2} \cdot \mathtt{u5} \cdot \mathtt{h} - \\
& \mathtt{U7} \cdot \mathtt{u8} \cdot \mathtt{u5} \cdot \mathtt{h} - \mathtt{U6} \cdot \mathtt{u8} \cdot \mathtt{u5} \cdot \mathtt{h} - \mathtt{U6} \cdot \mathtt{u7} \cdot \mathtt{u5} \cdot \mathtt{h} + \mathtt{U8} \cdot \mathtt{U3} \cdot \mathtt{u4} \cdot \mathtt{h} + \\
& \mathtt{U7} \cdot \mathtt{U3} \cdot \mathtt{u4} \cdot \mathtt{h} + \mathtt{U6} \cdot \mathtt{U3} \cdot \mathtt{u4} \cdot \mathtt{h} + \mathtt{U5} \cdot \mathtt{U3} \cdot \mathtt{u4} \cdot \mathtt{h} + \mathtt{U8} \cdot \mathtt{U2} \cdot \mathtt{u4} \cdot \mathtt{h} + \\
& \mathtt{U7} \cdot \mathtt{U2} \cdot \mathtt{u4} \cdot \mathtt{h} + \mathtt{U6} \cdot \mathtt{U2} \cdot \mathtt{u4} \cdot \mathtt{h} + \mathtt{U5} \cdot \mathtt{U2} \cdot \mathtt{u4} \cdot \mathtt{h} - \mathtt{U7} \cdot \mathtt{u8} \cdot \mathtt{u4} \cdot \mathtt{h} - \\
& \mathtt{U6} \cdot \mathtt{u8} \cdot \mathtt{u4} \cdot \mathtt{h} - \mathtt{U5} \cdot \mathtt{u8} \cdot \mathtt{u4} \cdot \mathtt{h} - \mathtt{U6} \cdot \mathtt{u7} \cdot \mathtt{u4} \cdot \mathtt{h} - \mathtt{U5} \cdot \mathtt{u7} \cdot \mathtt{u4} \cdot \mathtt{h} - \\
& \mathtt{U5} \cdot \mathtt{u6} \cdot \mathtt{u4} \cdot \mathtt{h} + \mathtt{U8} \cdot \mathtt{U2} \cdot \mathtt{u3} \cdot \mathtt{h} + \mathtt{U7} \cdot \mathtt{U2} \cdot \mathtt{u3} \cdot \mathtt{h} + \mathtt{U6} \cdot \mathtt{U2} \cdot \mathtt{u3} \cdot \mathtt{h} + \\
& \mathtt{U5} \cdot \mathtt{U2} \cdot \mathtt{u3} \cdot \mathtt{h} + \mathtt{U4} \cdot \mathtt{U2} \cdot \mathtt{u3} \cdot \mathtt{h} - \mathtt{U7} \cdot \mathtt{u8} \cdot \mathtt{u3} \cdot \mathtt{h} - \mathtt{U6} \cdot \mathtt{u8} \cdot \mathtt{u3} \cdot \mathtt{h} - \\
& \mathtt{U5} \cdot \mathtt{u8} \cdot \mathtt{u3} \cdot \mathtt{h} - \mathtt{U4} \cdot \mathtt{u8} \cdot \mathtt{u3} \cdot \mathtt{h} - \mathtt{U6} \cdot \mathtt{u7} \cdot \mathtt{u3} \cdot \mathtt{h} - \mathtt{U5} \cdot \mathtt{u7} \cdot \mathtt{u3} \cdot \mathtt{h} - \\
& \mathtt{U4} \cdot \mathtt{u7} \cdot \mathtt{u3} \cdot \mathtt{h} - \mathtt{U5} \cdot \mathtt{u6} \cdot \mathtt{u3} \cdot \mathtt{h} - \mathtt{U4} \cdot \mathtt{u6} \cdot \mathtt{u3} \cdot \mathtt{h} - \mathtt{U4} \cdot \mathtt{u5} \cdot \mathtt{u3} \cdot \mathtt{h} - \\
& \mathtt{U7} \cdot \mathtt{u8} \cdot \mathtt{u2} \cdot \mathtt{h} - \mathtt{U6} \cdot \mathtt{u8} \cdot \mathtt{u2} \cdot \mathtt{h} - \mathtt{U5} \cdot \mathtt{u8} \cdot \mathtt{u2} \cdot \mathtt{h} - \mathtt{U4} \cdot \mathtt{u8} \cdot \mathtt{u2} \cdot \mathtt{h} - \\
& \mathtt{U3} \cdot \mathtt{u8} \cdot \mathtt{u2} \cdot \mathtt{h} - \mathtt{U6} \cdot \mathtt{u7} \cdot \mathtt{u2} \cdot \mathtt{h} - \mathtt{U5} \cdot \mathtt{u7} \cdot \mathtt{u2} \cdot \mathtt{h} - \mathtt{U4} \cdot \mathtt{u7} \cdot \mathtt{u2} \cdot \mathtt{h} - \\
& \mathtt{U3} \cdot \mathtt{u7} \cdot \mathtt{u2} \cdot \mathtt{h} - \mathtt{U5} \cdot \mathtt{u6} \cdot \mathtt{u2} \cdot \mathtt{h} - \mathtt{U4} \cdot \mathtt{u6} \cdot \mathtt{u2} \cdot \mathtt{h} - \mathtt{U3} \cdot \mathtt{u6} \cdot \mathtt{u2} \cdot \mathtt{h} - \\
& \mathtt{U4} \cdot \mathtt{u5} \cdot \mathtt{u2} \cdot \mathtt{h} - \mathtt{U3} \cdot \mathtt{u5} \cdot \mathtt{u2} \cdot \mathtt{h} - \mathtt{U3} \cdot \mathtt{u4} \cdot \mathtt{u2} \cdot \mathtt{h}
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{i[15]} \quad = \quad & 2\cdot U7\cdot U5\cdot u8\cdot u6+2\cdot U7\cdot U4\cdot u8\cdot u6+2\cdot U7\cdot U3\cdot u8\cdot u6+\\
& 2\cdot U7\cdot U2\cdot u8\cdot u6+2\cdot U7\cdot U4\cdot u8\cdot u5+2\cdot U6\cdot U4\cdot u8\cdot u5+\\
& 2\cdot U7\cdot U3\cdot u8\cdot u5+2\cdot U6\cdot U3\cdot u8\cdot u5+2\cdot U7\cdot U2\cdot u8\cdot u5+\\
& 2\cdot U6\cdot U2\cdot u8\cdot u5-2\cdot U8\cdot U6\cdot u7\cdot u5+2\cdot U6\cdot U4\cdot u7\cdot u5+\\
& 2\cdot U6\cdot U3\cdot u7\cdot u5+2\cdot U6\cdot U2\cdot u7\cdot u5+2\cdot U7\cdot U3\cdot u8\cdot u4+\\
& 2\cdot U6\cdot U3\cdot u8\cdot u4+2\cdot U5\cdot U3\cdot u8\cdot u4+2\cdot U7\cdot U2\cdot u8\cdot u4+\\
& 2\cdot U6\cdot U2\cdot u8\cdot u4+2\cdot U5\cdot U2\cdot u8\cdot u4-2\cdot U8\cdot U6\cdot u7\cdot u4-\\
& 2\cdot U8\cdot U5\cdot u7\cdot u4+2\cdot U6\cdot U3\cdot u7\cdot u4+2\cdot U5\cdot U3\cdot u7\cdot u4+\\
& 2\cdot U6\cdot U2\cdot u7\cdot u4+2\cdot U5\cdot U2\cdot u7\cdot u4-2\cdot U8\cdot U5\cdot u6\cdot u4-\\
& 2\cdot U7\cdot U5\cdot u6\cdot u4+2\cdot U5\cdot U3\cdot u6\cdot u4+2\cdot U5\cdot U2\cdot u6\cdot u4+\\
& 2\cdot U7\cdot U2\cdot u8\cdot u3+2\cdot U6\cdot U2\cdot u8\cdot u3+2\cdot U5\cdot U2\cdot u8\cdot u3+\\
& 2\cdot U4\cdot U2\cdot u8\cdot u3-2\cdot U8\cdot U6\cdot u7\cdot u3-2\cdot U8\cdot U5\cdot u7\cdot u3-\\
& 2\cdot U8\cdot U4\cdot u7\cdot u3+2\cdot U6\cdot U2\cdot u7\cdot u3+2\cdot U5\cdot U2\cdot u7\cdot u3+\\
& 2\cdot U4\cdot U2\cdot u7\cdot u3-2\cdot U8\cdot U5\cdot u6\cdot u3-2\cdot U7\cdot U5\cdot u6\cdot u3-\\
& 2\cdot U8\cdot U4\cdot u6\cdot u3-2\cdot U7\cdot U4\cdot u6\cdot u3+2\cdot U5\cdot U2\cdot u6\cdot u3+\\
& 2\cdot U4\cdot U2\cdot u6\cdot u3-2\cdot U8\cdot U4\cdot u5\cdot u3-2\cdot U7\cdot U4\cdot u5\cdot u3-\\
& 2\cdot U6\cdot U4\cdot u5\cdot u3+2\cdot U4\cdot U2\cdot u5\cdot u3-2\cdot U8\cdot U6\cdot u7\cdot u2-\\
& 2\cdot U8\cdot U5\cdot u7\cdot u2-2\cdot U8\cdot U4\cdot u7\cdot u2-2\cdot U8\cdot U3\cdot u7\cdot u2-\\
& 2\cdot U8\cdot U5\cdot u6\cdot u2-2\cdot U7\cdot U5\cdot u6\cdot u2-2\cdot U8\cdot U4\cdot u6\cdot u2-\\
& 2\cdot U7\cdot U4\cdot u6\cdot u2-2\cdot U8\cdot U3\cdot u6\cdot u2-2\cdot U7\cdot U3\cdot u6\cdot u2-\\
& 2\cdot U8\cdot U4\cdot u5\cdot u2-2\cdot U7\cdot U4\cdot u5\cdot u2-2\cdot U6\cdot U4\cdot u5\cdot u2-\\
& 2\cdot U8\cdot U3\cdot u5\cdot u2-2\cdot U7\cdot U3\cdot u5\cdot u2-2\cdot U6\cdot U3\cdot u5\cdot u2-\\
& 2\cdot U8\cdot U3\cdot u4\cdot u2-2\cdot U7\cdot U3\cdot u4\cdot u2-2\cdot U6\cdot U3\cdot u4\cdot u2-\\
& 2\cdot U5\cdot U3\cdot u4\cdot u2-U8\cdot U6\cdot u7\cdot h-U8\cdot U5\cdot u7\cdot h-U8\cdot U4\cdot u7\cdot h-\\
& U8\cdot U3\cdot u7\cdot h-U8\cdot U2\cdot u7\cdot h-U8\cdot U5\cdot u6\cdot h-U7\cdot U5\cdot u6\cdot h-\\
& U8\cdot U4\cdot u6\cdot h-U7\cdot U4\cdot u6\cdot h-U8\cdot U3\cdot u6\cdot h-U7\cdot U3\cdot u6\cdot h-\\
& U8\cdot U2\cdot u6\cdot h-U7\cdot U2\cdot u6\cdot h+U7\cdot u8\cdot u6\cdot h-U8\cdot U4\cdot u5\cdot h-\\
& U7\cdot U4\cdot u5\cdot h-U6\cdot U4\cdot u5\cdot h-U8\cdot U3\cdot u5\cdot h-U7\cdot U3\cdot u5\cdot h-\\
& U6\cdot U3\cdot u5\cdot h-U8\cdot U2\cdot u5\cdot h-U7\cdot U2\cdot u5\cdot h-U6\cdot U2\cdot u5\cdot h+\\
& U7\cdot u8\cdot u5\cdot h+U6\cdot u8\cdot u5\cdot h+U6\cdot u7\cdot u5\cdot h-U8\cdot U3\cdot u4\cdot h-\\
& U7\cdot U3\cdot u4\cdot h-U6\cdot U3\cdot u4\cdot h-U5\cdot U3\cdot u4\cdot h-U8\cdot U2\cdot u4\cdot h-\\
& U7\cdot U2\cdot u4\cdot h-U6\cdot U2\cdot u4\cdot h-U5\cdot U2\cdot u4\cdot h+U7\cdot u8\cdot u4\cdot h+\\
& U6\cdot u8\cdot u4\cdot h+U5\cdot u8\cdot u4\cdot h+U6\cdot u7\cdot u4\cdot h+U5\cdot u7\cdot u4\cdot h+\\
& U5\cdot u6\cdot u4\cdot h-U8\cdot U2\cdot u3\cdot h-U7\cdot U2\cdot u3\cdot h-U6\cdot U2\cdot u3\cdot h-\\
& U5\cdot U2\cdot u3\cdot h-U4\cdot U2\cdot u3\cdot h+U7\cdot u8\cdot u3\cdot h+U6\cdot u8\cdot u3\cdot h+\\
& U5\cdot u8\cdot u3\cdot h+U4\cdot u8\cdot u3\cdot h+U6\cdot u7\cdot u3\cdot h+U5\cdot u7\cdot u3\cdot h+\\
& U4\cdot u7\cdot u3\cdot h+U5\cdot u6\cdot u3\cdot h+U4\cdot u6\cdot u3\cdot h+U4\cdot u5\cdot u3\cdot h+\\
& U7\cdot u8\cdot u2\cdot h+U6\cdot u8\cdot u2\cdot h+U5\cdot u8\cdot u2\cdot h+U4\cdot u8\cdot u2\cdot h+\\
& U3\cdot u8\cdot u2\cdot h+U6\cdot u7\cdot u2\cdot h+U5\cdot u7\cdot u2\cdot h+U4\cdot u7\cdot u2\cdot h+\\
& U3\cdot u7\cdot u2\cdot h+U5\cdot u6\cdot u2\cdot h+U4\cdot u6\cdot u2\cdot h+U3\cdot u6\cdot u2\cdot h+\\
& U4\cdot u5\cdot u2\cdot h+U3\cdot u5\cdot u2\cdot h+U3\cdot u4\cdot u2\cdot h
\end{aligned}
$$

