A Generic and Executable Formalization of Signature-Based Gröbner Basis Algorithms

Alexander Maletzky[1]

Research Institute for Symbolic Computation (RISC), Johannes Kepler University Linz, Altenberger Strasse 69, A-4040 Linz, Austria

Abstract

We present a generic and executable formalization of signature-based algorithms (like Faugère's F_5) for computing Gröbner bases, as well as their mathematical background, in the Isabelle/HOL proof assistant. Said algorithms are currently the best known algorithms for computing Gröbner bases in terms of computational efficiency, and so we believe that their formalization in a proof assistant represents an interesting and useful piece of work. The formal development attempts to be as generic as possible, generalizing most known variants of signature-based algorithms, but at the same time the implemented functions are effectively executable on concrete input for efficiently computing provenly correct Gröbner bases. Moreover, besides correctness the formalization also proves that under certain conditions the algorithms a-priori detect and avoid all useless reductions to zero, and return 'minimal' (in some sense) Gröbner bases.

To the best of our knowledge, the formalization presented here is the only formalization of signature-based Gröbner basis algorithms in existence so far.

Keywords: Gröbner bases, signature-based algorithms, interactive theorem proving, Isabelle/HOL

1. Introduction

Gröbner bases, introduced by Buchberger (1965), are a ubiquitous tool in computer algebra and beyond, as they allow to effectively solve many problems related to multivariate polynomial rings and ideals. Finding Gröbner bases is a computationally difficult task, and therefore many researchers have attempted to design more and more efficient algorithms over the years. This, finally, lead to the first *signature-based algorithm*, the F_5 algorithm invented by Faugère (2002). Nowadays, F_5 and its relatives are the most efficient algorithms for computing Gröbner bases, implemented in many modern computer algebra systems.

The work presented in this paper focuses on yet another implementation of signature-based algorithms, but this time not in a computer algebra system but in the proof assistant Isabelle/HOL (Nipkow et al. (2002)). The distinctive feature of our implementation

Email address: alexander.maletzky@risc.jku.at (Alexander Maletzky)

[1]The research was funded by the Austrian Science Fund (FWF): P 29498-N31.

is its being *formally verified* by the inference kernel of Isabelle. This, of course, necessitated formalizing also the vast theory behind signature-based algorithms, eventually leading to an *extensive*, *generic* and *executable* formalization of an important and highly interesting topic in modern computer algebra. Even more, it is the—to the best of our knowledge—first-ever formalization of this theory in *any* proof assistant. As such, it constitutes the ultimate certificate that the existing informal theory is indeed correct, without even the slightest mistake or overlooked gap.

In the remainder we assume familiarity with the basics of Gröbner bases theory, like polynomial reduction, S-polynomials, the definition of Gröbner bases, and Buchberger's algorithm. Although we present the key definitions, theorems and algorithms of the signature-based approach to Gröbner bases in this paper, readers totally new to the subject might also want to have a glance at the excellent survey article by Eder and Faugère (2017), which we took as the template for our formalization. We must also stress that besides Faugère, many more researchers have worked on signature-based algorithms, resulting in a great variety of such algorithms. Giving an exhaustive overview of *all* variations of signature-based algorithms in existence is out of scope here, though; see again Eder and Faugère (2017) instead.

The motivation and distinctive feature of signature-based algorithms is detecting and avoiding many more useless zero-reductions while computing Gröbner bases than other algorithms—and in some cases even *all* useless reductions. This saves a lot of computation time and thus leads to a drastic speed-up. How all this relates to signatures, and what signatures are in the first place, will be explained in Sections 3–5.

The main motivation for the formalization was to verify a state-of-the-art algorithm for computing Gröbner bases. That task could be expected to be challenging due to the inherent complexity of the underlying mathematical theory, illustrated by the fact that termination of the original F_5 algorithm was an open problem for a decade until it was settled by Galkin (2012). Our formal development of the theory may also serve as the basis for further theoretical investigations, e. g. by implementing, testing and verifying new improvements of the formalized algorithms, as well as for further formalizations in the vast area of Gröbner bases.

Summarizing, the key features of the work presented here are as follows:

- The formalization is generic, in the sense that we consider so-called *rewrite bases* and allow for arbitrary term orders and rewrite orders (Section 5). According to Eder and Faugère (2017) this set-up covers most, if not all, existing variations of signature-based algorithms.

- All algorithms are proved to be totally correct w. r. t. their specifications. In particular, the algorithm for computing rewrite bases (Algorithm 1) is shown to terminate for every input (Section 6.1).

- Besides correctness, we also prove that under certain conditions the algorithm indeed avoids *all* useless zero-reductions, and that with a particular choice of the rewrite order it returns minimal signature Gröbner bases (Section 7).

- All formally verified algorithms are efficiently executable on concrete input. 'Efficient' means that, for instance, the algorithms operate only on so-called *sig-poly-pairs* rather than full module elements (Section 8).

The entire formalization is freely available online (Maletzky (2018)) for the current version of Isabelle, as an entry of Isabelle's *Archive of Formal Proofs* (AFP).[2] Note that this guarantees its logical soundness, because the AFP only accepts correct and complete (in the sense that no proofs are missing) entries.

1.1. Organization of the Paper

The rest of the paper is organized as follows: Section 2 gives a brief overview of Isabelle/HOL, to make the paper as self-contained as possible. Section 3 presents the preliminaries of signature-based algorithms, Section 4 introduces $\mathfrak{s}$-reduction and signature Gröbner bases, Section 5 defines rewrite bases and S-pairs and establishes the connection between them, Section 6 presents the main algorithms and hints why they are totally correct, and Section 7 contains two results concerning the 'optimality' of the algorithms. Each of these sections first presents the various concepts and theorems in common mathematical style, before showing how they are formalized in Isabelle/HOL.

Section 8, then, explains how the formalized algorithms can be executed on concrete input and provides a comparison of the running times of these algorithms to other algorithms implemented in Isabelle/HOL and in *Mathematica*. Section 9, finally, concludes the paper by giving quantitative information on the formalization effort and listing related and future work.

2. Brief Overview of Isabelle/HOL

The purpose of this section is to give a brief overview of the most important aspects of Isabelle/HOL that are necessary for understanding the rest of the paper. Further information and documentation can be found in Paulson (1994); Nipkow et al. (2002); Wenzel (2018) and on the Isabelle homepage[3]. Readers already familiar with Isabelle may skip this section.

Isabelle is a *generic proof assistant*: it serves as a framework for implementing different object logics, like first-order logic or higher-order logic, in one single system. As such, it provides the basic infrastructure needed for automated and interactive theorem proving in general: a small inference kernel based on higher-order unification, theory- and proof contexts, a document preparation interface, and many more. Isabelle/HOL is a concrete object logic implemented in Isabelle, namely classical higher-order predicate logic. Being the most actively developed object logic of Isabelle, it comes with a library of hundreds of useful mathematical concepts, such as numbers, sets, lists, abstract algebraic structures, etc., which new formalizations can build upon.

Formalizing a mathematical theory in Isabelle/HOL normally proceeds by *definitional theory extensions*: new concepts are defined, properties of these concepts and their relation to existing concepts are proved, and so on; arbitrary axiomatizations, though possible in principle, are usually avoided to eliminate the risk of introducing inconsistencies to the theory.[4]

[2] `http://www.isa-afp.org`

[3] `http://isabelle.in.tum.de`

[4] Our formalization goes without any such axiomatizations.

2.1. Definitions

New constants can be introduced either via explicit non-recursive definitions or as (potentially recursive) functions. A simple example of the former is the following:

```
definition subset-eq :: "α set ⇒ α set ⇒ bool" (infix "⊆" 50)
  where "subset-eq A B ⟷ (∀a∈A. a ∈ B)"
```

This definition introduces a new constant, subset-eq, of type α set $\Rightarrow$ α set $\Rightarrow$ bool. That means, it is a function taking two sets of element-type α as arguments and returning a boolean value. α and other Greek letters always denote *type variables*; hence, subset-eq is a *polymorphic* function that cannot only be applied to sets of a particular element-type, but to all sets. set and bool are built-in type constructors of Isabelle/HOL. The 'infix' clause following the type is optional and instructs Isabelle to record the short infix notation $\subseteq$ for subset-eq. The actual definition of subset-eq comes after the 'where' keyword: subset-eq holds for two arguments A and B[5] if, and only if, every element a of A is also an element of B. $\forall$ and $\in$ are built-in constants with the usual meaning, and therefore subset-eq is indeed the usual subset relation.[6]

As can be seen, Isabelle uses so-called *Curried notation* for denoting function application: in the definition, subset-eq is applied to A and B by mere juxtaposition, without parentheses. Parentheses only become necessary in nested function applications, as in $f\ (g\ x)$. What can also be seen in the above definition is that single arrows are used to denote equivalences and implications: $\longrightarrow$ for logical implication, $\longleftrightarrow$ for equivalence.

Notation 2.1. Within enclosing informal text, we will adopt the standard mathematical notation for function application when writing Isabelle code. For instance, we shall write subset-eq(A, B) rather than subset-eq A B, because the latter does not fit very well with informal text. Furthermore, names of constants will be typeset in sans serif font for distinguishing them from variables, which will be typeset in *italics*, as usual.

As an example of a recursive definition, consider the following:

```
function set :: "α list ⇒ α set" where
  "set [] = {}" |
  "set (x # xs) = {x} ∪ set xs"
```

This command defines a polymorphic function set which maps lists (built-in type constructor list) to the set of their elements, which is achieved by structural recursion on the shape of the argument: if it is empty, the empty set is returned; otherwise, the list consists of a head x and a tail xs, in which case set is applied to xs recursively and x is added to the result. Recursively defined functions must always be shown to terminate, to avoid potential inconsistencies. In some cases, Isabelle can do the termination proofs itself, whereas in more difficult situations the user has to construct the proofs interactively.

[5] Free variables in definitions, theorems, etc. are implicitly universally quantified.

[6] subset-eq is of course also a built-in constant of Isabelle/HOL.

2.2. Theorems and Proofs

Theorems can be stated using the synonymous 'lemma', 'theorem' or 'corollary' keywords. For instance, a lemma expressing that the set of elements of the concatenation of two lists equals the union of the individual sets could be stated as follows:

```
lemma set-append: "set (xs @ ys) = set xs ∪ set ys"
```

Here, `@` is infix syntax denoting the concatenation of lists xs and ys, and recall from above that the two free variables xs and ys are implicitly universally quantified. So far, however, the lemma is only an unproved claim as far as Isabelle is concerned, so we now have to prove it. Proving in Isabelle rests on two pillars:

- An intuitive, human-readable formal proof language, called *Isar*, for proving theorems interactively. That means, the user writes down the individual steps of the proof, and Isabelle checks whether they are indeed correct.
- A huge machinery of automatic proof methods that are able to prove certain goals automatically, saving the user from doing tedious but (more or less) simple proofs manually. Existing automation is fairly sophisticated, incorporating even powerful state-of-art first-order reasoners.

To give a rough idea of how proofs in Isabelle/HOL look like, we show a (quite verbose) induction proof of the above lemma; long dashes (−) indicate explanatory comments:

```
lemma set-append: "set (xs @ ys) = set xs ∪ set ys"
proof (induction xs)
  −Induction base:
  show "set ([] @ ys) = set [] ∪ set ys" by simp
         −Prove the goal by simplification w.r.t. the definitions of ‹set› and ‹@›.
next
  −Induction step:
  fix x xs      −Choose fresh ‹x› and ‹xs› arbitrary, but fixed.
  assume "set (xs @ ys) = set xs ∪ set ys"      −Assume the induction hypothesis.
  then show "set ((x # xs) @ ys) = set (x # xs) ∪ set ys" by simp
         −Prove the goal again by simplification,
           but this time also using the induction hypothesis.
qed
```

Since we will not present any Isabelle-proofs in the remainder of this paper, we do not say more about proving in Isabelle here.

Finally, please note that more complicated lemmas involving assumptions can be stated following the 'fixes'/'assumes'/'shows' pattern, to increase readability:

```
lemma times-mono-int:
  fixes a b c :: int
  assumes "a ≤ b" and "0 ≤ c"
  shows "a * c ≤ b * c"
```

The (optional) 'fixes' clause locally fixes variables and potentially annotates them with types (here `int`, the type of integers). The (optional) 'assumes' clause states one or more assumptions, and the (mandatory) 'shows' clause states the ultimate conclusion.

Notation 2.2. Within enclosing informal text, names of lemmas and theorems will be typeset in *italics*, like *set-append*.

2.3. Frequently Used Functions

We conclude this section by listing built-in concepts we will use later on.

- The usual logical connectives and quantifiers. Syntax in Isabelle/HOL closely resembles ordinary mathematical notation, expect that logical implication is denoted by $\longrightarrow$ and equivalence by $\longleftrightarrow$.
- The usual operations from set theory, whose Isabelle-syntax resembles mathematical notation, too.
- f `` A, which denotes the image of set A under function f.
- $\{0..<n\}$, which denotes the set of natural numbers from 0 (inclusive) up to n (exclusive); analogously, $[0..<n]$ denotes the list of natural numbers from 0 up to n.
- set, [] and #, as explained above: set(xs) is the set of elements of list xs, [] is the empty list, and $x \# xs$ is the list whose first element is x and whose tail is xs.
- length(xs), which is the length of list xs.
- xs ! i, which is the i-th element of list xs (starting from 0).
- fst and snd, which project pairs of type $\alpha \times \beta$ onto their first and second entries, respectively. For instance, fst$((a, b)) = a$.

