Gröbner Bases of Modules andFaçgère’s $F_4$ Algorithm in Isabelle/HOL
– extended version –

Alexander Maletzky* 1 and Fabian Immler† 2

1 RISC, Johannes Kepler Universität Linz, Austria,
amAlexander.maletzky@risc.jku.at
2 Institut für Informatik, Technische Universität München, Germany,
immler@in.tum.de

Abstract
We present an elegant, generic and extensive formalization of Gröbner bases in Isabelle/HOL. The formalization covers all of the essentials of the theory (polynomial reduction, S-polynomials, Buchberger’s algorithm, Buchberger’s criteria for avoiding useless pairs), but also includes more advanced features like reduced Gröbner bases. Particular highlights are the first-time formalization of Faugère’s matrix-based $F_4$ algorithm and the fact that the entire theory is formulated for modules and submodules rather than rings and ideals. All formalized algorithms can be translated into executable code operating on concrete data structures, enabling the certified computation of (reduced) Gröbner bases and syzygy modules.

1 Introduction
Since their origins in Buchberger’s PhD thesis [6], Gröbner bases have become one of the most powerful and most widely used tools in computer algebra, a claim which is supported, for instance, by the 3400+ publications currently listed in the online Gröbner Bases Bibliography[1]. Their importance stems from the fact that they generalize, at the same time, Gauss’ algorithm for solving systems of linear equations and Euclid’s algorithm for computing the GCD of univariate polynomials: Gröbner bases enable the effective, systematic solution of a variety of problems in polynomial ideal theory, ranging from the decision of ideal membership and ideal congruence, the solution of systems of algebraic equations, to as far as automatic theorem proving. Since it is clearly beyond the scope of this paper to mention all the merits, applications, and generalizations of Gröbner bases, we refer the interested reader to any standard textbook about Gröbner bases (e.g. [1] [23]) instead; nonetheless, Section 2 briefly presents the mathematical background of Gröbner bases.

*The research was funded by the Austrian Science Fund (FWF): P 29498-N31
†The research was funded by DFG Koelleck Grant NI 491/16-1
[1]http://www.risc.jku.at/Groebner-Bases-Bibliography/
The main achievement we report on in this paper is the first-time formalization of said theory in the proof assistant Isabelle/HOL [27]. Although Gröbner bases have been formalized in other proof assistants already (a list of which can be found in Section 1.1), our work features the, to the best of our knowledge, first computer-certified implementation of Faugère’s $F_4$ algorithm [13] for computing Gröbner bases by matrix reductions, as well as the (again to the best of our knowledge) first-time formal treatment of the theory in the more general setting of modules and submodules rather than rings and ideals [23].

Summarizing, the highlights of our elaboration are

- an abstract view of power-products that does not refer to the notion of “indeterminate” at all and that allows us to represent power-products by functions of type nat ⇒ nat with finite support (Section 3.1),
- an abstract view of vectors of polynomials, that can be interpreted both by ordinary scalar polynomials and by functions mapping indices (of the components) to scalar polynomials (Section 3.2),
- the definition of Gröbner bases via confluence of the reduction relation they induce (definition is-Groebner-basis in Section 4.2),
- the proof of the main theorem about S-polynomials (theorem Buchberger-thm-finite in Section 4.2),
- an alternative characterization of Gröbner bases via divisibility of leading terms (theorem GB-alt-finite in Section 4.2),
- a generic algorithm schema for computing Gröbner bases, of which both Buchberger’s algorithm and Faugère’s $F_4$ algorithm are instances (function gb_schema in Section 4.3),
- the implementation of Buchberger’s criteria for increasing the efficiency of said algorithm schema (Section 4.5),
- the formally verified implementation of the $F_4$ algorithm (Section 5),
- the definition and a constructive proof of existence and uniqueness of reduced Gröbner bases (Section 6), and
- a formally verified algorithm for computing Gröbner bases of syzygy modules (Section 7),
- the proper set-up of Isabelle’s code generator to produce certified executable code (Section 8).

An Isabelle2017-compatible version of the formalization presented in this paper is available online [20]. Furthermore, a big portion of the formalization has already been added to the development version of the Archive of Formal Proofs (AFP). Note also that there is a Gröbner-bases entry in the release version of the AFP [19] (which will be replaced by the one in the development version upon the next release of Isabelle), but it lacks many features compared to [20].
1.1 Related Work

Gröbner bases have been formalized in a couple of other proof assistants already. The first formalizations date back to around 2000, when Théry [33] and Persson [28] formalized basically the same aspects in Coq [5] that we recently formalized in Isabelle/HOL (except $F_4$ and modules). The presentation of their theory is on a fairly abstract level, similar to our case. Moreover, it is possible to automatically extract executable, certified OCaml code for computing Gröbner bases from the formalization. In 2009, Jorge, Guilas and Freire [21] took the reverse direction: they first implemented an efficient version of Buchberger’s algorithm directly in OCaml and then proved it correct, making use of the underlying formal theory in Coq.