Gonnet-83-h

Polynomial ring in 18 variables: $\mathtt{a(0), a(2), a(3), a(4), a(5), b(0), b(1), b(2), b(3), b(4), b(5), c(0), c(1), c(2), c(3), c(4), c(5), h}$

$$
\begin{aligned}
\mathtt{i[1]} &= \mathtt{a(5)\cdot b(5)}\\
\mathtt{i[2]} &= \mathtt{a(5)\cdot b(4)+a(4)\cdot b(5)}\\
\mathtt{i[3]} &= \mathtt{a(4)\cdot b(4)}\\
\mathtt{i[4]} &= \mathtt{a(5)\cdot b(3)+a(3)\cdot b(5)}\\
\mathtt{i[5]} &= \mathtt{a(5)\cdot b(3)+a(3)\cdot b(5)+2\cdot a(5)\cdot b(5)}\\
\mathtt{i[6]} &= \mathtt{a(3)\cdot b(3)+a(5)\cdot b(3)+a(3)\cdot b(5)+a(5)\cdot b(5)}\\
\mathtt{i[7]} &= \mathtt{2\cdot a(3)\cdot b(3)+a(5)\cdot b(3)+a(3)\cdot b(5)}\\
\mathtt{i[8]} &= \mathtt{a(4)\cdot b(2)+a(2)\cdot b(4)}\\
\mathtt{i[9]} &= \mathtt{a(2)\cdot b(2)}\\
\mathtt{i[10]} &= \mathtt{a(5)\cdot b(1)+a(4)\cdot b(3)+a(3)\cdot b(4)+b(5)\cdot h}\\
\mathtt{i[11]} &= \mathtt{a(4)\cdot b(1)+b(4)\cdot h}\\
\mathtt{i[12]} &= \mathtt{a(2)\cdot b(1)+b(2)\cdot h}\\
\mathtt{i[13]} &= \mathtt{a(0)\cdot b(1)+a(4)\cdot b(1)+a(3)\cdot b(2)+a(2)\cdot b(3)+b(0)\cdot h+}\\
&\quad \mathtt{2\cdot b(1)\cdot h+b(4)\cdot h+c(1)\cdot h}\\
\mathtt{i[14]} &= \mathtt{a(5)\cdot b(0)+a(5)\cdot b(1)+a(4)\cdot b(3)+a(3)\cdot b(4)+2\cdot a(5)\cdot b(4)+}\\
&\quad \mathtt{a(0)\cdot b(5)+2\cdot a(4)\cdot b(5)+b(5)\cdot h+c(5)\cdot h}\\
\mathtt{i[15]} &= \mathtt{a(4)\cdot b(0)+a(4)\cdot b(1)+a(5)\cdot b(2)+a(0)\cdot b(4)+2\cdot a(4)\cdot b(4)+}\\
&\quad \mathtt{a(2)\cdot b(5)+b(4)\cdot h+c(4)\cdot h}\\
\mathtt{i[16]} &= \mathtt{a(3)\cdot b(0)+2\cdot a(3)\cdot b(1)+a(5)\cdot b(1)+a(0)\cdot b(3)+}\\
&\quad \mathtt{a(4)\cdot b(3)+a(3)\cdot b(4)+2\cdot b(3)\cdot h+b(5)\cdot h+c(3)\cdot h}\\
\mathtt{i[17]} &= \mathtt{a(3)\cdot b(0)+a(5)\cdot b(0)+a(3)\cdot b(1)+a(5)\cdot b(1)+a(0)\cdot b(3)+}\\
&\quad \mathtt{a(4)\cdot b(3)+a(3)\cdot b(4)+a(5)\cdot b(4)+a(0)\cdot b(5)+a(4)\cdot b(5)+}\\
&\quad \mathtt{b(3)\cdot h+b(5)\cdot h+c(3)\cdot h+c(5)\cdot h-h\hat{}2}\\
\mathtt{i[18]} &= \mathtt{a(2)\cdot b(0)+a(2)\cdot b(1)+a(0)\cdot b(2)+a(4)\cdot b(2)+a(2)\cdot b(4)+}\\
&\quad \mathtt{b(2)\cdot h+c(2)\cdot h}\\
\mathtt{i[19]} &= \mathtt{a(0)\cdot b(0)+a(4)\cdot b(0)+a(0)\cdot b(1)+a(4)\cdot b(1)+a(3)\cdot b(2)+}\\
&\quad \mathtt{a(5)\cdot b(2)+a(2)\cdot b(3)+a(0)\cdot b(4)+a(4)\cdot b(4)+a(2)\cdot b(5)+}\\
&\quad \mathtt{b(0)\cdot h+b(1)\cdot h+b(4)\cdot h+c(0)\cdot h+c(1)\cdot h+c(4)\cdot h}
\end{aligned}
$$

Katsura-8

Polynomial ring in 9 variables: $\mathtt{x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8)}$

$$
\begin{aligned}
\mathtt{i[1]} &= \mathtt{x(0)} + \mathtt{2} \cdot \mathtt{x(1)} + \mathtt{2} \cdot \mathtt{x(2)} + \mathtt{2} \cdot \mathtt{x(3)} + \mathtt{2} \cdot \mathtt{x(4)} + \mathtt{2} \cdot \mathtt{x(5)} + \mathtt{2} \cdot \mathtt{x(6)} + \\
&\quad \mathtt{2} \cdot \mathtt{x(7)} - \mathtt{1} \\
\mathtt{i[2]} &= \mathtt{x(0)\hat{}2} + \mathtt{2} \cdot \mathtt{x(1)\hat{}2} + \mathtt{2} \cdot \mathtt{x(2)\hat{}2} + \mathtt{2} \cdot \mathtt{x(3)\hat{}2} + \mathtt{2} \cdot \mathtt{x(4)\hat{}2} + \\
&\quad \mathtt{2} \cdot \mathtt{x(5)\hat{}2} + \mathtt{2} \cdot \mathtt{x(6)\hat{}2} + \mathtt{2} \cdot \mathtt{x(7)\hat{}2} - \mathtt{x(0)} \\
\mathtt{i[3]} &= \mathtt{2} \cdot \mathtt{x(0)} \cdot \mathtt{x(1)} + \mathtt{2} \cdot \mathtt{x(1)} \cdot \mathtt{x(2)} + \mathtt{2} \cdot \mathtt{x(2)} \cdot \mathtt{x(3)} + \mathtt{2} \cdot \mathtt{x(3)} \cdot \mathtt{x(4)} + \\
&\quad \mathtt{2} \cdot \mathtt{x(4)} \cdot \mathtt{x(5)} + \mathtt{2} \cdot \mathtt{x(5)} \cdot \mathtt{x(6)} + \mathtt{2} \cdot \mathtt{x(6)} \cdot \mathtt{x(7)} - \mathtt{x(1)} \\
\mathtt{i[4]} &= \mathtt{x(1)\hat{}2} + \mathtt{2} \cdot \mathtt{x(0)} \cdot \mathtt{x(2)} + \mathtt{2} \cdot \mathtt{x(1)} \cdot \mathtt{x(3)} + \mathtt{2} \cdot \mathtt{x(2)} \cdot \mathtt{x(4)} + \\
&\quad \mathtt{2} \cdot \mathtt{x(3)} \cdot \mathtt{x(5)} + \mathtt{2} \cdot \mathtt{x(4)} \cdot \mathtt{x(6)} + \mathtt{2} \cdot \mathtt{x(5)} \cdot \mathtt{x(7)} - \mathtt{x(2)} \\
\mathtt{i[5]} &= \mathtt{2} \cdot \mathtt{x(1)} \cdot \mathtt{x(2)} + \mathtt{2} \cdot \mathtt{x(0)} \cdot \mathtt{x(3)} + \mathtt{2} \cdot \mathtt{x(1)} \cdot \mathtt{x(4)} + \mathtt{2} \cdot \mathtt{x(2)} \cdot \mathtt{x(5)} + \\
&\quad \mathtt{2} \cdot \mathtt{x(3)} \cdot \mathtt{x(6)} + \mathtt{2} \cdot \mathtt{x(4)} \cdot \mathtt{x(7)} - \mathtt{x(3)} \\
\mathtt{i[6]} &= \mathtt{x(2)\hat{}2} + \mathtt{2} \cdot \mathtt{x(1)} \cdot \mathtt{x(3)} + \mathtt{2} \cdot \mathtt{x(0)} \cdot \mathtt{x(4)} + \mathtt{2} \cdot \mathtt{x(1)} \cdot \mathtt{x(5)} + \\
&\quad \mathtt{2} \cdot \mathtt{x(2)} \cdot \mathtt{x(6)} + \mathtt{2} \cdot \mathtt{x(3)} \cdot \mathtt{x(7)} - \mathtt{x(4)} \\
\mathtt{i[7]} &= \mathtt{2} \cdot \mathtt{x(2)} \cdot \mathtt{x(3)} + \mathtt{2} \cdot \mathtt{x(1)} \cdot \mathtt{x(4)} + \mathtt{2} \cdot \mathtt{x(0)} \cdot \mathtt{x(5)} + \mathtt{2} \cdot \mathtt{x(1)} \cdot \mathtt{x(6)} + \\
&\quad \mathtt{2} \cdot \mathtt{x(2)} \cdot \mathtt{x(7)} - \mathtt{x(5)} \\
\mathtt{i[8]} &= \mathtt{x(3)\hat{}2} + \mathtt{2} \cdot \mathtt{x(2)} \cdot \mathtt{x(4)} + \mathtt{2} \cdot \mathtt{x(1)} \cdot \mathtt{x(5)} + \mathtt{2} \cdot \mathtt{x(0)} \cdot \mathtt{x(6)} + \\
&\quad \mathtt{2} \cdot \mathtt{x(1)} \cdot \mathtt{x(7)} - \mathtt{x(6)}
\end{aligned}
$$