Remark 2.1. Throughout the paper we will follow the common convention in papers about Isabelle of using dashes instead of underscores, for the sake of better readability. So, subset-eq would in reality be subset_eq in the actual Isabelle sources.

3. Preliminaries

3.1. Mathematical Preliminaries

In this and the subsequent sections we present signature-based Gröbner basis algorithms and their formalization in Isabelle/HOL. Notation is mainly borrowed from Eder and Faugère (2017), with some small adjustments here and there to resemble the notation we use in the formalization. In fact, since the formalization itself closely follows Sections 4–7 of Eder and Faugère (2017), most of the mathematical details omitted in this exposition for the sake of brevity can be found there instead. The informal proofs that served as the templates for our formal development were exclusively taken from the above-mentioned article and from Roune and Stillman (2012) and Eder and Roune (2013).

In the remainder of this paper let $\mathfrak{K}$ be a field and let $\mathfrak{R} = \mathfrak{K}[x_1, \ldots, x_n]$ be the n-variate polynomial ring over $\mathfrak{K}$. Every polynomial $p \in \mathfrak{R}$ can be written as a $\mathfrak{K}$-linear combination of *power-products*, where a power-product is a product of the indeterminates $x_1, \ldots, x_n$, like $x_1^2\, x_2\, x_3^4$. We will write $[X]$ for the commutative monoid of power-products in $x_1, \ldots, x_n$ and typically denote power-products by the typed variables s and t, unless stated otherwise.

Now, fix a finite sequence $F = (f_1, \ldots, f_m)$ of polynomials in $\mathfrak{R}$; these polynomials play the role of the set we want to compute a Gröbner basis of. F gives rise to a module-homomorphism $\bar{\cdot} : \mathfrak{R}^m \to \mathfrak{R}$ by setting $\overline{\mathbf{e}_i} := f_i$ for $1 \leq i \leq m$ and canonical basis vectors $\mathbf{e}_i$ of the free module $\mathfrak{R}^m$. A module element $a \in \mathfrak{R}^m$ is called a *syzygy* of F if $\overline{a} = 0$. Note that $\mathfrak{R}^m$ can be viewed as a $\mathfrak{K}$-vector space, meaning that every $a \in \mathfrak{R}^m$ can be written as a $\mathfrak{K}$-linear combination of *terms*, where a term is a product of the form $t\,\mathbf{e}_i$ for some power-product t and some $1 \leq i \leq m$. We will write $\mathfrak{T}$ for the set of terms and typically denote terms by the typed variables u and v. For a term $v = t\,\mathbf{e}_i$, t is called the power-product of v and i is called the component of v.

For a polynomial $p \in \mathfrak{R}$, $\mathrm{supp}(p)$ is the *support* of p, which the set of all power-products appearing in p with non-zero coefficient. Likewise, $\mathrm{supp}(a)$ for $a \in \mathfrak{R}^m$ is the set of all terms appearing in a with non-zero coefficient. $\mathrm{coeff}(p, t)$ denotes the coefficient of power-product t in $p \in \mathfrak{R}$, and analogous for $\mathrm{coeff}(a, u)$ with $u \in \mathfrak{T}$ and $a \in \mathfrak{R}^m$.

Finally we must also fix an admissible order relation $\preceq$ on $[X]$ and some compatible extension $\preceq_{\mathsf{t}}$ as a term order on $\mathfrak{T}$. *Compatible*, in this context, just means that $s \preceq t$ if, and only if, $s\,\mathbf{e}_i \preceq_{\mathsf{t}} t\,\mathbf{e}_i$, for all $s, t \in [X]$ and $1 \leq i \leq m$. The most important extension of $\preceq$ to a term order is the so-called POT extension, denoted by $\preceq_{\mathsf{pot}}$ and defined as $s\,\mathbf{e}_i \preceq_{\mathsf{pot}} t\,\mathbf{e}_j :\Leftrightarrow i < j \vee (i = j \wedge s \preceq t)$.

Every $p \in \mathfrak{R}$ has a *leading power-product* $\mathrm{lp}(p)$ and a *leading coefficient* $\mathrm{lc}(p)$: if $p \neq 0$, the leading power-product of p is the largest (w. r. t. $\preceq$) power-product appearing in $\mathrm{supp}(p)$, and the leading coefficient is its coefficient; $\mathrm{lp}(0)$ is left undefined and $\mathrm{lc}(0) := 0$. Likewise, every module element $a \in \mathfrak{R}^m$ has a *leading term* $\mathfrak{s}(a)$ and a leading coefficient $\mathrm{lc}(a)$, defined completely analogously w. r. t. $\preceq_{\mathsf{t}}$. The reason why the leading term of a is denoted by $\mathfrak{s}(a)$ rather than $\mathrm{lt}(a)$ becomes clear in the following definition:

Definition 3.1 (Signature). Let $a \in \mathfrak{R}^m$. The *signature* of a is the leading term $\mathfrak{s}(a)$ of a.

Therefore, the all-important *signature* of a module element $a \in \mathfrak{R}^m$ is nothing else but the leading term of a. In the remainder, we will exclusively use the word 'signature' instead of 'leading term'. Note that in contrast to Eder and Faugère (2017), in our case the signature only consists of a term *without* coefficient.

Summarizing, every $a \in \mathfrak{R}^m$ has two important values associated to it: its signature $\mathfrak{s}(a) \in \mathfrak{T}$ and the polynomial $\overline{a} \in \mathfrak{R}$.

3.2. Isabelle/HOL

For our formal development of signature-based Gröbner basis algorithms we did not have to start completely from scratch, but could build upon on existing extensive formalizations of multivariate polynomials and Gröbner bases. Here, we will explain the most important aspects of these formalizations that are relevant for signature-based algorithms; the interested reader is referred to Maletzky and Immler (2018a,b) for more details. The formalizations are freely available as separate entries in the Archive of Formal Proofs (Sternagel et al. (2010); Immler and Maletzky (2016)).

In Isabelle/HOL, multivariate polynomials are represented as so-called *polynomial mappings*, which are functions from some type α to another type β such that all but finitely many values are mapped to 0. The meaning of such a mapping is clear: α plays the role of the power-products or terms, and β plays the role of the coefficient-ring;

the value a power-product or term is mapped to is then precisely its coefficient in the polynomial. The type of polynomial mappings from α to β is denoted by $\alpha \Rightarrow_0 \beta$.

Terms are simply represented as pairs consisting of a power-product and a component of type nat, the type of natural numbers; hence, if α is the type of power-products, then the type of terms is $\alpha \times \mathsf{nat}$. Note that because of nat being infinite, the components of terms can become arbitrarily large; this point deserves a bit more attention, which will be paid below. Before, we summarize what we have so far:

- Here and henceforth, α will always denote the type of power-products. How exactly power-products are represented is not important, since they essentially only have to form a cancellative commutative monoid and a lattice w. r. t. divisibility. For more details see Maletzky and Immler (2018b).
- τ will abbreviate the type of terms, i. e. the type $\alpha \times \mathsf{nat}$. In the actual formalization, τ only needs to be isomorphic to $\alpha \times \mathsf{nat}$, but this is a mere technicality without any further implications.
- β will always be the type of coefficients, which is required to be a field.
- The polynomial ring $\mathfrak{R}$ hence corresponds to the type $\alpha \Rightarrow_0 \beta$, and the module $\mathfrak{R}^m$ to $\tau \Rightarrow_0 \beta$.

Example 3.1. Let $p = 3x^2 - 2xy + y^3 - 4 \in \mathfrak{K}[x, y]$. Then p corresponds to an object of type $\alpha \Rightarrow_0 \beta$ which maps $x^2 \mapsto 3$, $xy \mapsto -2$, $y^3 \mapsto 1$, $1 \mapsto -4$, and all other power-products $t \mapsto 0$. As indicated above, how the individual power-products are represented is not important here.

Likewise, let

$$a = \begin{pmatrix} 2xy^2 - 3 \\ x^3 + y^3 \end{pmatrix} \in \mathfrak{K}[x, y]^2.$$

Then a corresponds to an object of type $\tau \Rightarrow_0 \beta$ which maps $(xy^2, 0) \mapsto 2$, $(1, 0) \mapsto -3$, $(x^3, 1) \mapsto 1$, $(y^3, 1) \mapsto 1$, and all other terms $u \mapsto 0$. Note in particular that the first component is indexed by 0, the second by 1, etc.

One may now ask the following legitimate question:

> How can $\mathfrak{R}^m$ be represented by $\tau \Rightarrow_0 \beta$, if $\mathfrak{R}^m$ has dimension m (and therefore all terms appearing in its elements have components $\leq m$), but τ allows for arbitrarily large components?

Indeed, strictly speaking $\mathfrak{R}^m$ and $\tau \Rightarrow_0 \beta$ are not isomorphic. However, we can circumvent the problem of components of terms being greater than m (or, in fact $m - 1$, since the first component is indexed by 0) by explicitly putting certain constraints on all module elements of type $\tau \Rightarrow_0 \beta$ appearing anywhere in the formal development. More precisely, we define the set sig-inv-set(m), parameterized over the natural number m, of all module elements whose terms have components in the range $[0, \ldots, m-1]$. Then, we constrain all theorems where it is necessary by the additional condition that all module elements a occurring in the theorem belong to sig-inv-set(m).

Of course, m is not just an arbitrary natural number, but it is the length of the implicitly fixed sequence $F = (f_1, \ldots, f_m)$ of polynomials. So, in the formalization we also fix a sequence, or more precisely a list, *fs* in the implicit theory context:

```
context fixes fs :: "(α ⇒₀ β) list"
```

This instruction ensures that all subsequent definitions, theorems, algorithms, etc. are implicitly parameterized over the list *fs*. Consequently, we can define the set Rm of all 'valid' module elements:

```
definition Rm :: "(τ ⇒₀ β) set"
  where "Rm = sig-inv-set (length fs)"
```

Hence, Rm is the set of module elements whose terms have components in the range $[0, \ldots, \mathsf{length}(fs) - 1]$, and as such precisely corresponds to $\mathfrak{R}^m$.

Remark 3.1.
1. In a dependently-typed system like Coq (Bertot and Castéran (2004)), Rm could be turned into a type, meaning that the additional assumptions $a \in \mathsf{Rm}$ of theorems could be encoded implicitly in the type of a. Isabelle/HOL is only simply-typed, so this approach does not work in our case.
2. The definition of Rm shown above is not exactly the one of the formalization. Namely, analogous to components, we also have to take care that all indeterminates appearing in module elements also appear in *fs*; similar as for the components, this cannot be encoded (easily) in the type α. The details are technical and omitted here for the sake of simplicity.

Having *fs* fixed in the context, we next formalize the module-homomorphism $\bar{\cdot}$, called poly in the formal theory, and prove its characteristic properties:[7]

```
lemma poly-zero: "poly 0 = 0"

lemma poly-plus: "poly (a + b) = poly a + poly b"

lemma poly-mult-scalar: "poly (p ⊙ a) = p * poly a"
```

Here one should note that poly is defined in such a way that these identities hold unconditionally, even if $a, b \notin \mathsf{Rm}$. $p \odot a$ is scalar multiplication of the module element a by the polynomial p. Two further important lemmas about poly describe its relationship to the ideal generated by the elements of *fs*:

```
lemma poly-in-ideal: "poly a ∈ ideal (set fs)"

lemma in-idealE-poly-Rm:
  assumes "p ∈ ideal (set fs)"
  shows "∃a∈Rm. p = poly a"
```

The first lemma obviously expresses that $\mathsf{poly}(a)$ is always an element of the ideal generated by the elements of *fs*, whereas the second lemma states the opposite: every element p of the ideal can be written as $p = \mathsf{poly}(a)$ for some $a \in \mathsf{Rm}$; a being an element of Rm is of particular importance here. $\mathsf{ideal}(B)$ is a predefined notion taken from Sternagel et al. (2010), which denotes the ideal generated by the set B.