Another formalization that focuses very much on the actual computation of Gröbner bases is that of Medina-Bulo, Palomo-Lozano, Alonso-Jiménez and Ruiz-Reina [26] in ACL2 [22], dating back to 2010. There, however, the representation of power-products and polynomials is fixed to ordered lists of exponents and monomials, respectively, owing to the limited expressiveness of the underlying system.

In 2006, Schwarzweller [29] formalized Gröbner bases in Mizar [3]. He also dealt with polynomial reduction, Buchberger’s algorithm and reduced Gröbner bases, and in addition proved some other equivalent characterizations of Gröbner bases (e.g. via so-called standard representations of polynomials).

Very recently, the first author formalized a generalization of Gröbner bases [24] in the proof assistant Theorema 2.0 [9]. His work is not confined to polynomial rings over fields, but considers much wider classes of commutative rings where Gröbner bases can be defined and computed (so-called reduction rings). All certified algorithms are directly executable within Theorema. In the same system, Buchberger [8] and Craciun [11] took Buchberger’s algorithm as a case study for the automatic synthesis of algorithms: they managed to synthesize the algorithm only from its specification by the so-called lazy thinking method.

Apart from the formalizations listed above, Gröbner bases have been successfully employed by various proof assistants (among them HOL [18] and Isabelle/HOL [10]) as proof methods for proving universal propositions over rings. In a nutshell, this proceeds by showing that the system of polynomial equalities and inequalities arising from refuting the original formula is unsolvable, which in turn is accomplished by finding a combination of these polynomials that yields a non-zero constant polynomial – and this is exactly what Gröbner bases can do. The computation of Gröbner bases, however, is taken care of by a “black-box” ML program whose correct behavior is irrelevant for the correctness of the overall proof step, since the obtained witness is independently checked by the trusted inference kernel of the system. The work described in this paper is orthogonal to [10] in the sense that it formalizes the theory underlying Gröbner bases and proves the total correctness of the algorithm, which is not needed in [10].

Our work builds upon existing formal developments of multivariate polynomials [32] (to which we also contributed) and abstract rewrite systems [31], and $F_4$ in addition builds upon Gauss-Jordan normal forms of matrices [34].
2 Mathematical Background

We now give a very brief overview of the mathematical theory of Gröbner bases for (sub)modules, only to keep this paper self-contained. The interested reader is referred to any standard textbook about Gröbner bases, e.g. [1, 23], for a more thorough account on the subject; in particular, [23] also presents Gröbner bases for modules. Readers familiar with Gröbner bases in rings but not with Gröbner bases in modules will spot only few differences to the ring-setting, as the module-setting parallels the other in many respects.

The theory of Gröbner bases for modules is concerned with vectors of commutative multivariate polynomials over fields, and more precisely with effectively solving module-theoretic problems. Hence, let in the remainder of this section $K$ be some field, $X = \{x_0, \ldots, x_{n-1}\}$ be a finite set of indeterminates, $k \neq 0$ be a natural number, and let $K[X]^k = K[x_0, \ldots, x_{n-1}]^k$ denote the $k$-dimensional free module over the (commutative) ring of $n$-variate polynomials over $K$. We will refer to products of indeterminates as power-products (e.g. $x_0, x_2^2x_3^3$); the set of all power-products in the indeterminates $X$ is denoted by $[X]$. The $k$-elements of the canonical basis of $K^k$ are denoted by $e_j$, for $0 \leq j < k$; hence, every element $p$ of $K[X]^k$ can be uniquely written as a sum $p = \sum_{j=0}^k p_j e_j$ for polynomials $p_j \in K[X]$. Terms are polynomials of the form $t e_j$ for a power-product $t$ and $0 \leq j < k$; their importance stems from the fact that the set of terms, denoted by $[X]^k$, is a basis of the infinite-dimensional $K$-vector space $K[X]^k$.