Katsura-8-h

Polynomial ring in 10 variables: x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), h

$$
\begin{array}{lcl}
\mathtt{i[1]} & = & \mathtt{x(0)} + 2\cdot\mathtt{x(1)} + 2\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(6)} + \\
 & & 2\cdot\mathtt{x(7)} - \mathtt{h} \\
\mathtt{i[2]} & = & \mathtt{x(0)\hat{}2} + 2\cdot\mathtt{x(1)\hat{}2} + 2\cdot\mathtt{x(2)\hat{}2} + 2\cdot\mathtt{x(3)\hat{}2} + 2\cdot\mathtt{x(4)\hat{}2} + \\
 & & 2\cdot\mathtt{x(5)\hat{}2} + 2\cdot\mathtt{x(6)\hat{}2} + 2\cdot\mathtt{x(7)\hat{}2} - \mathtt{x(0)}\cdot\mathtt{h} \\
\mathtt{i[3]} & = & 2\cdot\mathtt{x(0)}\cdot\mathtt{x(1)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(4)} + \\
 & & 2\cdot\mathtt{x(4)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(6)}\cdot\mathtt{x(7)} - \mathtt{x(1)}\cdot\mathtt{h} \\
\mathtt{i[4]} & = & \mathtt{x(1)\hat{}2} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(4)} + \\
 & & 2\cdot\mathtt{x(3)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(7)} - \mathtt{x(2)}\cdot\mathtt{h} \\
\mathtt{i[5]} & = & 2\cdot\mathtt{x(1)}\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(5)} + \\
 & & 2\cdot\mathtt{x(3)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(7)} - \mathtt{x(3)}\cdot\mathtt{h} \\
\mathtt{i[6]} & = & \mathtt{x(2)\hat{}2} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(5)} + \\
 & & 2\cdot\mathtt{x(2)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(7)} - \mathtt{x(4)}\cdot\mathtt{h} \\
\mathtt{i[7]} & = & 2\cdot\mathtt{x(2)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(6)} + \\
 & & 2\cdot\mathtt{x(2)}\cdot\mathtt{x(7)} - \mathtt{x(5)}\cdot\mathtt{h} \\
\mathtt{i[8]} & = & \mathtt{x(3)\hat{}2} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(6)} + \\
 & & 2\cdot\mathtt{x(1)}\cdot\mathtt{x(7)} - \mathtt{x(6)}\cdot\mathtt{h}
\end{array}
$$

Katsura-9

Polynomial ring in 10 variables: $x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), x(9)$

$$
\begin{aligned}
i[1] &= x(0) + 2 \cdot x(1) + 2 \cdot x(2) + 2 \cdot x(3) + 2 \cdot x(4) + 2 \cdot x(5) + 2 \cdot x(6) + \\
&\quad 2 \cdot x(7) + 2 \cdot x(8) - 1 \\
i[2] &= x(0)\hat{}2 + 2 \cdot x(1)\hat{}2 + 2 \cdot x(2)\hat{}2 + 2 \cdot x(3)\hat{}2 + 2 \cdot x(4)\hat{}2 + \\
&\quad 2 \cdot x(5)\hat{}2 + 2 \cdot x(6)\hat{}2 + 2 \cdot x(7)\hat{}2 + 2 \cdot x(8)\hat{}2 - x(0) \\
i[3] &= 2 \cdot x(0) \cdot x(1) + 2 \cdot x(1) \cdot x(2) + 2 \cdot x(2) \cdot x(3) + 2 \cdot x(3) \cdot x(4) + \\
&\quad 2 \cdot x(4) \cdot x(5) + 2 \cdot x(5) \cdot x(6) + 2 \cdot x(6) \cdot x(7) + 2 \cdot x(7) \cdot x(8) - \\
&\quad x(1) \\
i[4] &= x(1)\hat{}2 + 2 \cdot x(0) \cdot x(2) + 2 \cdot x(1) \cdot x(3) + 2 \cdot x(2) \cdot x(4) + \\
&\quad 2 \cdot x(3) \cdot x(5) + 2 \cdot x(4) \cdot x(6) + 2 \cdot x(5) \cdot x(7) + 2 \cdot x(6) \cdot x(8) - \\
&\quad x(2) \\
i[5] &= 2 \cdot x(1) \cdot x(2) + 2 \cdot x(0) \cdot x(3) + 2 \cdot x(1) \cdot x(4) + 2 \cdot x(2) \cdot x(5) + \\
&\quad 2 \cdot x(3) \cdot x(6) + 2 \cdot x(4) \cdot x(7) + 2 \cdot x(5) \cdot x(8) - x(3) \\
i[6] &= x(2)\hat{}2 + 2 \cdot x(1) \cdot x(3) + 2 \cdot x(0) \cdot x(4) + 2 \cdot x(1) \cdot x(5) + \\
&\quad 2 \cdot x(2) \cdot x(6) + 2 \cdot x(3) \cdot x(7) + 2 \cdot x(4) \cdot x(8) - x(4) \\
i[7] &= 2 \cdot x(2) \cdot x(3) + 2 \cdot x(1) \cdot x(4) + 2 \cdot x(0) \cdot x(5) + 2 \cdot x(1) \cdot x(6) + \\
&\quad 2 \cdot x(2) \cdot x(7) + 2 \cdot x(3) \cdot x(8) - x(5) \\
i[8] &= x(3)\hat{}2 + 2 \cdot x(2) \cdot x(4) + 2 \cdot x(1) \cdot x(5) + 2 \cdot x(0) \cdot x(6) + \\
&\quad 2 \cdot x(1) \cdot x(7) + 2 \cdot x(2) \cdot x(8) - x(6) \\
i[9] &= 2 \cdot x(3) \cdot x(4) + 2 \cdot x(2) \cdot x(5) + 2 \cdot x(1) \cdot x(6) + 2 \cdot x(0) \cdot x(7) + \\
&\quad 2 \cdot x(1) \cdot x(8) - x(7)
\end{aligned}
$$

Katsura-9-h

Polynomial ring in 11 variables: $\mathtt{x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), x(9), h}$

$$
\begin{aligned}
\mathtt{i[1]} &= \mathtt{x(0) + 2\cdot x(1) + 2\cdot x(2) + 2\cdot x(3) + 2\cdot x(4) + 2\cdot x(5) + 2\cdot x(6) +} \\
&\quad \mathtt{2\cdot x(7) + 2\cdot x(8) - h} \\
\mathtt{i[2]} &= \mathtt{x(0)\hat{}2 + 2\cdot x(1)\hat{}2 + 2\cdot x(2)\hat{}2 + 2\cdot x(3)\hat{}2 + 2\cdot x(4)\hat{}2 +} \\
&\quad \mathtt{2\cdot x(5)\hat{}2 + 2\cdot x(6)\hat{}2 + 2\cdot x(7)\hat{}2 + 2\cdot x(8)\hat{}2 - x(0)\cdot h} \\
\mathtt{i[3]} &= \mathtt{2\cdot x(0)\cdot x(1) + 2\cdot x(1)\cdot x(2) + 2\cdot x(2)\cdot x(3) + 2\cdot x(3)\cdot x(4) +} \\
&\quad \mathtt{2\cdot x(4)\cdot x(5) + 2\cdot x(5)\cdot x(6) + 2\cdot x(6)\cdot x(7) + 2\cdot x(7)\cdot x(8) -} \\
&\quad \mathtt{x(1)\cdot h} \\
\mathtt{i[4]} &= \mathtt{x(1)\hat{}2 + 2\cdot x(0)\cdot x(2) + 2\cdot x(1)\cdot x(3) + 2\cdot x(2)\cdot x(4) +} \\
&\quad \mathtt{2\cdot x(3)\cdot x(5) + 2\cdot x(4)\cdot x(6) + 2\cdot x(5)\cdot x(7) + 2\cdot x(6)\cdot x(8) -} \\
&\quad \mathtt{x(2)\cdot h} \\
\mathtt{i[5]} &= \mathtt{2\cdot x(1)\cdot x(2) + 2\cdot x(0)\cdot x(3) + 2\cdot x(1)\cdot x(4) + 2\cdot x(2)\cdot x(5) +} \\
&\quad \mathtt{2\cdot x(3)\cdot x(6) + 2\cdot x(4)\cdot x(7) + 2\cdot x(5)\cdot x(8) - x(3)\cdot h} \\
\mathtt{i[6]} &= \mathtt{x(2)\hat{}2 + 2\cdot x(1)\cdot x(3) + 2\cdot x(0)\cdot x(4) + 2\cdot x(1)\cdot x(5) +} \\
&\quad \mathtt{2\cdot x(2)\cdot x(6) + 2\cdot x(3)\cdot x(7) + 2\cdot x(4)\cdot x(8) - x(4)\cdot h} \\
\mathtt{i[7]} &= \mathtt{2\cdot x(2)\cdot x(3) + 2\cdot x(1)\cdot x(4) + 2\cdot x(0)\cdot x(5) + 2\cdot x(1)\cdot x(6) +} \\
&\quad \mathtt{2\cdot x(2)\cdot x(7) + 2\cdot x(3)\cdot x(8) - x(5)\cdot h} \\
\mathtt{i[8]} &= \mathtt{x(3)\hat{}2 + 2\cdot x(2)\cdot x(4) + 2\cdot x(1)\cdot x(5) + 2\cdot x(0)\cdot x(6) +} \\
&\quad \mathtt{2\cdot x(1)\cdot x(7) + 2\cdot x(2)\cdot x(8) - x(6)\cdot h} \\
\mathtt{i[9]} &= \mathtt{2\cdot x(3)\cdot x(4) + 2\cdot x(2)\cdot x(5) + 2\cdot x(1)\cdot x(6) + 2\cdot x(0)\cdot x(7) +} \\
&\quad \mathtt{2\cdot x(1)\cdot x(8) - x(7)\cdot h}
\end{aligned}
$$

Katsura-10

Polynomial ring in 11 variables: $\mathtt{x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), x(9), x(10)}$

$$
\begin{aligned}
\mathtt{i[1]} &= \mathtt{x(0)} + 2\cdot\mathtt{x(1)} + 2\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(6)} + \\
&\quad 2\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(9)} - 1 \\
\mathtt{i[2]} &= \mathtt{x(0)\hat{}2} + 2\cdot\mathtt{x(1)\hat{}2} + 2\cdot\mathtt{x(2)\hat{}2} + 2\cdot\mathtt{x(3)\hat{}2} + 2\cdot\mathtt{x(4)\hat{}2} + \\
&\quad 2\cdot\mathtt{x(5)\hat{}2} + 2\cdot\mathtt{x(6)\hat{}2} + 2\cdot\mathtt{x(7)\hat{}2} + 2\cdot\mathtt{x(8)\hat{}2} + 2\cdot\mathtt{x(9)\hat{}2} - \mathtt{x(0)} \\
\mathtt{i[3]} &= 2\cdot\mathtt{x(0)}\cdot\mathtt{x(1)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(4)} + \\
&\quad 2\cdot\mathtt{x(4)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(6)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(7)}\cdot\mathtt{x(8)} + \\
&\quad 2\cdot\mathtt{x(8)}\cdot\mathtt{x(9)} - \mathtt{x(1)} \\
\mathtt{i[4]} &= \mathtt{x(1)\hat{}2} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(4)} + \\
&\quad 2\cdot\mathtt{x(3)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(6)}\cdot\mathtt{x(8)} + \\
&\quad 2\cdot\mathtt{x(7)}\cdot\mathtt{x(9)} - \mathtt{x(2)} \\
\mathtt{i[5]} &= 2\cdot\mathtt{x(1)}\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(5)} + \\
&\quad 2\cdot\mathtt{x(3)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(6)}\cdot\mathtt{x(9)} - \\
&\quad \mathtt{x(3)} \\
\mathtt{i[6]} &= \mathtt{x(2)\hat{}2} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(5)} + \\
&\quad 2\cdot\mathtt{x(2)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(9)} - \\
&\quad \mathtt{x(4)} \\
\mathtt{i[7]} &= 2\cdot\mathtt{x(2)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(6)} + \\
&\quad 2\cdot\mathtt{x(2)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(9)} - \mathtt{x(5)} \\
\mathtt{i[8]} &= \mathtt{x(3)\hat{}2} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(6)} + \\
&\quad 2\cdot\mathtt{x(1)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(9)} - \mathtt{x(6)} \\
\mathtt{i[9]} &= 2\cdot\mathtt{x(3)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(7)} + \\
&\quad 2\cdot\mathtt{x(1)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(9)} - \mathtt{x(7)} \\
\mathtt{i[10]} &= \mathtt{x(4)\hat{}2} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(7)} + \\
&\quad 2\cdot\mathtt{x(0)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(9)} - \mathtt{x(8)}
\end{aligned}
$$