The only things that are still missing from Section 3.1 are the order relations $\preceq$ and $\preceq_t$, and the various concepts they induce (leading power-product etc.). Similar to *fs*, the remaining formal development shall be parameterized over these orderings, so we introduce a so-called *locale* to fix them implicitly:

[7] We omit the—slightly technical—definition of poly.

```
locale qpm-inf-term =
  ordered-powerprod ord +
  linorder ord-term
    for ord :: "α ⇒ α ⇒ bool" (infixl "≼" 50)
    and ord-term :: "τ ⇒ τ ⇒ bool" (infixl "≼t" 50) +
  assumes stimes-mono: "u ≼t v ⟶ t ⊗ u ≼t t ⊗ v"
  assumes ord-termI: "fst u ≼ fst v ⟶ snd u ≤ snd v ⟶ u ≼t v"
```

Locales are a sophisticated mechanism for structuring Isabelle-theories into sub-theories that can later be combined in a convenient an efficient way; for more information see Ballarin (2010). In the present case, locale qpm-inf-term does the following:

- It fixes the two relations $\preceq$ and $\preceq_{\mathsf{t}}$, and assumes that $\preceq$ is an admissible order on type α (the power-products) and that $\preceq_{\mathsf{t}}$ is a linear order on type τ (the terms).
- It furthermore assumes the two properties *stimes-mono* and *ord-termI*. *stimes-mono* expresses that $\preceq_{\mathsf{t}}$ is monotonic w. r. t. $\otimes$, where $t \otimes u$ denotes the term obtained from u by multiplying its power-product by t. *ord-termI* states that if the power-product and the component of u are not greater than their respective counterparts of v, then $u \preceq_{\mathsf{t}} v$. Note that $\mathsf{fst}(u)$ gives the first entry of term u, i. e. its power-product, and $\mathsf{snd}(u)$ gives the second entry of u, i. e. its component.

All subsequent definitions, theorems, etc. will be stated in the context of this locale, meaning that they are implicitly parameterized over $\preceq$ and $\preceq_{\mathsf{t}}$ (just as they are parameterized over *fs*), and that furthermore all theorems are implicitly constrained by the two additional assumptions *stimes-mono* and *ord-termI*.

Therefore, leading power-products, -terms and -coefficients of polynomials and module elements can be defined readily. However, since locale qpm-inf-term is in fact part of the existing formalization of multivariate polynomials (Sternagel et al. (2010)) and leading power-products etc. are defined there, too, there is nothing left to be done.

We conclude this section by pointing the reader to Appendix A, containing a 'dictionary' for translating between mathematical notions and notations occurring in this paper, and their counterparts in the formalization.

Remark 3.2. This remark is only relevant for readers interested in the actual Isabelle sources. There, power-products are written *additively* rather than *multiplicatively*, for technical reasons. So, 0 is used instead of 1, + instead of $\cdot$, and $\oplus$ instead of $\otimes$. In this paper we decided to stick to the standard multiplicative writing for the sake of uniformity.

4. Signature Reduction and Signature Gröbner Bases

Let us now turn to the key concept in the theory of Gröbner bases: *polynomial reduction*. In the 'traditional', non-signature approach the reduction relation is a binary relation on polynomials, parameterized over a set of polynomials. In the signature-based world, it becomes a binary relation on *module elements*, i. e. on $\mathfrak{R}^m$, defined as follows:

Definition 4.1 ($\mathfrak{s}$-Reduction). Let $a, b \in \mathfrak{R}^m$ and $G \subseteq \mathfrak{R}^m$. a $\mathfrak{s}$-reduces to b modulo G if, and only if, there exist $g \in G$ and $t \in [X]$ such that

1. $\overline{g} \neq 0$,

2. $t\,\mathrm{lp}(\overline{g}) \in \mathrm{supp}(\overline{a})$,
3. $b = a - t\,g$, and
4. $t\,\mathfrak{s}(g) \preceq_{\mathtt{t}} \mathfrak{s}(a)$, which is equivalent to $\mathfrak{s}(b) \preceq_{\mathtt{t}} \mathfrak{s}(a)$.

So, if a $\mathfrak{s}$-reduces to b modulo G, it simply means that $\overline{a}$ reduces to $\overline{b}$ modulo $\overline{G}$ in the usual sense of polynomial reduction,[8] and that furthermore the signature of b is not greater than that of a. So, in short, $\mathfrak{s}$-reduction is like polynomial reduction with the additional requirement that signatures do not grow.

$\mathfrak{s}$-reduction comes in different flavors, depending on whether $t\,\mathrm{lp}(\overline{g})$ in Definition 4.1 equals $\mathrm{lp}(\overline{a})$ or not, and whether in the last condition of that definition we have $t\,\mathfrak{s}(g) = \mathfrak{s}(a)$ or $t\,\mathfrak{s}(g) \prec_{\mathtt{t}} \mathfrak{s}(a)$. If $t\,\mathrm{lp}(\overline{g}) = \mathrm{lp}(\overline{a})$, we shall say that the $\mathfrak{s}$-reduction is a *top* $\mathfrak{s}$-reduction; otherwise, if $t\,\mathrm{lp}(\overline{g}) \prec \mathrm{lp}(\overline{a})$, we call it a *tail* $\mathfrak{s}$-reduction.[9] If $t\,\mathfrak{s}(g) = \mathfrak{s}(a)$ the $\mathfrak{s}$-reduction is a *singular* $\mathfrak{s}$-reduction, whereas if $t\,\mathfrak{s}(g) \prec_{\mathtt{t}} \mathfrak{s}(a)$ it is a *regular* $\mathfrak{s}$-reduction. It is easy to see that in a regular $\mathfrak{s}$-reduction we always have $\mathfrak{s}(b) = \mathfrak{s}(a)$.

Notation 4.1. Let $\mathrm{r}_1 \in \{\preceq_{\mathtt{t}}, \prec_{\mathtt{t}}, =\}$ and $\mathrm{r}_2 \in \{\preceq, \prec, =\}$. We will use the following notation: $a \xrightarrow{\mathrm{r}_1,\mathrm{r}_2}_G b$ means that a $\mathfrak{s}$-reduces to b modulo G, and that additionally $t\,\mathfrak{s}(g)\ \mathrm{r}_1\ \mathfrak{s}(a)$ and $t\,\mathrm{lp}(\overline{g})\ \mathrm{r}_2\ \mathrm{lp}(\overline{a})$ hold, where t, g are as in Definition 4.1. As usual, $\xrightarrow{\mathrm{r}_1,\mathrm{r}_2*}_G$ denotes the reflexive-transitive closure of $\xrightarrow{\mathrm{r}_1,\mathrm{r}_2}_G$. So, $\xrightarrow{\preceq,\preceq_{\mathtt{t}}}_G$ stands for general $\mathfrak{s}$-reduction, $\xrightarrow{\prec_{\mathtt{t}},=}_G$ for regular top $\mathfrak{s}$-reduction, and so on. To ease notation, we will simply write $\longrightarrow_G$ instead of $\xrightarrow{\preceq_{\mathtt{t}},\preceq}_G$.

What has been said above about the relationship between $\mathfrak{s}(b)$ and $\mathfrak{s}(a)$ if b $\mathfrak{s}$-reduces to a is of course also true for the reflexive-transitive closure of $\mathfrak{s}$-reduction. In particular, if $a \xrightarrow{\prec_{\mathtt{t}},\mathrm{r}_2*}_G b$, i. e. a regular $\mathfrak{s}$-reduces to b in several steps, then $\mathfrak{s}(b) = \mathfrak{s}(a)$. This trivial observation will play a crucial role later on.

The definition of $\mathfrak{s}$-reduction in the formalization closely follows Definition 4.1, but in addition also incorporates Notation 4.1:

```
definition sig-red-single :: "(τ ⇒ τ ⇒ bool) ⇒ (α ⇒ α ⇒ bool) ⇒
                              (τ ⇒₀ β) ⇒ (τ ⇒₀ β) ⇒ (τ ⇒₀ β) ⇒ α ⇒ bool"
  where "sig-red-single r1 r2 a b g t ⟷
         (poly g ≠ 0 ∧ coeff (poly a) (t * lp (poly g)) ≠ 0 ∧
           b = a - monom-mult ((coeff (poly a) (t * lp (poly g))) / lc (poly g)) t g ∧
           r1 (t ⊗ 𝔰 g) (𝔰 a) ∧ r2 (t * lp (poly g)) (lp (poly a)))"

definition sig-red :: "(τ ⇒ τ ⇒ bool) ⇒ (α ⇒ α ⇒ bool) ⇒
                       (τ ⇒₀ β) set ⇒ (τ ⇒₀ β) ⇒ (τ ⇒₀ β) ⇒ bool"
  where "sig-red r1 r2 G a b ⟷ (∃g∈G. ∃t. sig-red-single r1 r2 a b g t)"

definition is-sig-red :: "(τ ⇒ τ ⇒ bool) ⇒ (α ⇒ α ⇒ bool) ⇒
                          (τ ⇒₀ β) set ⇒ (τ ⇒₀ β) ⇒ bool"
  where "is-sig-red r1 r2 G a ⟷ (∃b. sig-red r1 r2 G a b)"
```

So, sig-red-single$(\mathrm{r}_1, \mathrm{r}_2, a, b, g, t)$ expresses that a $\mathfrak{s}$-reduces to b modulo the singleton $\{g\}$ using the given power-product t as the multiplier.[10] The two relations r_1 and r_2 have

[8] $\overline{G}$, of course, denotes the image of G under the homomorphism $\bar{\cdot}$.
[9] Obviously $\mathrm{lp}(\overline{a}) \prec t\,\mathrm{lp}(\overline{g})$ is not possible, since $t\,\mathrm{lp}(\overline{g}) \in \mathrm{supp}(\overline{a})$.
[10] monom-mult(c, t, a) is multiplication of a by the coefficient c and power-product t.

exactly the same meaning as in Notation 4.1, i. e. they specify whether the $\mathfrak{s}$-reduction is singular/regular/arbitrary and top/tail/arbitrary, respectively. $\mathsf{sig\text{-}red}(\mathrm{r}_1, \mathrm{r}_2, G, a, b)$ precisely corresponds to $a \xrightarrow{\mathrm{r}_1,\mathrm{r}_2}_G b$; the reflexive-transitive closure of $\mathfrak{s}$-reduction is thus given by $\mathsf{sig\text{-}red}(\mathrm{r}_1, \mathrm{r}_2, G)^{**}$, using Isabelle/HOL's built-in notation r^{**} for denoting the reflexive-transitive closre of an arbitrary binary relation r. is-sig-red, finally, is an auxiliary notion expressing $\mathfrak{s}$-reducibility.

Since traditional polynomial reduction is known to be Noetherian, and $\mathfrak{s}$-reduction in some sense 'refines' polynomial reduction, we can immediately infer that $\mathfrak{s}$-reduction is Noetherian, too:

Lemma 4.1. *For all $G \subseteq \mathfrak{R}^m$, $\xrightarrow{\mathrm{r}_1,\mathrm{r}_2}_G$ is Noetherian, that is, there are no infinite chains $a_1 \xrightarrow{\mathrm{r}_1,\mathrm{r}_2}_G a_2 \xrightarrow{\mathrm{r}_1,\mathrm{r}_2}_G \ldots$.*

This lemma can be translated easily into Isabelle/HOL, employing the built-in predicate wfP for expressing well-foundedness of the converse of $\mathfrak{s}$-reduction (denoted by $^{--}$):

```
lemma sig-red-wf-Rm:
  assumes "G ⊆ Rm"
  shows "wfP (sig-red r1 r2 G)⁻⁻"
```

Proving this lemma is a matter of only a couple of lines, thanks to the fact that Immler and Maletzky (2016) already proved Noetherianity of traditional polynomial reduction in Isabelle/HOL. The assumption of G being a subset of Rm is necessary because of the observations made in Section 3.2.

Before we define signature Gröbner bases, we introduce an auxiliary notion:

Definition 4.2. We say that a *$\mathfrak{s}$-reduces to zero* modulo G if, and only if, there exists b such that $a \xrightarrow{\prec_t,\preceq}{}^*_G b$ and $\overline{b} = 0$, i. e. b is a syzygy. Just as for $\mathfrak{s}$-reduction, we will also use the phrases *singular* and *regular* $\mathfrak{s}$-reduction to zero, if $a \xrightarrow{=,\prec}{}^*_G b$ or $a \xrightarrow{\prec_t,\prec}{}^*_G b$, respectively.

Note that even though we use the word 'zero' in Definition 4.2 it does not mean that b itself has to be 0, only that it must be a syzygy. This terminology is taken from Eder and Faugère (2017).

In the non-signature world, a Gröbner basis is a set $G \subseteq \mathfrak{R}$ such that every $p \in \langle G \rangle$ can be reduced to 0 modulo G. This definition can be translated readily into the signature-based setting:

Definition 4.3 (Signature Gröbner Basis). Let u be a term. A set $G \subseteq \mathfrak{R}^m$ is a *signature Gröbner basis* in u if, and only if, every $a \in \mathfrak{R}^m$ with $\mathfrak{s}(a) = u$ $\mathfrak{s}$-reduces to zero modulo G.

G is a signature Gröbner basis up to u if it is a signature Gröbner basis in all $v \prec_t u$. If G is a signature Gröbner basis in all terms, we simply call it a signature Gröbner basis.