Katsura-10-h

Polynomial ring in 12 variables: $\mathtt{x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), x(9), x(10), h}$

$$\mathtt{i[1]} = \mathtt{x(0)} + 2\cdot\mathtt{x(1)} + 2\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(9)} - \mathtt{h}$$

$$\mathtt{i[2]} = \mathtt{x(0)}\hat{}2 + 2\cdot\mathtt{x(1)}\hat{}2 + 2\cdot\mathtt{x(2)}\hat{}2 + 2\cdot\mathtt{x(3)}\hat{}2 + 2\cdot\mathtt{x(4)}\hat{}2 + 2\cdot\mathtt{x(5)}\hat{}2 + 2\cdot\mathtt{x(6)}\hat{}2 + 2\cdot\mathtt{x(7)}\hat{}2 + 2\cdot\mathtt{x(8)}\hat{}2 + 2\cdot\mathtt{x(9)}\hat{}2 - \mathtt{x(0)}\cdot\mathtt{h}$$

$$\mathtt{i[3]} = 2\cdot\mathtt{x(0)}\cdot\mathtt{x(1)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(6)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(7)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(8)}\cdot\mathtt{x(9)} - \mathtt{x(1)}\cdot\mathtt{h}$$

$$\mathtt{i[4]} = \mathtt{x(1)}\hat{}2 + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(6)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(7)}\cdot\mathtt{x(9)} - \mathtt{x(2)}\cdot\mathtt{h}$$

$$\mathtt{i[5]} = 2\cdot\mathtt{x(1)}\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(6)}\cdot\mathtt{x(9)} - \mathtt{x(3)}\cdot\mathtt{h}$$

$$\mathtt{i[6]} = \mathtt{x(2)}\hat{}2 + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(9)} - \mathtt{x(4)}\cdot\mathtt{h}$$

$$\mathtt{i[7]} = 2\cdot\mathtt{x(2)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(9)} - \mathtt{x(5)}\cdot\mathtt{h}$$

$$\mathtt{i[8]} = \mathtt{x(3)}\hat{}2 + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(9)} - \mathtt{x(6)}\cdot\mathtt{h}$$

$$\mathtt{i[9]} = 2\cdot\mathtt{x(3)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(9)} - \mathtt{x(7)}\cdot\mathtt{h}$$

$$\mathtt{i[10]} = \mathtt{x(4)}\hat{}2 + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(9)} - \mathtt{x(8)}\cdot\mathtt{h}$$

Katsura-11

Polynomial ring in 12 variables: $\mathtt{x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), x(9), x(10), x(11)}$

$$
\begin{aligned}
\mathtt{i[1]} &= \mathtt{x(0)} + 2\cdot\mathtt{x(1)} + 2\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(6)} + \\
&\quad 2\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(9)} + 2\cdot\mathtt{x(10)} - 1 \\
\mathtt{i[2]} &= \mathtt{x(0)\hat{\ }2} + 2\cdot\mathtt{x(1)\hat{\ }2} + 2\cdot\mathtt{x(2)\hat{\ }2} + 2\cdot\mathtt{x(3)\hat{\ }2} + 2\cdot\mathtt{x(4)\hat{\ }2} + \\
&\quad 2\cdot\mathtt{x(5)\hat{\ }2} + 2\cdot\mathtt{x(6)\hat{\ }2} + 2\cdot\mathtt{x(7)\hat{\ }2} + 2\cdot\mathtt{x(8)\hat{\ }2} + 2\cdot\mathtt{x(9)\hat{\ }2} + \\
&\quad 2\cdot\mathtt{x(10)\hat{\ }2} - \mathtt{x(0)} \\
\mathtt{i[3]} &= 2\cdot\mathtt{x(0)}\cdot\mathtt{x(1)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(4)} + \\
&\quad 2\cdot\mathtt{x(4)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(6)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(7)}\cdot\mathtt{x(8)} + \\
&\quad 2\cdot\mathtt{x(8)}\cdot\mathtt{x(9)} + 2\cdot\mathtt{x(9)}\cdot\mathtt{x(10)} - \mathtt{x(1)} \\
\mathtt{i[4]} &= \mathtt{x(1)\hat{\ }2} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(4)} + \\
&\quad 2\cdot\mathtt{x(3)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(6)}\cdot\mathtt{x(8)} + \\
&\quad 2\cdot\mathtt{x(7)}\cdot\mathtt{x(9)} + 2\cdot\mathtt{x(8)}\cdot\mathtt{x(10)} - \mathtt{x(2)} \\
\mathtt{i[5]} &= 2\cdot\mathtt{x(1)}\cdot\mathtt{x(2)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(5)} + \\
&\quad 2\cdot\mathtt{x(3)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(6)}\cdot\mathtt{x(9)} + \\
&\quad 2\cdot\mathtt{x(7)}\cdot\mathtt{x(10)} - \mathtt{x(3)} \\
\mathtt{i[6]} &= \mathtt{x(2)\hat{\ }2} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(5)} + \\
&\quad 2\cdot\mathtt{x(2)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(9)} + \\
&\quad 2\cdot\mathtt{x(6)}\cdot\mathtt{x(10)} - \mathtt{x(4)} \\
\mathtt{i[7]} &= 2\cdot\mathtt{x(2)}\cdot\mathtt{x(3)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(6)} + \\
&\quad 2\cdot\mathtt{x(2)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(9)} + 2\cdot\mathtt{x(5)}\cdot\mathtt{x(10)} - \\
&\quad \mathtt{x(5)} \\
\mathtt{i[8]} &= \mathtt{x(3)\hat{\ }2} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(6)} + \\
&\quad 2\cdot\mathtt{x(1)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(9)} + 2\cdot\mathtt{x(4)}\cdot\mathtt{x(10)} - \\
&\quad \mathtt{x(6)} \\
\mathtt{i[9]} &= 2\cdot\mathtt{x(3)}\cdot\mathtt{x(4)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(0)}\cdot\mathtt{x(7)} + \\
&\quad 2\cdot\mathtt{x(1)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(9)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(10)} - \mathtt{x(7)} \\
\mathtt{i[10]} &= \mathtt{x(4)\hat{\ }2} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(7)} + \\
&\quad 2\cdot\mathtt{x(0)}\cdot\mathtt{x(8)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(9)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(10)} - \mathtt{x(8)} \\
\mathtt{i[11]} &= 2\cdot\mathtt{x(4)}\cdot\mathtt{x(5)} + 2\cdot\mathtt{x(3)}\cdot\mathtt{x(6)} + 2\cdot\mathtt{x(2)}\cdot\mathtt{x(7)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(8)} + \\
&\quad 2\cdot\mathtt{x(0)}\cdot\mathtt{x(9)} + 2\cdot\mathtt{x(1)}\cdot\mathtt{x(10)} - \mathtt{x(9)}
\end{aligned}
$$

Katsura-11-h

Polynomial ring in 13 variables: x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), x(9), x(10), x(11), h

$$\begin{aligned}
\mathtt{i[1]} &= \mathtt{x(0) + 2\cdot x(1) + 2\cdot x(2) + 2\cdot x(3) + 2\cdot x(4) + 2\cdot x(5) + 2\cdot x(6) +}\\
&\quad \mathtt{2\cdot x(7) + 2\cdot x(8) + 2\cdot x(9) + 2\cdot x(10) - h}\\
\mathtt{i[2]} &= \mathtt{x(0)\hat{}2 + 2\cdot x(1)\hat{}2 + 2\cdot x(2)\hat{}2 + 2\cdot x(3)\hat{}2 + 2\cdot x(4)\hat{}2 +}\\
&\quad \mathtt{2\cdot x(5)\hat{}2 + 2\cdot x(6)\hat{}2 + 2\cdot x(7)\hat{}2 + 2\cdot x(8)\hat{}2 + 2\cdot x(9)\hat{}2 +}\\
&\quad \mathtt{2\cdot x(10)\hat{}2 - x(0)\cdot h}\\
\mathtt{i[3]} &= \mathtt{2\cdot x(0)\cdot x(1) + 2\cdot x(1)\cdot x(2) + 2\cdot x(2)\cdot x(3) + 2\cdot x(3)\cdot x(4) +}\\
&\quad \mathtt{2\cdot x(4)\cdot x(5) + 2\cdot x(5)\cdot x(6) + 2\cdot x(6)\cdot x(7) + 2\cdot x(7)\cdot x(8) +}\\
&\quad \mathtt{2\cdot x(8)\cdot x(9) + 2\cdot x(9)\cdot x(10) - x(1)\cdot h}\\
\mathtt{i[4]} &= \mathtt{x(1)\hat{}2 + 2\cdot x(0)\cdot x(2) + 2\cdot x(1)\cdot x(3) + 2\cdot x(2)\cdot x(4) +}\\
&\quad \mathtt{2\cdot x(3)\cdot x(5) + 2\cdot x(4)\cdot x(6) + 2\cdot x(5)\cdot x(7) + 2\cdot x(6)\cdot x(8) +}\\
&\quad \mathtt{2\cdot x(7)\cdot x(9) + 2\cdot x(8)\cdot x(10) - x(2)\cdot h}\\
\mathtt{i[5]} &= \mathtt{2\cdot x(1)\cdot x(2) + 2\cdot x(0)\cdot x(3) + 2\cdot x(1)\cdot x(4) + 2\cdot x(2)\cdot x(5) +}\\
&\quad \mathtt{2\cdot x(3)\cdot x(6) + 2\cdot x(4)\cdot x(7) + 2\cdot x(5)\cdot x(8) + 2\cdot x(6)\cdot x(9) +}\\
&\quad \mathtt{2\cdot x(7)\cdot x(10) - x(3)\cdot h}\\
\mathtt{i[6]} &= \mathtt{x(2)\hat{}2 + 2\cdot x(1)\cdot x(3) + 2\cdot x(0)\cdot x(4) + 2\cdot x(1)\cdot x(5) +}\\
&\quad \mathtt{2\cdot x(2)\cdot x(6) + 2\cdot x(3)\cdot x(7) + 2\cdot x(4)\cdot x(8) + 2\cdot x(5)\cdot x(9) +}\\
&\quad \mathtt{2\cdot x(6)\cdot x(10) - x(4)\cdot h}\\
\mathtt{i[7]} &= \mathtt{2\cdot x(2)\cdot x(3) + 2\cdot x(1)\cdot x(4) + 2\cdot x(0)\cdot x(5) + 2\cdot x(1)\cdot x(6) +}\\
&\quad \mathtt{2\cdot x(2)\cdot x(7) + 2\cdot x(3)\cdot x(8) + 2\cdot x(4)\cdot x(9) + 2\cdot x(5)\cdot x(10) -}\\
&\quad \mathtt{x(5)\cdot h}\\
\mathtt{i[8]} &= \mathtt{x(3)\hat{}2 + 2\cdot x(2)\cdot x(4) + 2\cdot x(1)\cdot x(5) + 2\cdot x(0)\cdot x(6) +}\\
&\quad \mathtt{2\cdot x(1)\cdot x(7) + 2\cdot x(2)\cdot x(8) + 2\cdot x(3)\cdot x(9) + 2\cdot x(4)\cdot x(10) -}\\
&\quad \mathtt{x(6)\cdot h}\\
\mathtt{i[9]} &= \mathtt{2\cdot x(3)\cdot x(4) + 2\cdot x(2)\cdot x(5) + 2\cdot x(1)\cdot x(6) + 2\cdot x(0)\cdot x(7) +}\\
&\quad \mathtt{2\cdot x(1)\cdot x(8) + 2\cdot x(2)\cdot x(9) + 2\cdot x(3)\cdot x(10) - x(7)\cdot h}\\
\mathtt{i[10]} &= \mathtt{x(4)\hat{}2 + 2\cdot x(3)\cdot x(5) + 2\cdot x(2)\cdot x(6) + 2\cdot x(1)\cdot x(7) +}\\
&\quad \mathtt{2\cdot x(0)\cdot x(8) + 2\cdot x(1)\cdot x(9) + 2\cdot x(2)\cdot x(10) - x(8)\cdot h}\\
\mathtt{i[11]} &= \mathtt{2\cdot x(4)\cdot x(5) + 2\cdot x(3)\cdot x(6) + 2\cdot x(2)\cdot x(7) + 2\cdot x(1)\cdot x(8) +}\\
&\quad \mathtt{2\cdot x(0)\cdot x(9) + 2\cdot x(1)\cdot x(10) - x(9)\cdot h}
\end{aligned}$$

Schrans-Troost-h

Polynomial ring in 9 variables: $\mathtt{x(1), x(2), x(3), x(4), x(5), x(6), x(7), x(8), h}$

$$
\begin{aligned}
\mathtt{i[1]} &= \mathtt{8 \cdot x(1)\hat{}2 + 8 \cdot x(1) \cdot x(2) + 8 \cdot x(1) \cdot x(3) - 8 \cdot x(2) \cdot x(3) +} \\
&\quad \mathtt{2 \cdot x(1) \cdot x(4) + 2 \cdot x(1) \cdot x(5) + 2 \cdot x(1) \cdot x(6) - 2 \cdot x(5) \cdot x(6) +} \\
&\quad \mathtt{2 \cdot x(1) \cdot x(7) - 2 \cdot x(4) \cdot x(7) - x(1) \cdot h} \\
\mathtt{i[2]} &= \mathtt{8 \cdot x(1) \cdot x(2) + 8 \cdot x(2)\hat{}2 - 8 \cdot x(1) \cdot x(3) + 8 \cdot x(2) \cdot x(3) +} \\
&\quad \mathtt{2 \cdot x(2) \cdot x(4) + 2 \cdot x(2) \cdot x(5) + 2 \cdot x(2) \cdot x(6) - 2 \cdot x(4) \cdot x(6) +} \\
&\quad \mathtt{2 \cdot x(2) \cdot x(7) - 2 \cdot x(5) \cdot x(7) - x(2) \cdot h} \\
\mathtt{i[3]} &= \mathtt{-8 \cdot x(1) \cdot x(2) + 8 \cdot x(1) \cdot x(3) + 8 \cdot x(2) \cdot x(3) + 8 \cdot x(3)\hat{}2 +} \\
&\quad \mathtt{2 \cdot x(3) \cdot x(4) + 2 \cdot x(3) \cdot x(5) - 2 \cdot x(4) \cdot x(5) + 2 \cdot x(3) \cdot x(6) +} \\
&\quad \mathtt{2 \cdot x(3) \cdot x(7) - 2 \cdot x(6) \cdot x(7) - x(3) \cdot h} \\
\mathtt{i[4]} &= \mathtt{2 \cdot x(1) \cdot x(4) + 2 \cdot x(2) \cdot x(4) + 2 \cdot x(3) \cdot x(4) + 8 \cdot x(4)\hat{}2 -} \\
&\quad \mathtt{2 \cdot x(3) \cdot x(5) + 8 \cdot x(4) \cdot x(5) - 2 \cdot x(2) \cdot x(6) + 2 \cdot x(4) \cdot x(6) -} \\
&\quad \mathtt{2 \cdot x(1) \cdot x(7) + 2 \cdot x(4) \cdot x(7) + 6 \cdot x(4) \cdot x(8) - 6 \cdot x(5) \cdot x(8) -} \\
&\quad \mathtt{x(4) \cdot h} \\
\mathtt{i[5]} &= \mathtt{-2 \cdot x(1) \cdot x(4) - 2 \cdot x(2) \cdot x(5) - 2 \cdot x(3) \cdot x(6) + 2 \cdot x(1) \cdot x(7) +} \\
&\quad \mathtt{2 \cdot x(2) \cdot x(7) + 2 \cdot x(3) \cdot x(7) + 2 \cdot x(4) \cdot x(7) + 2 \cdot x(5) \cdot x(7) +} \\
&\quad \mathtt{8 \cdot x(6) \cdot x(7) + 8 \cdot x(7)\hat{}2 - 6 \cdot x(6) \cdot x(8) + 6 \cdot x(7) \cdot x(8) -} \\
&\quad \mathtt{x(7) \cdot h} \\
\mathtt{i[6]} &= \mathtt{-2 \cdot x(2) \cdot x(4) - 2 \cdot x(1) \cdot x(5) + 2 \cdot x(1) \cdot x(6) + 2 \cdot x(2) \cdot x(6) +} \\
&\quad \mathtt{2 \cdot x(3) \cdot x(6) + 2 \cdot x(4) \cdot x(6) + 2 \cdot x(5) \cdot x(6) + 8 \cdot x(6)\hat{}2 -} \\
&\quad \mathtt{2 \cdot x(3) \cdot x(7) + 8 \cdot x(6) \cdot x(7) + 6 \cdot x(6) \cdot x(8) - 6 \cdot x(7) \cdot x(8) -} \\
&\quad \mathtt{x(6) \cdot h} \\
\mathtt{i[7]} &= \mathtt{-2 \cdot x(3) \cdot x(4) + 2 \cdot x(1) \cdot x(5) + 2 \cdot x(2) \cdot x(5) + 2 \cdot x(3) \cdot x(5) +} \\
&\quad \mathtt{8 \cdot x(4) \cdot x(5) + 8 \cdot x(5)\hat{}2 - 2 \cdot x(1) \cdot x(6) + 2 \cdot x(5) \cdot x(6) -} \\
&\quad \mathtt{2 \cdot x(2) \cdot x(7) + 2 \cdot x(5) \cdot x(7) - 6 \cdot x(4) \cdot x(8) + 6 \cdot x(5) \cdot x(8) -} \\
&\quad \mathtt{x(5) \cdot h} \\
\mathtt{i[8]} &= \mathtt{-6 \cdot x(4) \cdot x(5) - 6 \cdot x(6) \cdot x(7) + 6 \cdot x(4) \cdot x(8) + 6 \cdot x(5) \cdot x(8) +} \\
&\quad \mathtt{6 \cdot x(6) \cdot x(8) + 6 \cdot x(7) \cdot x(8) + 8 \cdot x(8)\hat{}2 - x(8) \cdot h}
\end{aligned}
$$

Bibliography

[1] Adams, W. W., and Loustaunau, P. *An Introduction to Gröbner Bases*. Graduate Studies in Mathematics, AMS, 1994.

[2] Albrecht, M. *Algorithmic Algebraic Techniques and their Application to Block Cipher Cryptanalysis*. PhD thesis, Royal Holloway, University of London, 2010.

[3] Albrecht, M., and Perry, J. F4/5. `http://arxiv.org/abs/1006.4933`.

[4] Amrhein, B., Gloor, O., and Kuchlin, W. On the Walk. *Theoretical Computer Science 187* (1997), 179–202.

[5] Arnold, E. A. *Computing Gröbner bases with Hilbert Lucky Primes*. PhD thesis, University of Maryland, College Park, MD, 2000.

[6] Arnold, E. A. Modular algorithms for computing Gröbner bases. *Journal of Symbolic Computation 35* (April 2003), 403–419.

[7] Arri, A., and Perry, J. The F5 Criterion revised. `http://arxiv.org/abs/1012.3664v3`.

[8] Ars, G. *Applications des bases de Gröbner à la cryptographie.* PhD thesis, Université de Rennes I, 2005.

[9] Ars, G., and Hashemi, A. Extended F5 Criteria. *Journal of Symbolic Computation, MEGA 2009 special issue 45*, 12 (2010), 1330–1340.

[10] Ars, G., and Hashemi, A. Computing Syzygies by Faugère's F5 Algorithm. *Results in Mathematics, Springer 59* (2011), 35–42.

[11] Astrelin, A. V., Golubitsky, O. D., and Pankratiev, E. V. Gröbner Bases and Involutive Bases. Walter Gruyter, pp. 49–55.

[12] Atiyah, M. F., and MacDonald, I. G. *Introduction to Commutative Algebra.* Addison-Wesley, London, 1969.

[13] Auerbach, R. L. The Gröbner fan and Gröbner walk for modules. *Journal of Symbolic Computation 39* (2005), 127–153.

[14] Bardet, M. *Étude des systèmes algébriques surdéterminés. Applications aux codes correcteurs et à la cryptographie.* PhD thesis, Université Paris 6, 2004.

[15] Bardet, M. On the Complexity of a Gröbner Basis Algorithm. INRIA Algorithms seminar 2002–2004.

[16] Bardet, M., Faugère, J.-C., and Salvy, B. Asympotic Expansion of the Degree of Regularity for Semi-Regular Systems of Equations. Manuscript downloaded from `www-calfor.lip6.fr/~jcf/Papers/BFS05.pdf`.

[17] Bayer, D., and Stillman, M. On the complexity of computing syzygies. *Journal of Symbolic Computation 6*, 2-3 (1988), 135 – 147.

[18] Becker, T., Weispfenning, V., and Kredel, H. *Gröbner Bases.* Graduate Texts in Mathematics, Springer Verlag, 1993.

[19] Bettale, L., Faugère, J.-C., and Perret, L. Cryptanalysis of Multivariate and Odd-Characteristic HFE Variants. In *Public Key Cryptography - PKC 2011* (2011), D. Catalano et al., Ed., vol. 6571 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 441–458.

[20] Bigatti, A. M. Computation of Hilbert-Poincaré series. *Journal of Pure and Applied Algebra 119* (1997), 237–253.

[21] Bigatti, A. M., Caboara, M., and Robbiano, L. On the computation of the Hilbert-Poincaré series. *AAECC Journal 2* (1991), 21–33.

[22] Bigatti, A. M., Conti, P., Robbiano, L., and Traverso, C. On the computation of the Hilbert-Poincaré series.

[23] Bini, D., and Pan, V. Improved parallel division and its extensions. In *Foundations of Computer Science* (1992), vol. 33, pp. 131–136.

[24] Blinkov, Y. A., and Gerdt, V. P. Involutive bases of polynomial ideals. *Mathematics and Computers in Simulation 45* (1998), 519–541.

[25] Blinkov, Y. A., and Gerdt, V. P. Minimal involutive bases. *Math. Comput. Simul. 45* (March 1998), 543–560.

[26] Blinkov, Y. A., and Zharkov, A. Y. Involution approach to investigating polynomial systems. *Math. Comput. Simul. 42* (January 1996), 323–332.

[27] Bogart, T., Jensen, A. N., Speyer, D., Sturmfels, B., and Thomas, R. R. Computing tropical varieties. *Journal of Symbolic Computation 42*, 1-2 (2007), 54 – 73. Effective Methods in Algebraic Geometry (MEGA 2005).

[28] Borosh, I. Exact solutions of linear equations with rational coefficients by congruence techniques. *Mathematics of Computation 20*, 107–112.

[29] Bosma, W., Cannon, J., and Playoust, C. The Magma algebra system. I. The user language. *Journal of Symbolic Computation 24*, 3-4 (1997), 235–265. http://magma.maths.usyd.edu.au/magma/.

[30] Brickenstein, M. *Neue Varianten zur Berechnung von Gröbner Basen*. Diploma thesis, University of Kaiserslautern, 2004.

[31] Brickenstein, M. Slimgb: Gröbner bases with slim polynomials. *Revista Matemática Complutense 23*, 2 (2010), 453–466. the final publication is available at www.springerlink.com.

[32] Brickenstein, M., and Dreyer, A. PolyBoRi: A framework for Gröbner basis computations with Boolean polynomials. *Journal of Symbolic Computation 44*, 9 (September 2009), 1326–1345.

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

[34] Buchberger, B. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, University of Innsbruck, 1965.

[35] Buchberger, B. A criterion for detecting unnecessary reductions in the construction of Gröbner bases. In *EUROSAM '79, An International Symposium on Symbolic and Algebraic Manipulation* (1979), vol. 72 of *Lecture Notes in Computer Science*, Springer, pp. 3–21.

[36] Cabarcas, D. An Implementation of Faugère's F4 Algorithm for Computing Gröbner Bases. Master's thesis, University of Cincinnati, Engineering, 2010.

[37] Cabarcas, D., and Ding, J. Linear Algebra to Compute Syzygies and Gröbner Bases. In *ISSAC 2011: Proceedings of the 2011 international symposium on Symbolic and algebraic computation* (2011), pp. 67–74.

[38] Caboara, M. A Dynamic Algorithm for Gröbner basis computation. In *ISSAC'93* (1993).

[39] Caboara, M., De Dominicis. G, and Robbiano, L. Multigraded Hilbert Functions and Buchberger Algorithm. In *ISSAC'96, Zürich, Switzerland* (1996).