Translating Definitions 4.2 and 4.3 into Isabelle/HOL is again immediate; the only real differences are some Rm conditions, as usual:

```
definition sig-red-zero :: "(τ ⇒ τ ⇒ bool) ⇒ (τ ⇒₀ β) set ⇒ (τ ⇒₀ β) ⇒ bool"
  where "sig-red-zero r1 G a ⟷ (∃b. (sig-red r1 (≼) G)** a b ∧ poly b = 0)"
```

```
definition is-sig-GB-in :: "(τ ⇒₀ β) set ⇒ τ ⇒ bool"
  where "is-sig-GB-in G u ⟷
                 (∀a. 𝔰 a = u ⟶ a ∈ Rm ⟶ sig-red-zero (≼ₜ) G a)"

definition is-sig-GB_upt :: "(τ ⇒₀ β) set ⇒ τ ⇒ bool"
  where "is-sig-GB-upt G u ⟷
       (G ⊆ Rm ∧ (∀v. v ≺ₜ u ⟶ snd v < length fs ⟶ is-sig-GB-in G v))"
```

Please note that sig-red-zero is only parameterized over the relation r_1 for signatures, but not over r_2 for leading power-products: there is no need to distinguish between top/tail/arbitrary $\mathfrak{s}$-reductions to zero.

The connection between signature Gröbner bases and ordinary non-signature Gröbner bases follows immediately from the definition of $\mathfrak{s}$-reduction:

Proposition 4.1. *Let $G \subseteq \mathfrak{R}^m$ be a signature Gröbner basis. Then $\overline{G}$ is a Gröbner basis of $\langle f_1, \ldots, f_m \rangle$.*

Signature-based Gröbner basis algorithms, like F_5, compute signature Gröbner bases.[11] Proposition 4.1 tells us that from a signature Gröbner basis one can easily obtain a Gröbner basis of the ideal $\langle f_1, \ldots, f_m \rangle$ under consideration by applying the module-homomorphism $\bar{\cdot}$ to all elements.

The formalization of Proposition 4.1 looks as follows, where is-Groebner-basis is defined in Immler and Maletzky (2016):

```
lemma is-sig-GB-is-Groebner-basis:
  assumes "G ⊆ Rm" and "∀u. is-sig-GB-in G u"
  shows "is-Groebner-basis (poly ` G)"
```

The next result about signature Gröbner bases will prove very useful later on, for instance in Lemma 4.3. Readers interested in its proof are referred to Lemma 3 in Roune and Stillman (2012).

Lemma 4.2. *Let $a, b \in \mathfrak{R}^m \backslash \{0\}$, let G be a signature Gröbner basis up to $\mathfrak{s}(a)$, and assume $\mathfrak{s}(a) = \mathfrak{s}(b)$ and $\mathrm{lc}(a) = \mathrm{lc}(b)$.*

1. *If both a and b are regular top $\mathfrak{s}$-irreducible modulo G, then $\mathrm{lp}(\overline{a}) = \mathrm{lp}(\overline{b})$ and $\mathrm{lc}(\overline{a}) = \mathrm{lc}(\overline{b})$.*
2. *If both a and b are regular $\mathfrak{s}$-irreducible modulo G, then $\overline{a} = \overline{b}$.*

In the formalization this lemma is split into two lemmas:

```
lemma sig-regular-top-reduced-lp-lc-unique:
  assumes "is-sig-GB-upt G (𝔰 a)" and "a ∈ Rm" and "b ∈ Rm"
    and "𝔰 a = 𝔰 b" and "lc a = lc b"
    and "¬ is-sig-red (≺ₜ) (=) G a" and "¬ is-sig-red (≺ₜ) (=) G b"
  shows "lp (poly a) = lp (poly b)" and "lc (poly a) = lc (poly b)"

lemma sig-regular-reduced-unique:
  assumes "is-sig-GB-upt G (𝔰 a)" and "a ∈ Rm" and "b ∈ Rm"
    and "𝔰 a = 𝔰 b" and "lc a = lc b"
    and "¬ is-sig-red (≺ₜ) (≼) G a" and "¬ is-sig-red (≺ₜ) (≼) G b"
  shows "poly a = poly b"
```

[11] Strictly speaking, they compute *rewrite bases*, which are a subclass of signature Gröbner bases; see Section 5.

We conclude this section by introducing the concept of a *syzygy signature* and proving an important lemma about it:

Definition 4.4 (Syzygy Signature). Let u be a term. u is called a *syzygy signature* if there exists $a \in \mathfrak{R}^m \backslash \{0\}$ with $\mathfrak{s}(a) = u$ and $\overline{a} = 0$.

Syzygy signatures play a key role for detecting useless zero-reductions when computing signature Gröbner bases. Namely, by virtue of Lemma 4.2, we obtain the following result whose importance will become clear in Section 5:

Lemma 4.3 (Syzygy Criterion)**.** *Let $a \in \mathfrak{R}^m$ and let G be a signature Gröbner basis up to $\mathfrak{s}(a)$. If $\mathfrak{s}(a)$ is a syzygy signature, then a regular $\mathfrak{s}$-reduces to zero modulo G.*

Moreover, if u is a syzygy signature and $u \mid v$,[12] then v is a syzygy signature, too.

By now it should hopefully be clear how Definition 4.4 and Lemma 4.3 translate into Isabelle/HOL:

```
definition is-syz-sig :: "'τ ⇒ bool"
  where "is-syz-sig u ⟷ (∃a∈Rm. a ≠ 0 ∧ 𝔰 a = u ∧ poly a = 0)"

lemma syzygy-crit:
  assumes "is-sig-GB-upt G (𝔰 a)" and "is-syz-sig G (𝔰 a)" and "a ∈ Rm"
  shows "sig-red-zero (≺t) G a"

lemma is-syz-sig-dvd:
  assumes "is-syz-sig u" and "u dvdt v"
  shows "is-syz-sig v"
```

5. Rewrite Bases and S-Pairs

Besides signature Gröbner bases, we need another class of sets $G \subseteq \mathfrak{R}^m$ of module elements, namely so-called *rewrite bases*. In order to define them, however, we first need to introduce the notion of a *rewrite order*. We are aware that this notion comes 'out of the blue' here, but think it is not in the scope of this exposition to give a detailed account on the motivation for and intuition behind rewrite orders. As usual, readers are referred to Eder and Faugère (2017) instead.

Definition 5.1 (Sig-Poly-Pair). A *sig-poly-pair* is a pair $(u, p) \in \mathfrak{T} \times \mathfrak{R}$ such that there exists $a \in \mathfrak{R}^m \backslash \{0\}$ with $\mathfrak{s}(a) = u$ and $\overline{a} = p$.

Our definition of rewrite orders is slightly more technical and that given in the literature. The reason for this deviation is that there, rewrite orders are defined for module elements rather than sig-poly-pairs. We, however, found it more reasonable to define rewrite orders on sig-poly-pairs, because in any case the only information concrete rewrite orders may take into account for deciding which of the two arguments is greater are the signatures and the polynomial parts of the arguments.

[12] $u \mid v$, for two terms u and v, means that there exists $t \in [X]$ with $v = t\,u$.

Definition 5.2 (Rewrite Order). A binary relation $\trianglelefteq$ on sig-poly-pairs is called a *rewrite order* if, and only if, it is a reflexive, transitive and linear relation, additionally satisfying

1. $(u, p) \trianglelefteq (v, q) \wedge (v, q) \trianglelefteq (u, p) \Longrightarrow u = v$ for all sig-poly-pairs (u, p) and (v, q), and
2. $a \in G \backslash \{0\} \wedge b \in G \backslash \{0\} \wedge \mathfrak{s}(a) \mid \mathfrak{s}(b) \Longrightarrow (\mathfrak{s}(a), \overline{a}) \trianglelefteq (\mathfrak{s}(b), \overline{b})$ for all a, b, G such that G is a signature Gröbner basis up to $\mathfrak{s}(b)$ and b is regular top $\mathfrak{s}$-irreducible modulo G.

The last condition essentially expresses that $\trianglelefteq$ shall refine the divisibility relation on signatures—but only under some technical assumptions which are necessary for proving that $\trianglelefteq_{\mathsf{rat}}$ (Definition 5.3) is indeed a rewrite order.

To get some intuition about rewrite orders, we present the two 'standard' rewrite orders appearing in the literature right away:

Definition 5.3 ($\trianglelefteq_{\mathsf{rat}}$, $\trianglelefteq_{\mathsf{add}}$). The relation $\trianglelefteq_{\mathsf{rat}}$ is defined as

$$(u, p) \trianglelefteq_{\mathsf{rat}} (v, q) \;:\Leftrightarrow\; \mathrm{lp}(q)\, u \prec_{\mathsf{t}} \mathrm{lp}(p)\, v \vee (\mathrm{lp}(q)\, u = \mathrm{lp}(p)\, v \wedge u \preceq_{\mathsf{t}} v).$$

The relation $\trianglelefteq_{\mathsf{add}}$ is defined as

$$(u, p) \trianglelefteq_{\mathsf{add}} (v, q) \;:\Leftrightarrow\; u \preceq_{\mathsf{t}} v.$$

As explained in Remark 7.3 in Eder and Faugère (2017), the suffix 'rat' of $\trianglelefteq_{\mathsf{rat}}$ originates from an alternative presentation of this relation, in which the *ratios* $\frac{u}{\mathrm{lp}(p)}$ and $\frac{v}{\mathrm{lp}(q)}$ are compared.

Above we claimed that $\trianglelefteq_{\mathsf{rat}}$ and $\trianglelefteq_{\mathsf{add}}$ are rewrite orders. The proof for $\trianglelefteq_{\mathsf{add}}$ is fairly straight-forward, but the proof of the last requirement of rewrite orders is a bit more involved for $\trianglelefteq_{\mathsf{rat}}$; one essentially has to make use of Lemma 4.2 again.

The definition of rewrite orders in the formalization closely resembles Definition 5.2; only note that $\mathsf{spp\text{-}of}(a)$ is a mere abbreviation for $(\mathfrak{s}(a), \mathsf{poly}(a))$:

```
definition is-rewrite-ord :: "((τ × (α ⇒₀ β)) ⇒ (τ × (α ⇒₀ β)) ⇒ bool) ⇒ bool"
  where "is-rewrite-ord ord ⟷
                   (reflp ord ∧ transp ord ∧ (∀a b. ord a b ∨ ord b a) ∧
                    (∀a b. ord a b ⟶ ord b a ⟶ fst a = fst b) ∧
                    (∀G a b. is-sig-GB-upt G (𝔰 b) ⟶ a ∈ G ⟶ b ∈ G ⟶
                      a ≠ 0 ⟶ b ≠ 0 ⟶ 𝔰 a dvdₜ 𝔰 b ⟶
                      ¬ is-sig-red (≺ₜ) (=) G b ⟶ ord (spp-of a) (spp-of b)))"
```

Since there is nothing special about the formal definitions of $\trianglelefteq_{\mathsf{rat}}$ and $\trianglelefteq_{\mathsf{add}}$ compared to the informal ones, we omit them here.

Just as we have implicitly fixed $\preceq$ and $\preceq_{\mathsf{t}}$, let us now also fix an arbitrary rewrite order $\trianglelefteq$. The last prerequisite we need before we can define rewrite bases are *canonical rewriters*:

Definition 5.4 (Canonical Rewriter). Let $G \subseteq \mathfrak{R}^m$, $a \in \mathfrak{R}^m$ and $u \in \mathfrak{T}$. a is called a *canonical rewriter* in signature u w. r. t. G if, and only if, $a \in G \backslash \{0\}$, $\mathfrak{s}(a) \mid u$, and a is maximal w. r. t. $\trianglelefteq$ with these properties.[13]

[13] By abuse of notation we also compare module elements in $\mathfrak{R}^m$ w. r. t. $\trianglelefteq$, in the sense that $a \trianglelefteq b \Leftrightarrow (\mathfrak{s}(a), \overline{a}) \trianglelefteq (\mathfrak{s}(b), \overline{b})$.

Definition 5.5 (Rewrite Basis). Let $G \subseteq \mathfrak{R}^m$ and $u \in \mathfrak{T}$. G is said to be a *rewrite basis* in u if, and only if, u is a syzygy signature or there exists a canonical rewriter g in signature u w. r. t. G such that $\frac{u}{\mathfrak{s}(g)} g$ is regular top $\mathfrak{s}$-irreducible modulo G.[14]

G is a rewrite basis up to u if it is a rewrite basis in all $v \prec_{\mathfrak{t}} u$. If G is a rewrite basis in all terms, we simply call it a rewrite basis.