[40] Calmet, J., Hausdorf, M., and Seiler, W. M. A Constructive Introduction to Involution. pp. 33–50.

[41] Caniglia, L., Galligo, A., and Heintz, J. Some New Effectivity Bounds in Computational Geometry. In *Proceedings of the 6th International Conference, on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes* (London, UK, 1989), Springer-Verlag, pp. 131–151.

[42] Caniglia, L. and Galligo, A. and Heintz, J. Equations for the projective closure and effective Nullstellensatz. *Discrete Appl. Math. 33* (October 1991), 11–23.

[43] Cayley, A. On the theory of elimination. *Mathematical Journal 3* (1848), 116–120.

[44] Collart, S., Kalkbrener, M., and Mall, D. Converting Bases with the Groebner Walk. *Journal of Symbolic Computation 24* (1997), 265–469.

[45] Collins, G. E., and Encarnación, M. J. Efficient Rational Number Reconstruction. *Journal of Symbolic Computation 20* (1994), 287–297.

[46] Cox, D. A., Little, J., and O'Shea, D. *Using Algebraic Geometry*, 2nd ed. Graduate Texts in Mathematics, Springer Verlag, 2008.

[47] Cox, D. A., Little, J., and O'Shea, D. B. *Ideals, Varieties, and Algorithms*, 3rd ed. Undergraduate Texts in Mathematics, Springer, 2007.

[48] de Kleine, J., and Monagan, M. A Modular Method for computing Gröbner bases.

[49] Decker, W., Greuel, G.-M., Pfister, G., and Schönemann, H. Singular *3-1-3 — A computer algebra system for polynomial computations*, 2011. http://www.singular.uni-kl.de.

[50] Decker, W., and Lossen, C. *Computing in Algebraic Geometry - A Quick Start in* Singular. ACM 16, Springer Verlag, 2006.

[51] Decker, W., and Schreyer, F.-O. *Varieties, Gröbner Bases, and Algebraic Curves*. Springer Verlag, tbc.

[52] Dellaca, R. D. *Gröbner Basis Algorithms*. PhD thesis, California State University, Fullerton, 2009.

[53] Dubé, T. W. The Structure of Polynomial Ideals and Gröbner Bases. *SIAM Journal of Computation 19*, 4 (1990), 750–773.

[54] Ebert, G. L. Some comments on the modular approach to Gröbner-bases. *ACM SIGSAM Bulletin 17* (1983), 28–32.

[55] Eder, C. A new attempt on the F5 Criterion. *The Computer Science Journal of Moldova 16* (2008), 4–14.

[56] Eder, C. On the criteria of the F5 Algorithm. *preprint math.AC/0804.2033* (2008).

[57] Eder, C., Gash, J., and Perry, J. Modifying Faugère's F5 Algorithm to ensure termination. *ACM SIGSAM Communications in Computer Algebra 45*, 2 (2011), 70–89. `http://arxiv.org/abs/1006.0318`.

[58] Eder, C., and Perry, J. F5C: A Variant of Faugère's F5 Algorithm with reduced Gröbner bases. *Journal of Symbolic Computation, MEGA 2009 special issue 45*, 12 (2010), 1442–1458. `dx.doi.org/10.1016/j.jsc.2010.06.019`.

[59] Eder, C., and Perry, J. Signature-based Algorithms to Compute Gröbner Bases. In *ISSAC 2011: Proceedings of the 2011 international symposium on Symbolic and algebraic computation* (2011), pp. 99–106.

[60] Eisenbud, D. *Commutative Algebra: with a View Toward Algebraic Geometry*, 3rd ed. Graduate Texts in Mathematics, Springer Verlag, 2008.

[61] Faugère, J.-C. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra 139*, 1–3 (June 1999), 61–88.

[62] Faugère, J.-C. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). 75–83.

[63] Faugère, J.-C. A new efficient algorithm for computing Gröbner bases without reduction to zero F5. In *ISSAC'02, Villeneuve d'Ascq, France* (July 2002), pp. 75–82. Revised version from `http://fgbrs.lip6.fr/jcf/Publications/index.html`.

[64] Faugère, J.-C. Algebraic cryptanalysis of HFE using Gröbner bases. INRIA Research Report, n 4738.

[65] Faugère, J.-C. Interactions between computer algebra (Gröbner bases) and cryptology. In *ISSAC '09: Proceedings of the 2009 international symposium on Symbolic and algebraic computation* (New York, NY, USA, 2009), ISSAC '09, ACM, pp. 383–384.

[66] Faugère, J.-C. FGb: A Library for Computing Gröbner Bases. In *Mathematical Software - ICMS 2010* (Berlin, Heidelberg, September 2010), Fukuda, Komei and Hoeven, Joris and Joswig, Michael and Takayama, Nobuki, Ed., vol. 6327 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 84–87.

[67] Faugère, J.-C., Gianni, P. M., Lazard, D., and Mora, T. Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering. *Journal of Symbolic Computation 16*, 4 (1993), 329–344.

[68] Faugère, J.-C., and Joux, A. Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems Using Gröbner Bases. 44–60.

[69] Faugère, J.-C. and Lachartre, S. Parallel Gaussian Elimination for Gröbner bases computations in finite fields. In *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation* (New York, NY, USA, July 2010), M. Moreno-Maza and J.L. Roch, Ed., PASCO '10, ACM, pp. 89–97.

[70] Faugère, J.-C. and Mou, C. Fast Algorithm for Change of Ordering of Zero-dimensional Gröbner Bases with Sparse Multiplication Matrices. In *ISSAC 2011: Proceedings of the 2011 international symposium on Symbolic and algebraic computation* (New York, NY, USA, 2011), ISSAC '11, ACM, pp. 115–122.

[71] Faugère, J.-C. and "Safey El Din", M. and Spaenlehauer, P.-J. Gröbner Bases of Bihomogeneous Ideals Generated by Polynomials of Bidegree (1,1): Algorithms and Complexity. *Journal of Symbolic Computation 46*, 4 (2011), 406–437. Available online 4 November 2010.

[72] Fröberg, R. *An Introduction to Gröbner Bases*. John Wiley & Sons, 1997.

[73] Fukuda, K., Jensen, A. N., Lauritzen, N., and Thomas, R. The generic Gröbner walk. *Journal of Symbolic Computation 42*, 3 (2007), 298–312.

[74] Fukuda, K., Jensen, A. N., and Thomas, R. R. Computing Gröbner fans. *Mathematics of Computation 76* (2007), 2189–2212. PRO 060522.

[75] Fulton, W. *Introduction to Toric Varieties*. Princeton University Press, 1993.

[76] Gao, S., Guan, Y., and Volny IV, F. A New Incremental Algorithm for Computing Groebner Bases. *Journal of Symbolic Computation – ISSAC 2010 Special Issue 1* (2010), 13–19.

[77] Gao, S., Volny IV, F., and Wang, D. A new algorithm for computing Groebner bases. 2010.

[78] Gash, J. M. *On efficient computation of Gröbner bases*. PhD thesis, University of Indiana, Bloomington, IN, 2008.

[79] Gathen, J. von zur, and Gerhard, J. *Modern Computer Algebra*, 2nd ed. Cambridge University Press, Cambridge, England, 2003.

[80] Gebauer, R., and Möller, H. M. Buchberger's algorithm and staggered linear bases. In *Proceedings of the fifth ACM symposium on Symbolic and algebraic computation* (New York, NY, USA, 1986), SYMSAC '86, ACM, pp. 218–221.

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

[82] Gerdt, V. P. On an Algorithmic Optimization in Computation of Involutive Bases. *Program. Comput. Softw. 28* (March 2002), 62–65.

[83] Gerdt, V. P., and Yanovich, D. A. Implementation of the FGLM Algorithm and Finding Roots of Polynomial Involutive Systems. *Programming and Computer Software 29* (2003), 72–74.

[84] Gerdt, V. P., and Yanovich, D. A. Parallel computation of Janet and Gröbner bases over rational numbers. *Program. Comput. Softw. 31* (March 2005), 73–80.

[85] Giovini, A., Mora, T., Niesi, G., Robbiano, L., and Traverso, C. "One sugar cube, please" or selection strategies in the Buchberger algorithm. In *ISSAC'91* (1991), pp. 49–54.

[86] Giusti, M. A Note on the Complexity of Constructing Standard Bases. In *European Conference on Computer Algebra (2)* (1985), pp. 411–412.

[87] Giusti, M. Complexity of standard bases in projective dimension zero. In *EUROCAL* (1987), pp. 333–335.

[88] Giusti, M. Complexity of Standard Bases in Projective Dimension Zero II. In *AAECC* (1990), pp. 322–328.

[89] Golubitsky, O. D. Converging term order sequences and the dynamic Buchberger algorithm.

[90] Gordan, P. Neuer Beweis des Hilbertschen Satzes über homogene Funktionen. *Nachrichten König. Ges. der Wiss. zu G¨tt.* (1899), 240–242.

[91] Gräbe, H.-G. On lucky primes. *Journal of Symbolic Computation 15* (1994), 199–209.

[92] Gräbe, H.-G. The Tangent Cone Algorithm and Homogenization. *Journal of Pure and Applied Algebra 97* (1994), 303–312.

[93] Gräbe, H.-G. *The SymbolicData Project – Tools and Data for Testing Computer Algebra Software*, 2011. `http://www.symbolicdata.org`.

[94] Grassmann, H., Greuel, G.-M., Martin, W., Neumann, W., Pfister, G., Pohl, W., Schönemann, H., and Siebert, T. Standard Bases, Syzygies and their implementation in Singular. *Beiträge zur angewandten Analysis und Informatik, Aachen* (1994), 69–96.

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

[96] Greuel, G.-M., and Pfister, G. Advances and Improvements in the Theory of Standard Bases and Syzygies. *Archiv der Mathematik 66* (1996), 163–176.

[97] Greuel, G.-M., and Pfister, G. *A* Singular *Introduction to Commutative Algebra*, 2nd ed. Springer Verlag, 2007.

[98] Gröbner, W. Über die algebraischen Eigenschaften der Integrale von linearen Differentialgleichungen mit konstanten Koeffizienten. *Mohatsh. der Mathematik*, 47 (1939), 247–284.

[99] Harris, J. *Algebraic Geometry – A first course*, 3rd ed. Graduate Texts in Mathematics, Springer Verlag, 2010.

[100] Hartshorne, R. *Algebraic Geometry*, 8th ed. Graduate Texts in Mathematics, Springer Verlag, 1997.

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

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

[103] Idrees, N., Pfister, G., and Steidel, S. Parallelization of Modular Algorithms. *Journal of Symbolic Computation 46* (2011), 672–684.

[104] Janet, M. Sur les systèmes d'equations aux Dérivées Partielles. *Gauthier-Villars, Paris* (1920).

[105] Jensen, A. N. Gfan, a software system for Gröbner fans and tropical varieties. Available at http://www.math.tu-berlin.de/ jensen/software/gfan/gfan.html.

[106] Joux, A., and Vitse, V. A Variant of the F4 Algorithm. In *Topics in Cryptology – CT-RSA 2011*, Kiayias, Aggelos, Ed., vol. 6558 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2011, pp. 356–375.

[107] Kalkbrener, M. On the complexity of Gröbner Bases Conversion. *Journal of Symbolic Computation 28* (1999), 265–273.

[108] Kobayashi, H., Moritsugu, S., and Hogan, R. W. Solving Systems of Algebraic Equations. In *Proceedings of the International Symposium ISSAC'88 on Symbolic and Algebraic Computation* (London, UK, 1989), ISAAC '88, Springer-Verlag, pp. 139–149.

[109] Kollreider, C., and Buchberger, B. An improved algorithmic construction of Gröbner-bases for polynomial ideals. *SIGSAM Bull. 12* (May 1978), 27–36.

[110] Kornerup, P., and Gregory, R. T. Mapping integers and hensel codes onto Farey fractions. *BIT Numerical Mathematics 23* (1983), 9–20. 10.1007/BF01937322.

[111] Kreuzer, M., and Robbiano, L. *Computational Commutative Algebra 2*, 1st ed. Springer Verlag, 2005.

[112] Kreuzer, M., and Robbiano, L. *Computational Commutative Algebra 1*, 2nd ed. Springer Verlag, 2009.

[113] Kühnle, K., and Mayr, E. W. Exponential space computation of Gröbner bases. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC 96, Zürich, July 24-26* (1996), pp. 63–71.

[114] Lazard, D. Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations. In *EUROCAL'83, European Computer Algebra Conference* (1983), J. A. van Hulzen, Ed., vol. 162 of *Springer LNCS*, pp. 146–156.

[115] Lazard, D. Algèbre linéaire sur $\mathcal{K}[x_1, \ldots, x_n]$ et élimination. *Bull. Soc. Math. France 105* (1977), 165–190.

[116] Lazard, D. Systems of algebraic equations. In *EUROSAM* (1979), pp. 88–94.

[117] Lazard, D. Resolution des Systemes d'Equations Algebriques. *Theor. Comput. Sci. 15* (1981), 77–110.

[118] Lazard, D. Solving zero-dimensional algebraic systems. *Journal of Symbolic Computation 13*, 2 (1992), 117–131.

[119] Macaulay, F. S. On some Formulæ in Elimination. *Proceedings of the London Mathematical Society 33*, 1 (1902), 3–27.

[120] Macaulay, F. S. *The algebraic theory of modular systems*. Cambridge University Press, 1916.

[121] Macaulay, F. S. Some Properties of Enumeration in the Theory of Modular Systems. *Proceedings of the London Mathematical Society*, 26 (1939), 531–555.

[122] Mandache, A. M. The Gröbner basis algorithm and subresultant theory. In *ISSAC'94 Proceedings* (1994), pp. 123–128.

[123] Marc Giusti. Some Effectivity Problems in Polynomial Ideal Theory. In *EUROSAM* (1984), no. Computation, pp. 159–171.

[124] Mayr, E. W. Some complexity results for polynomial ideals. *Journal of Complexity 13*, 3 (1997), 303–325.

[125] Möller, H. M., and Mora, T. Upper and lower bounds for the degree of Gröbner bases. In *EUROSAM 84, Cambridge, July 9–11, 1984* (1984), pp. 172–183.

[126] Möller, H. M., Mora, T., and Traverso, C. Gröbner bases computation using syzygies. In *ISSAC 92: Papers from the International Symposium on Symbolic and Algebraic Computation* (1992), pp. 320–328.

[127] Monagan, M., Geddes, K. O., Heal, K. M., Pearce, R., Labahn, G., Vorkoetter, S. M., McCarron, J., and DeMarco, P. *Maple 15 Programming Guide*. Maplesoft, Waterloo ON, Canada, 2011.

[128] Monagan, M., and Pearce, R. Parallel Sparse Polynomial Multiplication Using Heaps. In *ISSAC 2009* (2009), pp. 295–315.

[129] Monagan, M., and Pearce, R. Parallel Sparse Polynomial Division Using Heaps. In *PASCO 2010 in Grenoble, France* (2010), pp. 105–111.

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

[131] Mora, T. *Solving Polynomial Equation Systems II:Macaulay's Paradigm and Gröbner Technology: Macaulay's Paradigm and Gröbner Technology: v. 2 (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, 2005.

[132] Mora, T. The FGLM Problem and Möller's Algorithm on Zero-dimensional Ideals. *in Gröbner Bases, Coding, and Cryptography, Springer*, 1 (2009), 27–45.

[133] Mora, T., and Robbiano, L. The Gröbner fan of an ideal. *Journal of Symbolic Computation*, 6 (1988), 183–208.

[134] Noro, M. An Efficient Implementation for Computing Gröbner Bases over Algebraic Number Fields. In *Mathematical Software – ICMS 2006*, Iglesias, A. and Takayama, N., Eds., vol. 4151 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 99–109.

[135] Noro, M. Modular Algorithms for Computing a Generating Set of the Syzygy Module. In *Computer Algebra in Scientific Computing*, Gerdt, V., Mayr, E., and Vorozhtsov, E., Eds., vol. 5743 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009, pp. 259–268.

[136] Pan, V. Y., and Wang, X. On Rational Number Reconstruction and Approximation. *SIAM J. Comput. 33* (February 2004), 502–503.

[137] Pauer, F. On lucky ideals for Gröbner basis computations. *Journal of Symbolic Computation 14*, 5 (1992), 471 – 482.

[138] Pfister, G. On Modular Computation of Standard Basis. *Analele Stiintifice al Universtitatii Ovidius, Mathematical Series XV*, 1 (2007), 129 – 137.

[139] Pommaret, J. F. Systems of Parital Differential Equations and Lie Pseudogroups. *Gordon & Breach, New York* (1978).

[140] Rau, J. *Tropical intersection theory and gravitational descendants: Intersections of tropical cycles and applications to enumerative geometry*. Südwestdeutscher Verlag für Hoschulschriften, 2010.

[141] Sala, M., Mora, T., Perret, L., Sakata, S., and Traverso, C., Eds. *Gröbner Bases, Coding, and Cryptography*. Springer Verlag, 2009.

[142] Sasaki, T., and Takeshima, T. Modular Method for Grobner-basis Construction over Q and Solving System of Algebraic Equations. *Journal of information processing 12*, 4 (1989), 371–379.

[143] Schwartz, N. Stability of Gröbner bases. *Journal of Pure and Applied Algebra*, 53 (1988), 171–186.

[144] Stegers, T. Faugère's F5 Algorithm revisited. Master's thesis, Technische Univerität Darmstadt, revised version 2007.

[145] Sturmfels, B. *Gröbner Bases and Convex Polytopes*. University Lecture Series, No. 8, 1996.

[146] Sturmfels, B. Solving Systems of Polynomial Equations, 2002.

[147] Sturmfels, B. *Combinatorial Commutative Algebra*. Graduate Texts in Mathematics, Springer Verlag, 2004.

[148] Sun, Y., and Wang, D. A New Proof of the F5 Algorithm. *CoRR abs/1004.0084* (2010).

[149] Sun, Y., and Wang, D. A generalized criterion for signature related Gröbner basis algorithms. In *ISSAC 2011: Proceedings of the 2011 international symposium on Symbolic and algebraic computation* (2011), pp. 337–344.

[150] Sylvester, J. J. *Collected Mathematical Papers of James Joseph Sylvester*. Cambridge University Press, Cambridge, England, 1912.

[151] Thomas, J. Differential Systems. *American Mathematical Society, New York* (1937).

[152] Tran, Q.-N. A fast algorithm for Gröbner basis conversion and its applications. *Journal of Symbolic Computation 30*, 4 (2000), 451–467.

[153] Traverso, C. Gröbner Trace Algorithms. In *ISSAC'88* (1988).

[154] Traverso, C. Hilbert Functions and the Buchberger Algorithm. *Journal of Symbolic Computation 22*, 4 (1996), 355–376.

[155] Wall, B. On the computation of syzygies. *ACM SIGSAM Bulletin 23* (1989), 5–14.

[156] Wang, P. S. Parallel polynomial operations on SMPs: an overview. *Journal of Symbolic Computation 21* (1996), 397–410.

[157] Wang, P. S., Guy, M. J. T., and Davenport, J. H. P-adic reconstruction of rational numbers. *ACM SIGSAM Bulletin 16* (May 1982), 2–3.

[158] Wang, X., and Pan, V. Y. Acceleration of Euclidean algorithm and rational number reconstruction. *SIAM Journal on Computing 32* (2003), 548–556.

[159] Wichmann, T. Der FGLM-Algorithmus: verallgemeinert und implementiert in Singular. *Diploma thesis at the university of Kaiserslautern* (1997).

[160] Winkler, F. A p-adic approach to the computation ofGröbner bases. *Journal of Symbolic Computation 6* (December 1988), 287–304.

[161] Yanovich, D. A. Parallelization of an Algorithm for Computation of Involutive Janet Bases. *Program. Comput. Softw. 28* (March 2002), 66–69.

[162] Zharkov, A. Y. *Solving zero-dimensional involutive systems*. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1996, pp. 389–399.

[163] Zobnin, A. I. Generalization of the F5 algorithm for calculating Gröbner bases for polynomial ideals. *Programming and Computer Software 36* (2010), 75–82. 10.1134/S0361768810020040.

Index

addF5Crit, 187
AP, 162

basis of a module, 4
bijective
  ring homomorphism, 8
Buchberger's 1st Criterion, 50
  extended version, 50
Buchberger's 2nd Criterion, 51
Buchberger's Algorithm, 37

canonical basis of a module, 4
Chain Criterion, 51
characteristic, 10
  of a field, 10
  of a ring, 10
coefficient, 19
coefficient growth, 88
coefficient of a polynomial, 11
complete intersection, 73
complete labeled polynomial, 261
complete sig–safe reduction, 132
complex, 112
  exact sequence, 112
  of modules, 112
complexity classes, 41
complexity of an algorithm, 41
complexity of standard basis computations, 41
constant, 11
convex polyhedral cone, 79
  dimension, 79
  dual, 79
  face, 79

facet, 79
critical pair, 37
normal, 163
of labeled polynomials, 133
sig–equivalent, 133
useless, 45
curr–index labeled polynomial, 249
polynomial part of the label, 249
cyclic module, 4

degree
of a module monomial, 19
of a monomial, 11
of a polynomial, 12
of a signature, 130
weighted, 71
degree of a variable, 11
dehomogenization of a polynomial, 23
Dickson's Lemma, 14
divisibility
of monomials, 14
involutive monomial, 98, 99
of module monomials, 21
dual of a convex polyhedral cone, 79
dual pairing, 79
dynamic Gröbner basis algorithm, 77

ecart
module element, 21
polynomial, 17
weighted, 71
Euclidean algorithm, 26
exact sequence, 112
extended Product Criterion, 50
extended version of Buchberger's 1st Criterion, 50

F4 Algorithm, 56, **57**
Gaussian elimination, 56
improved version, 64
reduction process, 60, 67
simplifying reduction, 68
symbolic preprocessing, 57, 59
F5 Algorithm, 103, 115, **184**, 185
computing syzygies, 262
normalized critical pair, 186
normalized element, 186
rule, 192
rules, 192
F5 Criterion, 186
F5+ Algorithm, 228
F5–critical pair, 225
F5B Algorithm, 223
F5C Algorithm, 207
F5t Algorithm, 223
face of a convex polyhedral cone, 79
facet, 79
facet, 79
fan, 79
Farey fractions, 89
Farey rational map, **89**, 91
FGB, 62
FGLM Algorithm, 85, **87**, 100
finitely generated module, 4
free module, 4
free resolution, 104
of finite length, 112
free resolution of a module, 112

G2V, 168
super top–reduction, 169
Gauss basis, 104
Gauss generating set, 104
Gaussian elimination, 26, 55, 56
Gebauer–Möller implementation, 53
GFAN, 85
global monomial order, 15
Gröbner basis, 26, *see also* standard basis
change of order, 77, 83, 85
dynamic algorithm, 77
FGLM Algorithm, 85, 87
Gröber fan, 82
Gröbner cone, 82
Gröbner walk, 83
staggered linear basis, 105
Gröbner cone, 82
Gröbner fan, 77, 82
convex polyhedral cone, 79
fan, 79
Gröbner trac algorithm