As for the formal definitions of canonical rewriters and rewrite bases in Isabelle/HOL, there should be nothing surprising about them; note in particular the parallels between the definitions of is-sig-GB-upt and is-RB-upt:

```
definition is-canon-rewriter :: "(τ ⇒₀ β) set ⇒ τ ⇒ (τ ⇒₀ β) ⇒ bool"
  where "is-canon-rewriter G u a ⟷
                      (a ∈ G ∧ a ≠ 0 ∧ 𝔰 a dvdₜ u ∧
                       (∀g∈G. g ≠ 0 ⟶ 𝔰 g dvdₜ u ⟶ spp-of g ◁ spp-of a)))"

definition is-RB-in :: "(τ ⇒₀ β) set ⇒ τ ⇒ bool"
  where "is-RB-in G u ⟷
                (is-syz-sig u ∨
                 (∃g. is-canon-rewriter G u g ∧
                      ¬ is-sig-red (≺ₜ) (=) G (monom-mult 1 (u / 𝔰 g) g)))"

definition is-RB-upt :: "(τ ⇒₀ β) set ⇒ τ ⇒ bool"
  where "is-RB-upt G u ⟷
            (G ⊆ Rm ∧ (∀v. v ≺ₜ u ⟶ snd v < length fs ⟶ is-RB-in G v))"
```

So far, so good: we know what rewrite bases are, and now it is time to establish the connection between rewrite bases and signature Gröbner bases, and hence to traditional Gröbner bases by virtue of Proposition 4.1. For an informal proof of the following proposition, see Lemma 8 in Eder and Roune (2013):

Proposition 5.1. *If G is a rewrite basis up to u, it is also a signature Gröbner basis up to u.*

We omit the (obvious) translation of this proposition into Isabelle/HOL. Summarizing, Propositions 4.1 and 5.1 justify computing a rewrite basis in order to find a (traditional) Gröbner basis of $\langle f_1, \ldots, f_m \rangle$. Therefore, we now need a means for actually *computing* rewrite bases—and it turns out that the key to an effective algorithm lies in a concept well-known from traditional Gröbner bases theory:

Definition 5.6 (S-Pair). Let $a, b \in \mathfrak{R}^m$, and let $t = \operatorname{lcm}(\operatorname{lp}(\overline{a}), \operatorname{lp}(\overline{b}))$. Then the *S-pair* of a and b, written $\operatorname{spair}(a, b)$, is defined as

$$\operatorname{spair}(a, b) \;:=\; \frac{t}{\operatorname{lc}(\overline{a})\operatorname{lp}(\overline{a})} a - \frac{t}{\operatorname{lc}(\overline{b})\operatorname{lp}(\overline{b})} b.$$

Furthermore, a and b are said to give rise to a *regular* S-pair if, and only if, $\overline{a}, \overline{b} \neq 0$ and $\frac{t}{\operatorname{lp}(\overline{a})}\mathfrak{s}(a) \neq \frac{t}{\operatorname{lp}(\overline{b})}\mathfrak{s}(b)$; otherwise they give rise to a *singular* S-pair.

[14] For two terms u, v with $u \mid v$, $\frac{u}{v}$ denotes the (unique) $t \in [X]$ with $v = t\,u$.

So, S-pairs correspond precisely to S-polynomials, but 'lifted' from $\mathfrak{R}$ to $\mathfrak{R}^m$: indeed, $\text{spair}(a, b) \in \mathfrak{R}^m$, and it is easy to see that $\overline{\text{spair}(a, b)} = \text{spoly}(\overline{a}, \overline{b})$, where $\text{spoly}(p, q)$ returns the usual S-polynomial of p and q.

The distinction between singular and regular S-pairs is important, because in Theorem 5.1 below we will show that only regular S-pairs are of interest. If a and b give rise to a regular S-pair, we have $\mathfrak{s}(\text{spair}(a, b)) = \max(\frac{t}{\text{lp}(\overline{a})}\mathfrak{s}(a), \frac{t}{\text{lp}(\overline{b})}\mathfrak{s}(b))$, where t is as in Definition 5.6.

The definitions of S-pairs and regular S-pairs in the formalization look as follows:

```
definition spair :: "('τ ⇒₀ 'β) ⇒ ('τ ⇒₀ 'β) ⇒ ('τ ⇒₀ 'β)"
  where "spair a b = (let t1 = lp (poly a); t2 = lp (poly b); t = lcm t1 t2 in
                        (monom-mult (1 / lc (poly a)) (t / t1) a) -
                        (monom-mult (1 / lc (poly b)) (t / t2) b))"

definition is-regular-spair :: "('τ ⇒₀ 'β) ⇒ ('τ ⇒₀ 'β) ⇒ bool"
  where "is-regular-spair a b ⟷
                   (poly a ≠ 0 ∧ poly b ≠ 0 ∧
                    (let t1 = lp (poly a); t2 = lp (poly b); t = lcm t1 t2 in
                       (t / t1) ⊗ 𝔰 a ≠ (t / t2) ⊗ 𝔰 b))"
```

Now we are ready to state the central theorem in this section, which links rewrite bases to (regular) S-pairs just as Buchberger's theorem links Gröbner bases to S-polynomials:

Theorem 5.1. *Let $G \subseteq \mathfrak{R}^m$ be finite and $u \in \mathfrak{T}$, assume that no two elements of G have the same signatures, and moreover assume that G is a rewrite basis in all $\mathfrak{s}(a) \prec_{\mathfrak{t}} u$, where a is either a regular S-pair of elements of G or $a = \mathbf{e}_i$ $(1 \leq i \leq m)$. Then G is a rewrite basis up to u.*

For a proof of this theorem see Lemma 10 in Eder and Roune (2013). Theorem 5.1 gives us some idea how to decide whether a given finite set G is a rewrite basis up to u: it suffices to check the *finitely many* signatures of regular S-pairs and canonical basis vectors. Note, however, that there is still an issue related to syzygy signatures: the definition of rewrite bases involves syzygy signatures, and deciding whether a given u is a syzygy signature is a difficult problem (actually, as difficult as computing a Gröbner basis of the module of syzygies). Luckily, Theorem 5.1 does not only suggest a method for (semi-)deciding whether a given set is a rewrite basis, but it also gives rise to an algorithm for *computing* rewrite bases which does not suffer from the problem with syzygy signatures just outlined. This algorithm will be presented in Section 6, but before, we still have to show the statement of Theorem 5.1 in the formalization:

```
lemma is-RB-upt-finite:
  assumes "G ⊆ Rm" and "inj-on 𝔰 G" and "finite G"
    and "∀g1∈G. ∀g2∈G. is-regular-spair g1 g2 ⟶ 𝔰 (spair g1 g2) ≺ₜ u ⟶
                        is-RB-in G (𝔰 (spair g1 g2))"
    and "∀i. i < length fs ⟶ (1, i) ≺ₜ u ⟶ is-RB-in G (1, i)"
  shows "is-RB-upt G u"
```

The second assumption of *is-RB-upt-finite* merely expresses that the function $\mathfrak{s}$ is injective on G, that is, no two elements of G have the same signatures.

6. Algorithms

As claimed above, Theorem 5.1 gives rise to an algorithm for computing rewrite bases, and in fact that algorithm bears close resemblance to Buchberger's algorithm for computing Gröbner bases: it is a critical-pair/completion algorithm that successively iterates through all S-pairs, applies a criterion for testing whether the S-pair under consideration must be reduced, $\mathfrak{s}$-reduces it to some normal form if necessary, and adds the result to the so-far computed basis unless it be zero. Algorithm 1 summarizes the method just sketched in an imperative programming style; it is a slight variation of Algorithm 3 in Eder and Faugère (2017). Several remarks on Algorithm 1 are in place:

Algorithm 1 An algorithm for computing rewrite bases.

Input: sequence $(f_1, \ldots, f_m)$ of polynomials in $\mathfrak{R}$, admissible order $\preceq$ on $[X]$, compatible extension $\preceq_{\mathtt{t}}$ on $\mathfrak{T}$, rewrite order $\trianglelefteq$

Output: rewrite basis G

1: **function** RB$((f_1, \ldots, f_m), \preceq, \preceq_{\mathtt{t}}, \trianglelefteq)$
2: $\quad G \leftarrow \emptyset$
3: $\quad S \leftarrow \{\mathfrak{s}(f_j\,\mathbf{e}_i - f_i\,\mathbf{e}_j) \mid 1 \leq i < j \leq m\}$
4: $\quad P \leftarrow \{\mathbf{e}_i \mid 1 \leq i \leq m\}$
5: $\quad$ **while** $P \neq \emptyset$ **do**
6: $\quad\quad a \leftarrow$ some element of P with $\preceq_{\mathtt{t}}$-minimal signature
7: $\quad\quad P \leftarrow P \backslash \{a\}$
8: $\quad\quad$ **if** $a = \mathbf{e}_i$ for some i **then**
9: $\quad\quad\quad S \leftarrow S \cup \{\mathfrak{s}(f_i\,g - \overline{g}\,\mathbf{e}_i) \mid g \in G\}$
10: $\quad\quad$ **if** $\neg$sigCrit$(\trianglelefteq, G, S, a)$ **then**
11: $\quad\quad\quad b \leftarrow$ result of regular $\mathfrak{s}$-reducing a modulo G
12: $\quad\quad\quad$ **if** $\overline{b} = 0$ **then**
13: $\quad\quad\quad\quad S \leftarrow S \cup \{\mathfrak{s}(b)\}$
14: $\quad\quad\quad$ **else**
15: $\quad\quad\quad\quad G \leftarrow G \cup \{b\}$
16: $\quad\quad\quad\quad P \leftarrow P \cup \{\text{spair}(g, b) \mid g \in G,\ \text{spair}(g, b) \text{ is regular}\}$
17: $\quad$ **return** G

- G is the so-far computed basis and P is the set of elements that still have to be considered. P does not only contain regular S-pairs, but also the m canonical basis vectors corresponding to the input-sequence $(f_1, \ldots, f_m)$; this little trick justifies initializing G by the empty set.
- S is the set of signatures of some known syzygies. It is initialized by the signatures of the *Koszul syzygies* of the input sequence, and successively enlarged in Lines 9 and 13. These syzygy-signatures are used to apply the syzygy criterion (Lemma 4.3) in function sigCrit, see Algorithm 2 below.
- It is important to note that in Line 6 of Algorithm 1, an element a with minimal signature is taken from P. This is crucial for the correctness of the algorithm, since a different choice could lead to wrong results.

- Also note that $\mathfrak{s}(b) = \mathfrak{s}(a)$, since b is the result of *regular* $\mathfrak{s}$-reducing a, and regular $\mathfrak{s}$-reductions do not change signatures. This, together with the particular choice of a, implies that G is computed by increasing signatures, i. e., the signatures of the elements b added to G in Line 15 are increasing.

Ignoring the sigCrit-test in Line 10 of Algorithm 1 for the moment, the algorithm is partially correct. This follows from the fact that either $\overline{b} = 0$, in which case $\mathfrak{s}(b) = \mathfrak{s}(a)$ is a syzygy signature, or b is added to G, in which case it becomes the canonical rewriter in $\mathfrak{s}(b) = \mathfrak{s}(a)$ (this follows from the definition of rewrite orders) and is by construction regular (top) $\mathfrak{s}$-irreducible. Therefore, in either case the (potentially enlarged) set G is a rewrite basis in $\mathfrak{s}(a)$ by Definition 5.5, and upon termination of the algorithm,[15] it is a rewrite basis in *all* terms u thanks to Theorem 5.1.

The auxiliary function sigCrit, which implements in Algorithm 2, tests whether an S-pair $\operatorname{spair}(a, b)$ has to be $\mathfrak{s}$-reduced in Algorithm 1. In a nutshell, it applies Lemma 4.3, the syzygy criterion, and moreover checks whether the constituents of the S-pair are canonical rewriters in certain terms u_a and u_b; if not, the S-pair does not have to be reduced, because either the canonical rewriters in these respective terms have been treated already, or will still be treated later on, and in either case there is nothing to be done for $\operatorname{spair}(a, b)$. There is one subtle point, though: Knowing that $\operatorname{spair}(a, b)$ is regular, one of u_a or u_b is strictly greater than the other by definition, and $\mathfrak{s}(\operatorname{spair}(a, b)) = \max(u_a, u_b)$. W. l. o. g. assume $u_b \prec_{\mathsf{t}} u_a$. So, by what has been said above, it should be clear that sigCrit is allowed to do the checks on $u_a = \mathfrak{s}(\operatorname{spair}(a, b))$ in Line 5 of Algorithm 2, but it is perhaps not clear why the same checks may also be performed on the *smaller* term u_b that does not contribute to $\mathfrak{s}(\operatorname{spair}(a, b))$ at all. Indeed, answering this question is slightly intricate, and we confine ourselves here to pointing the interested reader to Lemma 12 in Eder and Roune (2013) for an explanation.

Algorithm 2 An algorithm for testing whether S-pairs must be regular $\mathfrak{s}$-reduced.