Gröbner trace reconstruction algorithm, 95
modular, 97
Gröbner trace, 94
Gröbner trace algorithm, 94
Gröbner trace, 94
Gröbner trace reconstruction algorithm, 95
modular, 97
Gröbner trace reconstruction algorithm, 95, **95**
Gröbner walk, 83
Gröbner cone, 82
Gröbner fan, 82
graded lexicographical monomial order, 15
graded module, 22
homogeneous element, 22
inhomogeneous element, 22
graded reverse lexicographical monomial order, 15
graded ring, 22
homogeneous element, 22
inhomogeneous element, 22
greatest common divisor, 35

height of a prime ideal, 73
height of an ideal, 73
Hilbert function, 24
Hilbert polynomial, 25, **25**
Hilbert–driven standard basis computation, 72, **74**
Hilbert–lucky prime number, 90
Hilbert–Poincaré series, 25
Hilbert–driven standard basis computation, 72
Hilbert–Samuel function, 26
homogeneous
standard basis, 49
homogeneous element, 22
homogeneous ideal, 22
homogeneous module, 22
homogeneous polynomial, 23
homogenization of a polynomial, 23
homomorphism
module homomorphism, 7
ring homomorphism, 7

ideal, 3
height of, 73
initial, 81
leading ideal, 27
maximal ideal, 4
primary ideal, 4
prime ideal, 4
principal, 4
quotient, 4
radical of an ideal, 4
saturation, 4
vanishing set, 73
variety, 73
zero–dimensional, 86
ideal quotient, 4
image
of a module homomorphism, 8
of a ring homomorphism, 8
improved F4 Algorithm, 64, **65**
symbolic preprocessing, 66
improved version of the F4 Algorithm
simplifying reduction, 68
IncF5, 191
IncF5+, 230
IncF5B+, 235
IncF5C, 210
IncF5E, 215
IncF5Syz, 265
incremental standard basis computation, 137
IncSig, 138
IncSigCrit, 154
IncSigG2V, 172
index
of a labeled polynomial, 130
index of a module element, 19
induced homogenized order, 23
inhomogeneous element, 22
initial ideal, 81
initial monomial, 81
injective
module homomorphism, 8

ring homomorphism, 8
interreduced standard basis, 28
involutive autoreduced set, 99
involutive basis, **99**
autoreduced set, 99
involutive set, 99
monomial divison, 98, 99
normal form, 99
involutive monomial division, 98, **99**
involutive normal form, 99
involutive set, 99
isomorphism, 8

kernel
of a module homomorphism, 8
of a ring homomorphism, 8

label
of a labeled polynomial, 128
labeled polynomial, 128
coefficient of the signature, 130
complete, 261
critical pair
normal, 163
critical pair sig–equivalent, 133
curr–index, 249
polynomial part of the label, 249
degree, 130
degree of signature, 130
index of, 130
label, 128
leading coefficient, 130
leading monomial, 130
leading monomial of the label, 130
leading term, 130
least common multiple, 130
monomial part of the signature, 130
non–minimal signature criterion, 151
polynomial part, 128
redundant, 221
rewritable signature criterion, 151
sig–redundant, 136
signature of, 128
slim, 143
standard representation, 131
term of the signature, 130
labels of a polynomial, 127
leading coefficient
module element, 21
polynomial, 17
leading ideal, 27
boundary of, 85
edges of, 85
monomials not reducible by, 85
sides of, 85
leading monomial, 17
module element, 21
leading submodule, 27
leading term
module element, 21
polynomial, 17
least common multiple, 35
lexicographical monomial order, 15
line segment between two vectors, 81
local monomial order, 15
local ring, 12
Hilbert–Samuel function, 26
localization of polynomial ring, 17
localization of the polynomial ring, 13
lucky prime number, 90

Macaulay matrix, 56
Magma, 62
Maple, 62, 260
maximal ideal, 4
mixed monomial order, 15
modular Gröbner trace algorithm, 97, **97**
modular standard basis computation, 89, **92**
Farey fractions, 89
Farey rational map, 89, 91
Gröbner trace algorithm, 94
Hilbert–lucky prime number, 90
lucky prime number, 90
parallelization, 92
module, 3
basis, 4
canonical basis, 4
cyclic, 4
finitely generated module, 4

free module, 4
generators of an module, 4
graded, 22
greatest common divisor, 35
Hilbert function, 24
Hilbert polynomial, 25
Hilbert–Poincaré series, 25
homomorphism, 7
leading submodule, 27
monomial order, 19
monommial, 19
Noetherian, 6
quotient module, 5
rank of a module, 4
regular sequence, 74
residue class, 5
scalar multiplication, 3
submodule, 3
term, 19

module element
divisibility, 21
ecart, 21
leading coefficient, 21
leading monomial, 21
leading term, 21
least common multiple, 35
monic, 21
monomial support, 19
s–vector, 26, 35, 36
support, 19
tail, 21

module homomorphism, 7
image, 8
injective, 8
kernel, 8
preimage, 8

module monomial order, 19

monic
module element, 21
polynomial, 17

monoid module, *see* monomodule

monomial
degree, 11
degree of a variable, 11
divisibility, 14
in a module over a polynomial ring, 19
in a polynomial ring, 11

monomial order, 14
Dickson's Lemma, 14
global, 15
graded lexicographical, 15
graded reverse lexicographical, 15
induced homogenized, 23
lexicographical, 15
local, 15
mixed, 15
negative graded lexicographical, 15
negative graded reverse lexicographical, 15
negative lexicographical, 15
product, 16
refinement, 82
weight, 16

monomial support
of a module element, 19

monomodule, 3
Noetherian, 6

multiplicatively closed set, 17

natural order, 14
negative graded lexicographical monomial order, 15
negative graded reverse lexicographical monomial order, 15
negative lexicographical monomial order, 15

Noetherian
module, 6
monomodule, 6
Noetherian ring, 6

non–minimal signature criterion, 151
NonMin?, 155
NonMinAP?, 164
NonMinF5?, 187
NonMinG2V?, 170
normal critical pair, 163

normal form, 26
global, 31
local, 33
reduced, 30, 31

standard representation, 30
weak, 29
normalized critical pair, 186
normalized element, 186

order
global, 15
local, 15
mixed, 15
module monomial, 19
monomial, 14
natural, 14
partial, 14
refinement of a weight vector order, 82
strict partial, 13
strict total, 13
total, 14
well–order, 14
ordered list, 27

pair set, 37
critical pair, 37
parallel modular standard basis computation, 92
partial order, 14
PolyBoRi , 66
polynomial, 10, 12
coefficient of a, 11
constant, 11
degree of, 12
dehomogenization of, 23
ecart, 17
greatest common divisor, 35
homogeneous, 23
homogenization of, 23
initial monomial, 81
labels, 127
leading coefficient, 17
leading monomial, 17
leading term, 17
least common multiple, 35
monic, 17
monomial support, 12
s–polynomial, 36
support, 12
tail, 17
total degree of, 12
w–homogeneous, 81
weighted degree, 71
polynomial ring, 11
localization, 13, 17
monomial, 11
polynomial, 12
term, 11
preimage
of a module homomorphism, 8
of a ring homomorphism, 8
primary ideal, 4
prime ideal, 4
height of, 73
principal ideal, 4
principal ideal ring, 4
principal syzygy, 112
Product Criterion, 50
extended version, 50
product monomial order, 16

quotient map, 9
quotient module, 5
quotient ring, 5

radical of an ideal, 4
rank of a module, 4
reduced standard basis, 28
reduction, 28, 34
top–reduction, 34
redundant labeled polynomial, 221
refinement of a weight vector order, 82
regular sequence, 74
Replace? in SlimGB, 70
Coefficient–elimination strategy, 70
Coefficient–length strategy, 69
Elimination strategy, 69
Length strategy, 69
Property, 69
residue class, 5
rewritable signature criterion, 151
Rewrite?, 156
RewriteAP?, 165
RewriteG2V?, 171

RewriteMM?, 167
Rewritten Criterion, 192
ring, 2
  graded, 22
  homomorphism, 7
  local, 12
  Noetherian, 6
  principal ideal domain, 4
  principal ideal ring, 4
  quotient ring, 5
  subring, 2
ring homomorphism, 7
  bijective, 8
  image, 8
  injective, 8
  isomorphism, 8
  kernel, 8
  preimage, 8
  quotient map, 9
  surjective, 8

s–polynomial
  least common multiple, 35
s–vector, 26, 35, **36**
  least common multiple, 35
  of labeled polynomials, 132
saturation of an ideal, 4
SB–critical pair, 225
scalar multiplication, 3
selection strategy
  normal, 47
  sugar, 48
semi–complete sig–safe reduction, 132
sequence, 27
set of generators, 4
sig–equivalent critical pair, 133
sig–redundant labeled polynomial, 136
sig–safe reduction, 132
  complete, 132
  semi–complete, 132
sig–standard basis, 163
sig–unsafe reduction, 132
signature
  of a labeled polynomial, 128
  coefficient, 130
  degree, 130
  generalized, 253
  labeled polynomial, 128
  leading monomial, 130
  monomial part, 130
  non–minimal, **151**
    NonMin?, 155
    NonMinAP?, 164
    NonMinF5?, 187
  of a module element, 127
  of a polynomial, 128
  of length j, 253
  rewritable, **151**
    Rewrite?, 156
    RewriteAP?, 165
    RewriteMM?, 167
  set of all signatures of a polynomial, 128
  sig–safe reduction, 132
  sig–unsafe reduction, 132
  term, 130
SigRed, 139
SigRedF5, 189
SigRedF5+, 231
SigRedF5Syz, 266
SigRedG2V, 169
SigStd, 136
SigStdQ, 250
SigStdRed, 159
simplifying reduction, 68
Singular , 55, 66, 92
slim labeled polynomial, 143
SlimGB, **69**
  Replace?, 70
  SlimNF, 67, 70
SlimNF, 67, **70**
staggered linear basis, 104, 105, **110**
staggered linear basis algorithm, **108**
  initial version, 108
  normal form, 111
  revised version, 110
standard basis
  sig–, 163
standard basis, 1, 26, **27**
  normal form, 29

Buchberger's 1st Criterion, 50
Buchberger's 2nd Criterion, 51
Buchberger's Algorithm, 37
Chain Criterion, 51
coefficient growth, 88
complexity, 41
critical pair, 37
dynamic algorithm, 77
extended Product Criterion, 50
extended version of Buchberger's 1st Criterion, 50
F4 Algorithm, 56
F5 Algorithm, 184
FGLM Algorithm, 85, 87, 100
Gebauer–Möller implementation, 53
Gröbner basis, 27
Gröbner fan, 77
Gröbner trace algorithm, 94
Gröbner walk, 83
Hilbert–driven, 72, 74
homogeneous, 49
improved F4 Algorithm, 64
incremental computation, 137
interreduced, 28
leading ideal, 27
leading submodule, 27
modular computation, 89, 92
non–minimal signature criterion, 151
normal form, 26
normal selection strategy, 47
of degree d, 49
pair set, 37
parallelized computations, 92
Product Criterion, 50
product criterion, 50
reduced, 28
reduced normal form, 30
reduction, 34
regular sequence, 74
rewritable signature criterion, 151
s–polynomial, 36
s–vector, 26, 36
SigStd, 136
SigStdQ, 250
SigStdRed, 159
SlimGB, 66
SlimNF, 67, 70
standard representation, 30, 36
sugar selection strategy, 48
SyzNF, 119
SyzStd, 118
syzygy–based computation, 118
top–reduction, 34
useless critical pair, 45

standard grading, 23
standard representation, 30
- of a labeled polynomial, 131
- s–vector, 36

strict partial order, 13
strict total order, 13
submodule, 3
subring, 2
sugar degree, 48
super top–reduction, 169
support
- monomial support of a polynomial, 12
- of a module element, 19
- of a polynomial, 12

surjective
- ring homomorphism, 8

Sylvester matrix, 56
symbolic preprocessing, 57, **59**, 66
Syz1, 114
Syz2, 116
SyzNF, **119**
SyzStd, 118
syzygy, 103, 112
- matrix, 117
- principal, 112

syzygy algorithms, 114, 116
- Syz1, 114
- Syz2, 116

syzygy matrix, 117
syzygy module, 112

tail
- module element, 21
- polynomial, 17

term
- in a module, 19

in a polynomial ring, 11
top–reduction, 34
toric geometry, 81
total degree of a polynomial, 12
total order, 14

useless critical pair, 45

vanishing set
complete intersection, 73
vanishing set of an ideal, 73
variety
complete intersection, 73
variety of an ideal, 73
vector space, 3

w–homogeneous polynomial, 81
weight
ecart, 71
weight order, 16
weighted degree of a polynomial, 71
weighted ecart, 71
well–order, 14

Wissenschaftlicher Werdegang

2002	Abitur am Carl-Bosch-Gymnasium, Ludwigshafen
seit 04/2002	Studium der Mathematik an der TU Kaiserslautern
12/2005	Diplom in Mathematik, TU Kaiserslautern
seit 04/2008	Doktorand bei Prof. Dr. Gerhard Pfister, TU Kaiserslautern

Curriculum Vitae

2002	Abitur at the Carl-Bosch-Gymnasium, Ludwigshafen
since 04/2002	Study of mathematics at the University of Kaiserslautern, Germany
12/2005	Diplom in Mathematics, University of Kaiserslautern
since 04/2008	Ph.D. studies with Prof. Dr. Gerhard Pfister, University of Kaiserslautern