Input: rewrite order $\trianglelefteq$, $G \subseteq \mathfrak{R}^m$, $S \subseteq \mathfrak{T}$, regular $\operatorname{spair}(a, b)$ with $a, b \in G$
Output: 'False' if $\operatorname{spair}(a, b)$ has to be regular $\mathfrak{s}$-reduced in Algorithm 1

1: **function** sigCrit($\trianglelefteq, G, S, \operatorname{spair}(a, b)$)
2: $t \leftarrow \operatorname{lcm}(\operatorname{lp}(\overline{a}), \operatorname{lp}(\overline{b}))$
3: $u_a \leftarrow \frac{t}{\operatorname{lp}(\overline{a})}\mathfrak{s}(a)$
4: $u_b \leftarrow \frac{t}{\operatorname{lp}(\overline{b})}\mathfrak{s}(b)$
5: **if** $(\exists s \in S.\ s \mid u_a) \vee (a$ is not canonical rewriter in u_a w. r. t. $G)$ **then**
6: **return** True
7: **if** $(\exists s \in S.\ s \mid u_b) \vee (b$ is not canonical rewriter in u_b w. r. t. $G)$ **then**
8: **return** True
9: **return** False

We hope we could convince the reader about the partial correctness of Algorithms 1 and 2 now; if not, a more thorough account on the whole subject can, as usual, be found in Eder and Faugère (2017). However, the algorithm is not only partially correct, but also terminates for every input; this claim will be investigated in Section 6.1. We summarize the result in a theorem:

[15] Termination will be addressed in Section 6.1

Theorem 6.1 (Correctness of Algorithm 1)**.** *For every input, Algorithm 1 terminates and returns a rewrite basis G w. r. t. $(f_1, \ldots, f_m)$, $\preceq$, $\preceq_{\mathfrak{t}}$ and $\trianglelefteq$. Furthermore, $\langle \overline{G} \rangle = \langle f_1, \ldots, f_m \rangle$.*

Remark 6.1. Algorithm 2 corresponds to Algorithm 4 in Eder and Faugère (2017), which, however, is presented in a slightly different way. Namely, the two disjuncts in Lines 5 and 7 of Algorithm 2 are combined into one single 'rewritability' check in the cited article. This makes the formulation of the algorithm a bit more elegant.

Also, one has to take into account that the last argument of function sigCrit could be a canonical basis vector $\mathbf{e}_i$ rather than an S-pair. In that case, only the syzygy criterion is applied (i. e., $\exists s \in S.\ s \mid \mathbf{e}_i$).

Let us now turn to the formalization of RB in Isabelle/HOL. There, it is natural to implement functions as *functional* programs instead of imperative ones, so we define the tail-recursive function rb-aux for computing rewrite bases as follows:

```
function rb-aux ::
      "(((τ ⇒₀ β) list × τ list × (((τ ⇒₀ β) × (τ ⇒₀ β)) + nat) list) × nat) ⇒
       (((τ ⇒₀ β) list × τ list × (((τ ⇒₀ β) × (τ ⇒₀ β)) + nat) list) × nat)"
  where
    "rb-aux ((gs, ss, []), z) = ((gs, ss, []), z)" |
    "rb-aux ((gs, ss, a # ps'), z) =
      (let ss' = new-syz-sigs ss gs a in
        if sig-crit gs ss' a then
          rb-aux ((gs, ss', ps'), z)
        else
          let b = sig-trd gs (poly-of-pair a) in
            if poly b = 0 then
              rb-aux ((gs, (𝔰 b) # ss', ps'), Suc z)
            else
              rb-aux ((b # gs, ss', add-spairs ps' gs b), z))"
```

The function takes one argument, which in turn is a tuple consisting of four entries: a list gs corresponding to the set G in Algorithm 1, a list ss corresponding to S, a list ps corresponding to P, and a natural number z counting the total number of zero-reductions. z is mere technicality only needed in Section 7.1, and may thus be ignored for the moment. The function not only returns gs, but also the other arguments, to facilitate formal reasoning about it—but of course only gs is interesting from our perspective. Please note that the list fs and the various relations $\preceq$ etc. are still implicitly fixed in the theory context and therefore do not have to be passed as arguments to rb-aux explicitly.

The first part of the definition corresponds to the base case, where the list ps is empty. The second part corresponds to the case where ps contains at least one element, and can hence be decomposed into its head a and tail ps'. Since we ensure that the list is always kept sorted by increasing signatures, a is known to be an element with minimal signature, just as required in Line 6 of Algorithm 1. Then, ss is enlarged by new syzygy-signatures in the auxiliary function new-syz-sigs, and the result is stored in ss'; this corresponds precisely to Lines 8 and 9 of Algorithm 1. Afterward, the auxiliary function sig-crit is applied to gs, ss' and a to check whether a has to be $\mathfrak{s}$-reduced or not. sig-crit is the formalization of function sigCrit, and since there is nothing special about its definition,

we omit it here. Anyway, if sig-crit returns True, nothing needs to be done and rb-aux is called recursively on the remaining list ps'. Otherwise, a is regular $\mathfrak{s}$-reduced to b (taken care of by function sig-trd), and depending on whether b is a syzygy or not its signature is added to ss' or it is added to gs, and new S-pairs are added to ps' by function add-spairs. So, in short, rb-aux corresponds exactly to Lines 5–17 of Algorithm 1. The remaining lines, corresponding to the initialization of G, S and P, are covered by the way how the arguments of the initial call of rb-aux are constructed, as will be seen below.

Before, please note that the element-type of ps is a *sum type*, i. e. the disjoint union of two types: once the type of pairs of module elements, $(\tau \Rightarrow_0 \beta) \times (\tau \Rightarrow_0 \beta)$, and once the type nat of natural numbers. This owes to the fact that ps may both contain S-pairs and canonical basis vectors: S-pairs are represented by the two elements they originate from (because these elements themselves are needed in sig-crit), and canonical basis vectors are compactly represented by their component, which is of course a natural number. Function poly-of-pair converts an object of this sum type into an actual module element of type $\tau \Rightarrow_0 \beta$, by either constructing an S-pair or returning a 'full' basis vector.

The initial argument of rb-aux corresponds to the initial values of G, S and P:

- gs is the empty list,
- ss is Koszul-syz-sigs(fs), which returns the signatures of the Koszul syzygies of fs, and
- ps is the list map(Inr, $[0.. < \mathsf{length}(fs)]$), representing the canonical basis vectors in the sum type mentioned above.

So, we can finally define function rb as follows:

```
definition rb :: "(τ ⇒₀ β) list × nat"
  where "rb = (let ((gs, _, _), z) =
                      rb-aux (([], Koszul-syz-sigs fs, map Inr [0..<length fs]), 0)
                in (gs, z))"
```

As can be seen, rb does not take any explicit arguments in the above definition, but it is implicitly parameterized over the constants fixed in the theory context (fs, $\preceq$, $\preceq_{\mathfrak{t}}$ and $\trianglelefteq$).

In order to formally prove the correctness of rb-aux, and hence rb, we define an invariant rb-aux-inv of function rb-aux that holds for the initial argument, is preserved in every recursive call, and is strong enough to infer the desired properties of rb from it. Since the precise definition of the invariant is fairly lengthy, we only informally summarize its key characteristics here. rb-aux-inv(gs, ss, ps) holds if

- the signatures of the elements of gs are strictly decreasing (note that new elements with larger signatures are added up front to gs),
- every element in gs stems from regular $\mathfrak{s}$-reducing an S-pair of elements coming later in gs (i. e. earlier during execution of the function), or from regular $\mathfrak{s}$-reducing a canonical basis vector,
- every element in gs is regular $\mathfrak{s}$-irreducible modulo the elements coming after it in gs,

- every element of gs belongs to the set Rm,
- gs does not contain syzygies,
- for every g in gs, the elements coming after it in gs constitute a rewrite basis up to $\mathfrak{s}(g)$,
- every element in ss is a syzygy signature,
- ps is sorted by increasing signatures,
- no element in ps has a signature which is strictly smaller than the signature of any element in gs, and
- gs is a rewrite basis in all $\mathbf{e}_i$ which do not appear in ps any more, and similar for S-pairs.

The first three items are only needed for proving termination of rb-aux, see Section 6.1. This list is not exhaustive; it is only meant to give an impression of how challenging it is to prove correctness of rb-aux and rb in a formal environment. In absolute figures, the whole proof, distributed across several lemmas, takes roughly 1800 lines of Isabelle code—not counting the proofs of the necessary theoretical results shown in previous sections, like *is-RB-upt-finite*! Interestingly, the claim that the invariant is preserved in the third recursive call of rb-aux turns out have the most difficult proof:

```
lemma rb-aux-inv-preserved-3:
  fixes gs ss a ps
  defines "ss' ≡ new-syz-sigs ss gs a"
  defines "b ≡ sig-trd gs (poly-of-pair a)"
  assumes "rb-aux-inv (gs, ss, a # ps)" and "¬ sig-crit gs ss' a" and "poly b ≠ 0"
  shows "rb-aux-inv (b # gs, ss', add-spairs ps gs b)"
```

After having proved that rb-aux-inv holds for the initial argument of rb-aux and is preserved in each of the three recursive calls, and that rb-aux terminates (see Section 6.1), we can infer the following two key properties of rb which correspond to Theorem 6.1:

```
theorem rb-is-RB-upt: "is-RB-upt (set (fst rb)) u"

theorem ideal-rb-aux: "ideal (poly ` set (fst rb)) = ideal (set fs)"
```

Remark 6.2. Algorithm 1 and function rb could easily be adapted to not only compute a rewrite basis, and hence Gröbner basis of the ideal $\langle f_1, \ldots, f_m \rangle$, but also a Gröbner basis of the module of syzygies of $(f_1, \ldots, f_m)$. We do not consider this in the formalization, though.

6.1. Termination

Termination of the original F_5 algorithm has been an open problem for a long time, until it was eventually settled by Galkin (2012). Later, Pan et al. (2012) proved termination of a more general signature-based algorithm, which happens to be equivalent to Algorithm 1. The proof we modeled our formal Isabelle-proof after can be found in Eder and Roune (2013) (Theorem 20). Here, we present the key ideas of the proof, referring the interested reader to the cited article for more information about it.

Assume $(g_1, g_2, g_3, \ldots)$ is the sequence of elements added to G by Algorithm 1, in that order. We want to show that this sequence is finite. First, introduce the following relation $\sim$ on $\mathfrak{R}^m$: $a \sim b \;:\Leftrightarrow\; \mathrm{lp}(\overline{b})\mathfrak{s}(a) = \mathrm{lp}(\overline{a})\mathfrak{s}(b)$. $\sim$ is an equivalence relation, and therefore allows one to partition the sequence into subsets of equivalent (w. r. t. $\sim$) elements. Next, one can prove that only finitely many of these subset are non-empty, using Noetherianity of $\mathfrak{R}^m$ and further properties of the sequence that follow from its being constructed by Algorithm 1 (e. g., no element is regular $\mathfrak{s}$-reducible by the others). Finally, one can prove by induction on the finitely many non-empty sets R that each of them is finite, because every element of R corresponds to an S-pair of elements in 'previous' sets (which are finite by the induction hypothesis). This concludes the proof.

Remark 6.3. Readers not so familiar with signature-based algorithms might wonder why the well-known termination proof of Buchberger's algorithm does not work for signature-based algorithms. The reason is simple: a new element b added to the basis is only regular $\mathfrak{s}$-irreducible, which unfortunately does not imply that $\overline{b}$ is irreducible in the traditional sense of polynomial reduction. In particular, $\mathrm{lp}(\overline{b})$ might even be divisible by $\mathrm{lp}(\overline{g})$ for some g in the current basis—something which cannot happen in Buchberger's algorithm, which in turn is what the termination proof of Buchberger's algorithm mainly rests upon.

In the formalization, the theorem needed for establishing termination of function rb-aux is as follows:

```
lemma rb-termination:
  fixes seq :: "nat ⇒ (τ ⇒₀ β)"
  assumes "∀i j. i < j ⟶ 𝔰 (seq i) ≺ₜ 𝔰 (seq j)"
    and "∀i. (∃j<length fs. 𝔰 (seq i) = (0, j) ∧ lp (poly (seq i)) ≼ lp (fs ! j)) ∨
             (∃j k. is-regular-spair (seq j) (seq k) ∧
                    poly (spair (seq j) (seq k)) ≠ 0 ∧
                    𝔰 (seq i) = 𝔰 (spair (seq j) (seq k)) ∧
                    lp (poly (seq i)) ≼ lp (poly (spair (seq j) (seq k))))"
    and "∀i. ¬ is-sig-red (≺ₜ) (≼) (seq ` {0..<i}) (seq i)"
    and "range seq ⊆ Rm" and "0 ∉ poly ` range seq"
    and "∀i. is-sig-GB-upt (seq ` {0..<i}) (𝔰 (seq i))"
  shows False
```

So, we assume that there exists an infinite sequence *seq* with the listed properties and derive a contradiction; hence, any such sequence must be finite. *seq* is modeled as a function from the natural numbers to module elements of type $\tau \Rightarrow_0 \beta$, which means that the i-th element of *seq* is simply $seq(i)$ and the set of all elements of *seq* is range(seq). A close inspection of the presumed properties of *seq* reveals that they essentially correspond to the first six properties of *gs* in the above list characterizing rb-aux-inv. The only real difference is that the order of the elements in *seq* corresponds to the order in which they are generated by function rb-aux, which is the *reversed* order compared to *gs*. This explains why, for instance, the signatures in *seq* must be strictly increasing, whereas in *gs* they must be strictly decreasing.

From *rb-termination* we can conclude that function rb-aux terminates for all arguments satisfying the invariant rb-aux-inv, which in particular includes the initial argument specified by function rb. This finishes the proof of total correctness of rb.

7. Optimality Results

7.1. No Zero-Reductions

The original goal of signature-based algorithms is to detect and avoid as many useless zero-reductions as possible, and thus speed up the computation of Gröbner bases. Practical experience shows that this goal is indeed achieved (see Section 8), and theory even tells us that in some situations zero-reductions can be avoided altogether:

Theorem 7.1. *Let $(f_1, \ldots, f_m)$ be a regular sequence and assume $\preceq_{\mathsf{t}} = \preceq_{\mathsf{pot}}$, i. e. $\preceq_{\mathsf{t}}$ is a POT-extension of $\preceq$. Then Algorithm 1 does not $\mathfrak{s}$-reduce any element to zero, meaning that the test in Line 12 of that algorithm always yields 'False'.*

The proof of this celebrated result, which is presented as Corollary 7.1 in Eder and Faugère (2017),[16] is actually not very difficult. It proceeds along the following lines: Using $\preceq_{\mathsf{pot}}$, the rewrite basis is computed incrementally, i. e. first for (f_1), then for (f_1, f_2), and so on. The sequence $(f_1, \ldots, f_m)$ being regular implies that the only syzygies a satisfying $\mathfrak{s}(a) = t\,\mathbf{e}_i$, for $1 \leq i \leq m$ and $t \in [X]$, are in the module of principal syzygies of $(f_1, \ldots, f_i)$—a generating set of which is added to S in Line 9. However, every zero-reduction corresponds to precisely such a syzygy, and therefore is detected beforehand by the syzygy criterion implemented in function sigCrit.

The formalization of Theorem 7.1 in Isabelle/HOL begins with the definition of regular sequences:

```
definition is-regular-sequence :: "(α ⇒₀ β) list ⇒ bool"
  where "is-regular-sequence fs ⟷
                    (∀j<length fs. ∀q. q * fs ! j ∈ ideal (set (take j fs)) ⟶
                                    q ∈ ideal (set (take j fs)))"
```

As can be seen, is-regular-sequence is a predicate on lists of polynomials. The definition avoids any reference to quotient rings by unfolding the definition of zero-divisors in such rings. take(j, fs) returns the list of the first j elements of fs; fs ! j is of course *not* the last element of take(j, fs), but comes one element afterward.

Proving that there are no zero-reductions in function rb obviously boils down to proving that the second case in the second part in the definition of rb-aux cannot occur. This means that whenever sig-crit fails to hold for some a, the result of regularly $\mathfrak{s}$-reducing a cannot be a syzygy:

```
lemma rb-aux-inv2-no-zero-red:
  assumes "is-regular-sequence fs" and "is-pot-ord"
    and "rb-aux-inv2 (gs, ss, a # ps)" and "¬ sig-crit gs (new-syz-sigs ss gs a) a"
  shows "poly (sig-trd gs (poly-of-pair a)) ≠ 0"
```

Here, is-pot-ord expresses the fact that the implicitly fixed order $\preceq_{\mathsf{t}}$ is a POT-extension of $\preceq$. rb-aux-inv2 is a strengthened version of rb-aux-inv, which can also be proved to be an invariant of rb-aux if fs is a regular sequence and is-pot-ord holds. It additionally requires ss to contain all necessary syzygy-signatures, something which is not needed for proving correctness of rb-aux and hence is not encoded in rb-aux-inv.

[16] Eder and Faugère (2017) need the additional assumption that $\trianglelefteq$ be either $\trianglelefteq_{\mathsf{rat}}$ or $\trianglelefteq_{\mathsf{add}}$. We do not need this assumption because of our slightly different implementation of function sigCrit.

As a consequence of *rb-aux-inv2-no-zero-red* and the fact that rb-aux-inv2 holds for the initial argument of rb-aux as specified in rb, we can infer that indeed no zero-reductions take place. This result is formulated using the second return value, z, of rb, which counts the total number of zero-reductions:

```
corollary rb-aux-no-zero-red:
  assumes "is-regular-sequence fs" and "is-pot-ord"
  shows "snd rb = 0"
```

7.2. Minimal Signature Gröbner Bases

Just like traditional Gröbner bases, signature Gröbner bases are not unique. Hence, we can define so-called *minimal* signature Gröbner bases as follows:

Definition 7.1 (Minimal Signature Gröbner Basis). A signature Gröbner basis is called *minimal* if, and only if, none of its elements is top $\mathfrak{s}$-reducible modulo the other elements.

Note that minimal signature Gröbner bases have nothing to do with minimal Gröbner bases: if G is a minimal signature Gröbner basis, then $\overline{G}$ is not automatically a minimal Gröbner basis, that is, there could exist $p_1, p_2 \in \overline{G}$ with $p_1 \neq p_2$ and $\mathrm{lp}(p_1) \,|\, \mathrm{lp}(p_2)$. Nevertheless, minimal signature Gröbner bases deserve the name, since they are really 'minimal' in some sense; we omit the details here, referring to Lemma 4.3 in Eder and Faugère (2017) instead.

Surprisingly, when using $\trianglelefteq_{\mathsf{rat}}$ as the rewrite order, rb-aux automatically computes minimal signature Gröbner bases (recall from Proposition 5.1 that rewrite bases are also signature Gröbner bases). The following theorem corresponds to Corollary 7.3 in Eder and Faugère (2017):

Theorem 7.2. *Assume $\trianglelefteq = \trianglelefteq_{\mathsf{rat}}$. Then the rewrite basis computed by Algorithm 1 is also a minimal signature Gröbner basis.*

Therefore, $\trianglelefteq_{\mathsf{rat}}$ is the optimal rewrite order in terms of the size of the resulting basis and the number of S-pairs that must be dealt with. Still, as noted in point (c) of Section 14.3 in Eder and Faugère (2017), other rewrite orders, like $\trianglelefteq_{\mathsf{add}}$, can lead to a comparable overall performance of the algorithm.

There is nothing special about the formalization of Definition 7.1 and Theorem 7.2 in Isabelle/HOL, as shown below:

```
definition is-min-sig-GB :: "(τ ⇒₀ β) set ⇒ bool"
  where "is-min-sig-GB G ⟷
                   G ⊆ Rm ∧ (∀u. snd u < length fs ⟶ is-sig-GB-in G u) ∧
                   (∀g∈G. ¬ is-sig-red (≺t) (=) (G - {g}) g)"

corollary rb-aux-is-min-sig-GB:
  assumes "(⊴) = (⊴rat)"
  shows "is-min-sig-GB (set (fst rb))"
```

8. Code Generation and Computations

When it comes to actually computing rewrite bases, the following two observations are important:

- Algorithm 1 and function rb operate on module elements in $\mathfrak{R}^m$ (or objects of type $\tau \Rightarrow_0 \beta$, respectively). Operations on such objects, like addition, multiplication, etc., are of course m-times more expensive than on ordinary polynomials in $\mathfrak{R}$.
- A close investigation of said algorithms and their sub-algorithms, like *regular* $\mathfrak{s}$-reduction, reveals that in fact only the signature $\mathfrak{s}(a)$ and the polynomial part $\overline{a}$ of module elements $a \in \mathfrak{R}^m$ must be known for executing the algorithms. Therefore, the whole computation of rewrite bases can be made more efficient by letting the functions operate on sig-poly-pairs (see Definition 5.1) instead of full module elements.

In the formalization, we take the preceding observations into account by *refining* function rb and all other functions it depends on to new functions that operate on sig-poly-pairs, i. e. objects of type $\tau \times (\alpha \Rightarrow_0 \beta)$. Of course, we formally prove that the refined functions behave precisely as the original ones and therefore inherit all their main properties. Eventually we end up with a function gb-sig that takes a list of polynomials as input, employs the refined version of rb-aux (called rb-spp-aux) for computing a rewrite basis of it (which is a list of sig-poly-pairs), and finally projects the elements of this list onto their second entries to obtain again a list of polynomials which constitute a Gröbner basis of the input. Furthermore, gb-sig is parameterized over $\preceq$, $\preceq_t$ and $\trianglelefteq$.

Thanks to Isabelle's *code generator*, the provenly correct function gb-sig can be used to effectively compute Gröbner bases. In a nutshell, this works by translating the definitions of gb-sig and its sub-algorithms, which are universally quantified equalities in Isabelle/HOL, into operationally equivalent procedures operating on concrete data structures in SML, OCaml, Scala or Haskell. The translation is implemented in such a way that the generated executable programs can be trusted to inherit all (correctness) properties of the abstract Isabelle-functions. More information about code generation in Isabelle can be found in Haftmann et al. (2013); Haftmann and Bulwahn (2018).

In our concrete case, multivariate polynomials are represented efficiently as ordered (w. r. t. $\preceq$) associative lists, mapping power-products to coefficients. This formally verified concrete representation, which is part of Sternagel et al. (2010), allows us to provide efficient implementations of all frequently used operations (e. g. addition, lp, etc.).

A typical invocation of gb-sig within Isabelle, which automatically triggers code generation into SML and execution of the resulting program, could look as follows:[17]

```
value [code] "gb-sig-pprod (POT DRLEX) rw-rat-strict-pprod
                [X ^ 2 * Z ^ 3 + 3 * X ^ 2 * Y, X * Y * Z + 2 * Y ^ 2]"
```

This instruction immediately returns the following 4-element Gröbner basis, computed over the field of rational numbers w. r. t. the POT extension of the degree-reverse-lexicographic ordering and rewrite order $\trianglelefteq_{\mathsf{rat}}$:

[17] The suffixes '-pprod' are technical artifacts that may safely be ignored here.

Table 1: Timings (in seconds) and total number of zero-reductions of Gröbner basis computations. '?' indicates that the computation was aborted after 1200 seconds.

	gb-sig		gb		*Mathematica*
Benchmark	Time	#0-red	Time	#0-red	Time
cyclic-5	0.1	0	0.1	79	0.0
cyclic-6	2.0	8	186.2	517	0.3
cyclic-7	544.7	36	?	?	?
katsura-6	0.9	0	9.5	159	0.5
katsura-7	22.4	0	270.0	355	3.7
katsura-8	1005.4	0	?	?	42.0
eco-9	3.0	0	24.2	685	2.8
eco-10	32.0	0	255.7	1572	27.9
eco-11	297.2	0	?	?	263.0
noon-5	0.4	0	0.5	208	0.1
noon-6	8.7	0	13.8	738	1.0
noon-7	213.5	0	289.2	2467	12.4

```
"[(3 / 4) * X ^ 3 * Y ^ 2 - 2 * Y ^ 4, - 4 * Y ^ 3 * Z - 3 * X ^ 2 * Y ^ 2,
   X * Y * Z + 2 * Y ^ 2, X ^ 2 * Z ^ 3 + 3 * X ^ 2 * Y]"
```

X, Y and Z are auxiliary constants introduced for conveniently writing down trivariate polynomials; further indeterminates can easily be added on-the-fly, without even having to adapt the underlying type. More sample computations can be found in theory *Signature-Examples* of the formalization.

Besides simple examples as the one shown above, gb-sig can also be tested on common benchmark problems and compared to other implementations of Gröbner bases. Table 1 shows such a comparison to a formally verified implementation of Buchberger's algorithm with product- and chain-criterion in Isabelle/HOL, called gb and described in Maletzky and Immler (2018a), and to function GroebnerBasis in *Mathematica* 11.3. Since this article is not meant as an exhaustive survey on the efficiency of different Gröbner basis algorithms, we confine ourselves here to present results of computations over the rationals w. r. t. the POT extension of the degree-reverse-lexicographic ordering and rewrite order $\trianglelefteq_{\mathsf{rat}}$. We shall emphasize, however, that other order relations and rewrite orders are formalized, too, and may hence be used in computations without further ado.

Remark 8.1. The timings for *Mathematica* have to be read with care: *Mathematica* always computes a reduced Gröbner basis, whereas the results returned by gb-sig and gb are not necessarily reduced. So, the timings of *Mathematica* must be understood as a mere reference mark for highly sophisticated, state-of-the-art computer algebra software. It is not surprising that our formally verified function gb-sig cannot compete with it in most cases.

9. Conclusion

In this paper we presented a formalization of signature-based algorithms for computing Gröbner bases in Isabelle/HOL. The formalization is generic, executable, and covers

not only correctness but also optimality (no zero-reductions, minimal signature Gröbner bases) of the implemented algorithms.

The formalization effort was roughly three months of full-time work. This might not sound very much, but it must once again be noted that we could make heavy use of existing formalizations of multivariate polynomials and modules thereof, as well as Gröbner bases theory, in Isabelle/HOL. Otherwise, it would have taken a lot longer. The total number of lines of code is $\sim$ 11440, distributed over the five theories *Prelims* (general facts about lists, relations, etc.; $\sim$ 960 lines), *More-MPoly* (general properties of polynomials; $\sim$ 440 lines), *Quasi-PM-Power-Products* (facts about power-products; $\sim$ 290 lines), *Signature-Groebner* (main theory; $\sim$ 9370 lines) and *Signature-Examples* (code generation and sample computations; $\sim$ 380 lines). Proofs are intentionally given in a quite verbose style for better readability.

9.1. Related Work

Even though signature-based algorithms have, to the best of our knowledge, not been formalized in any other proof assistant so far, formalizations of traditional Gröbner bases theory exist in various systems.

The first formalization of Gröbner bases dates back to Théry (2001) and Persson (2001) in the Coq proof assistant (Bertot and Castéran (2004)). Later, Schwarzweller (2006) formalized the purely theoretical aspects of the theory in Mizar (Bancerek et al. (2015)). Jorge et al. (2009) and Medina-Bulo et al. (2010) implemented formally verified versions of Buchberger's algorithm in OCaml and Common LISP, respectively; the former was verified using Coq, and the latter using ACL2 (Kaufmann et al. (2000)). And, of course, the work presented in this paper heavily rests on the formalization of traditional Gröbner bases theory by Immler and Maletzky (2016) in Isabelle/HOL.

Buchberger (2004) and Crăciun (2008) took a slightly different approach: they managed to automatically synthesize Buchberger's algorithm from a formal description of its specification in the Theorema system (Buchberger et al. (2016)). In the same system, we formalized a generalization of Gröbner bases to so-called *reduction rings* (Maletzky (2016)).

Finally, it must also be mentioned that Gröbner bases methodology for a long time has been, and still is, successfully applied in automated theorem proving, as a black-box algorithm for proving universal equalities and inequations over algebraically closed fields; see for instance Harrison (2001) and Chaieb and Wenzel (2007).

9.2. Future Work

The present formalization could be extended in several ways. First of all, function gb-sig could be improved by *inter-reducing* intermediate bases when $\preceq_{\mathsf{pot}}$ is used as the module term order. This idea, due to Stegers (2006); Eder and Perry (2010), has the potential of speeding up computations, but inter-reducing intermediate bases turns out to be much more subtle in the signature-based setting than it is in the traditional setting.

Another possible improvement of gb-sig consists of implementing the so-called F_4-style reduction, as proposed by Faugère (1999). This approach not only $\mathfrak{s}$-reduces one polynomial at a time, but several polynomials simultaneously by row-reducing certain matrices. Incidentally, the F_4 algorithm and corresponding F_4-style reduction are part of the formalization by Immler and Maletzky (2016) (described in Maletzky and Immler

(2018a)), and therefore could be incorporated into the formalization presented here with only moderate effort. The main reason why we have not done so as of yet is that no increase in performance can be expected from it in this concrete case: matrices are represented densely as immutable arrays in Isabelle/HOL, but F_4-style reductions only make sense if (typically extremely sparse) matrices are stored *efficiently*, possibly even involving some sort of compression. Formalizing better representations of sparse matrices in Isabelle/HOL is left for future work.

A third potential improvement of the efficiency of the algorithms is the use of more sophisticated data-structures, like tournament trees, kd-trees, and others. Roune and Stillman (2012) review some of these data-structures and how they can reasonably be used in the computation of Gröbner bases by signature-based algorithms.

References

Ballarin, C., 2010. Tutorial to Locales and Locale Interpretation, in: Lambán, L., Romero, A., Rubio, J. (Eds.), Contribuciones Científicas en Honor de Mirian Andrés Gómez, Servicio de Publicaciones de la Universidad de La Rioja. pp. 123–140. Part of the Isabelle documentation.

Bancerek, G., Byliński, C., Grabowski, A., Korniłowicz, A., Matuszewski, R., Naumowicz, A., Pąk, K., Urban, J., 2015. Mizar: State-of-the-art and Beyond, in: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (Eds.), Intelligent Computer Mathematics (Proceedings of CICM 2015, Washington D.C., US, July 13–17), Springer. pp. 261–279. doi:10.1007/978-3-319-20615-8_17.

Bertot, Y., Castéran, P., 2004. Interactive Theorem Proving and Program Development – Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. An EATCS Series, Springer. doi:10.1007/978-3-662-07964-5.

Buchberger, B., 1965. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal (An Algorithm for Finding the Basis Elements in the Residue Class Ring Modulo a Zero Dimensional Polynomial Ideal). Ph.D. thesis. Mathematical Institute, University of Innsbruck, Austria. English translation in *J. Symb. Comput.* 41(3–4):475–511, Special Issue on Logic, Mathematics, and Computer Science: Interactions.

Buchberger, B., 2004. Towards the Automated Synthesis of a Gröbner Bases Algorithm. RACSAM - Revista de la Real Academia de Ciencias (Review of the Spanish Royal Academy of Science), Serie A: Mathematicas 98, 65–75.

Buchberger, B., Jebelean, T., Kutsia, T., Maletzky, A., Windsteiger, W., 2016. Theorema 2.0: Computer-Assisted Natural-Style Mathematics. J. Formalized Reason. 9, 149–185. doi:10.6092/issn.1972-5787/4568.

Chaieb, A., Wenzel, M., 2007. Context aware Calculation and Deduction: Ring Equalities via Gröbner Bases in Isabelle, in: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (Eds.), Towards Mechanized Mathematical Assistants (Proceedings of Calculemus'2007, Hagenberg, Austria, June 27–30), Springer. pp. 27–39. doi:10.1007/978-3-540-73086-6_3.

Crăciun, A., 2008. Lazy Thinking Algorithm Synthesis in Gröbner Bases Theory. Ph.D. thesis. RISC, Johannes Kepler University Linz, Austria.

Eder, C., Faugère, J.C., 2017. A Survey on Signature-Based Algorithms for Computing Gröbner Bases. J. Symb. Comput. 80, 719–784. doi:10.1016/j.jsc.2016.07.031.

Eder, C., Perry, J., 2010. F5C: A Variant of Faugère's F_5 Algorithm with Reduced Gröbner Bases. J. Symb. Comput. 45, 1442–1458. doi:10.1016/j.jsc.2010.06.019.

Eder, C., Roune, B.H., 2013. Signature Rewriting in Gröbner Basis Computation, in: Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation, ACM. pp. 331–338. doi:10.1145/2465506.2465522.

Faugère, J.C., 1999. A New Efficient Algorithm for Computing Gröbner Bases (F_4). J. Pure and Applied Algebra 139, 61–88. doi:10.1016/S0022-4049(99)00005-5.

Faugère, J.C., 2002. A New Efficient Algorithm for Computing Gröbner Bases Without Reduction to Zero (F_5), in: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, ACM. pp. 75–83. doi:10.1145/780506.780516.

Galkin, V., 2012. Termination of Original F_5. arXiv:1203.2402 [math.AC].

Haftmann, F., Bulwahn, L., 2018. Code Generation from Isabelle/HOL Theories. Part of the Isabelle documentation.

Haftmann, F., Krauss, A., Kunčar, O., Nipkow, T., 2013. Data Refinement in Isabelle/HOL, in: Blazy, S., Paulin-Mohring, C., Pichardie, D. (Eds.), Interactive Theorem Proving (Proceedings of ITP'2013, Rennes, France, July 22–26), Springer. pp. 100–115. doi:10.1007/978-3-642-39634-2_10.

Harrison, J., 2001. Complex quantifier elimination in HOL, in: Boulton, R.J., Jackson, P.B. (Eds.), TPHOLs 2001: Supplemental Proceedings, Division of Informatics, University of Edinburgh. pp. 159–174. URL: http://www.informatics.ed.ac.uk/publications/report/0046.html.

Immler, F., Maletzky, A., 2016. Gröbner Bases Theory. Archive of Formal Proofs http://afp.sf.net/entries/Groebner_Bases.shtml, Formal proof development.

Jorge, J.S., Guilas, V.M., Freire, J.L., 2009. Certifying properties of an efficient functional program for computing Gröbner bases. J. Symb. Comput. 44, 571–582. doi:10.1016/j.jsc.2007.07.016.

Kaufmann, M., Manolios, P., Moore, J.S., 2000. Computer-Aided Reasoning: An Approach. Kluwer Academic Publishers. doi:10.1007/978-1-4615-4449-4.

Maletzky, A., 2016. Computer-Assisted Exploration of Gröbner Bases Theory in Theorema. Ph.D. thesis. RISC, Johannes Kepler University Linz.

Maletzky, A., 2018. Signature-Based Gröbner Basis Algorithms. Archive of Formal Proofs http://isa-afp.org/entries/Signature_Groebner.html, Formal proof development.

Maletzky, A., Immler, F., 2018a. Gröbner Bases of Modules and Faugère's F_4 Algorithm in Isabelle/HOL, in: Rabe, F., Farmer, W., Passmore, G., Youssef, A. (Eds.), Intelligent Computer Mathematics (Proceedings of CICM 2018, Hagenberg, Austria, August 13-17), Springer. pp. 178–193. doi:10.1007/978-3-319-96812-4_16.

Maletzky, A., Immler, F., 2018b. Gröbner Bases of Modules and Faugère's F_4 Algorithm in Isabelle/HOL (extended version). Technical Report. RISC, JKU Linz. ArXiv:1805.00304 [cs.LO].

Medina-Bulo, I., Palomo-Lozano, F., Ruiz-Reina, J.L., 2010. A verified Common Lisp Implementation of Buchberger's Algorithm in ACL2. J. Symb. Comput. 45, 96–123. doi:10.1016/j.jsc.2009.07.002.

Nipkow, T., Paulson, L.C., Wenzel, M., 2002. Isabelle/HOL—A Proof Assistant for Higher-Order Logic. volume 2283 of *Lecture Notes in Computer Science*. Springer. doi:10.1007/3-540-45949-9.

Pan, S., Hu, Y., Wang, B., 2012. The Termination of Algorithms for Computing Gröbner Bases. arXiv:1202.3524 [math.AC].

Paulson, L.C., 1994. Isabelle: A Generic Theorem Prover. volume 828 of *Lecture Notes in Computer Science*. Springer. doi:10.1007/BFb0030541.

Persson, H., 2001. An Integrated Development of Buchberger's Algorithm in Coq. Technical Report 4271. INRIA Sophia Antipolis.

Roune, B.H., Stillman, M., 2012. Practical Gröbner Basis Computation, in: Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation, ACM. pp. 203–210. doi:10.1145/2442829.2442860.

Schwarzweller, C., 2006. Gröbner Bases – Theory Refinement in the Mizar System, in: Kohlhase, M. (Ed.), Mathematical Knowledge Management (4th International Conference, Bremen, Germany, July 15–17), Springer. pp. 299–314. doi:10.1007/11618027_20.

Stegers, T., 2006. Faugère's F_5 Algorithm Revisited. Master's thesis. Technische Universität Darmstadt, Germany.

Sternagel, C., Thiemann, R., Maletzky, A., Immler, F., Haftmann, F., Lochbihler, A., Bentkamp, A., 2010. Executable Multivariate Polynomials. Archive of Formal Proofs http://afp.sf.net/entries/Polynomials.shtml, Formal proof development.

Théry, L., 2001. A Machine-Checked Implementation of Buchberger's Algorithm. J. Autom. Reason. 26, 107–137. doi:10.1023/A:1026518331905.

Wenzel, M., 2018. The Isabelle/Isar Reference Manual. Part of the Isabelle documentation.

Table A.2: Translations of concepts between informal mathematics, the formalization as presented in this paper, and the actual Isabelle sources of the formalization.

Mathematics	Formalization (paper)	Formalization (sources)
$\mathfrak{R}$	$\alpha \Rightarrow_0 \beta$	$\alpha \Rightarrow_0 \beta$
$\mathfrak{R}^m$	$\tau \Rightarrow_0 \beta$; Rm	$\tau \Rightarrow_0 \beta$; dgrad-sig-set
$\langle \cdot \rangle$	ideal	ideal
$(f_1, \ldots, f_m)$	fs	fs
$\overline{a}$	poly a	rep-list a
supp	supp	keys
coeff	coeff	lookup
$\preceq$	$\preceq$	$\preceq$
$\preceq_{\mathsf{t}}$	$\preceq_{\mathsf{t}}$	$\preceq_{\mathsf{t}}$
lp	lp	punit.lt
lc	lc	punit.lc, lc
$\mathfrak{s}$	$\mathfrak{s}$	lt
$\xrightarrow{\mathrm{r}_1,\mathrm{r}_2}_G$	sig-red r_1 r_2 G	sig-red r_1 r_2 G
$t\,u \quad (t \in [X],\, u \in \mathfrak{T})$	$t \otimes u$	$t \oplus u$
$u \,\vert\, v \quad (u, v \in \mathfrak{T})$	u $\mathsf{dvd}_{\mathsf{t}}$ v	u $\mathsf{adds}_{\mathsf{t}}$ v
$c\,t\,a \quad (c \in \mathfrak{K},\, t \in [X],\, a \in \mathfrak{R}^m)$	monom-mult c t a	monom-mult c t a
$(\mathfrak{s}(a), \overline{a})$	spp-of a	spp-of a
$\trianglelefteq_{\mathsf{rat}}$	$\trianglelefteq_{\mathsf{rat}}$	rw-rat
$\trianglelefteq_{\mathsf{add}}$	$\trianglelefteq_{\mathsf{add}}$	rw-add

Appendix A. Translation between Mathematics and Formalization

Table A.2 lists several concepts of the theory and how they translate into our formalization as presented in this exposition, and into the actual Isabelle sources of the formalization. Differences between the latter two stem from increasing the readability of the paper and have no deeper significance; in fact, readers not intending to look at the Isabelle sources may safely ignore the last column.