Example 1. Let $p = \left( x_0^2 - x_0x_1 \right) \in \mathbb{Q}[x_0, x_1]^2$. Then $p$ can be written as a linear combination of terms as

$$p = 1 \cdot x_0^2 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}_{e_0} + (-1) \cdot x_0x_1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}_{e_0} + 2 \cdot x_0 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix}_{e_1} + 3 \cdot 1 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix}_{e_1}.$$  

2.1 Gröbner Bases

First of all, we now have to choose an admissible order on the power-products: a linear order $\preceq$ on power-products is called admissible iff 1 is the smallest element and if $s \preceq t$ implies $s \cdot u \preceq t \cdot u$, for all $s, t, u \in [X]$. An example of such an order is the purely lexicographic order, which compares two power-products $s, t$ by successively comparing the exponents of the $x_i$ in $s$ and $t$ until some are non-equal. In the remainder of this section we fix an admissible order $\preceq$.

In the next step, we must extend $\preceq$ to an ordering $\preceq_t$ on terms. As before, we have some freedom in doing so, as long as (i) $s e_i \preceq_t t e_j$ implies $(u \cdot s) e_i \preceq_t (u \cdot t) e_j$ for all $s, t, u \in [X]$, and (ii) $s \preceq_t t \wedge i \leq j$ implies $s e_i \preceq_t t e_j$ for all $s, t \in [X]$. Examples of such $\preceq_t$ are position-over-term (POT) and term-over-position orders: in a POT order $\preceq_{\text{pot}}^t$ we have $s e_i \preceq_{\text{pot}}^t t e_j$ iff $i < j \vee (i = j \wedge s \preceq_t)$, whereas in a TOP order $\preceq_{\text{top}}^t$ we have $s e_i \preceq_{\text{top}}^t t e_j$ iff $s \preceq t \vee (s = t \wedge i \leq j)$. Hence, both POT and TOP are lexicographic combinations of $\preceq$ and $\preceq_t$, and easily seen to satisfy the two requirements listed above.

Having $\preceq_t$, we can define the notions of leading term, leading power-product and leading coefficient of non-zero vector-polynomials $p \in K[X]^k$: the leading term of $p$, written $\uparrow t(p)$, is simply the largest term w.r.t. to $\preceq_t$ that appears
in \( p \) with non-zero coefficient, the leading power-product \( \ell p(p) \) is the power-product of the leading term, and the leading coefficient \( \ell c(p) \) is the coefficient of \( \ell t(p) \) in \( p \). In general, the coefficient of a term \( v \) in a polynomial \( p \) is denoted by \( \text{coeff}(p,v) \).

**Example 2.** Let \( p \) as in Example 1 and assume \( \preceq \) is the lexicographic order relation with \( x_0 \prec x_1 \). With \( \preceq_{\text{tot}} \) we obtain \( \ell t(p) = x_0 e_1 \) and \( \ell c(p) = 2 \); with \( \preceq_{\text{top}} \), \( \ell t(p) = x_1^2 e_0 \) and \( \ell c(p) = 1 \).

**Definition 1** (Reduction). Let \( p,q,f \in K[X]^k \) with \( f \neq 0 \) and \( t \in [X] \). Then \( p \) reduces to \( q \) modulo \( f \) using \( t \), written \( p \rightarrow_{f,t} q \), if \( \ell t(f) = s e_j \), \( \text{coeff}(p,(t \cdot s) e_j) \neq 0 \) and \( q = p - \frac{\text{coeff}(p,(t \cdot s) e_j)}{\ell c(f)} \cdot t \cdot f \).

If \( F \subseteq K[X]^k \) then write \( p \rightarrow_F q \) iff there exist \( f \in F \setminus \{0\} \) and \( t \in [X] \) with \( p \rightarrow_{f,t} q \). As usual, \( \rightarrow^*_F \) denotes the reflexive-transitive closure of \( \rightarrow_F \).

For any \( F \subseteq K[X] \), \( \rightarrow_F \) can be shown to be terminating, i.e. there do not exist infinite chains of reductions. However, in general \( \rightarrow_F \) is not confluent, as can be seen in the following example:

**Example 3** (Example 2.5.7. in [23]). Let \( F = \{ f_1, f_2 \} \subseteq Q[x_0, x_1] \cong Q[x_0, x_1]^1 \) where \( f_1 = x_1^2 \) and \( f_2 = x_0 x_1 + x_0^2 \), and assume \( \preceq \) and \( \preceq_{\text{tot}} \) are the lexicographic order with \( x_0 \prec x_1 \). Then \( x_0 x_1^2 \rightarrow_{f_1, x_0} 0 \) and \( x_0 x_1^2 \rightarrow_{f_2, x_1} -x_0^2 x_1 \rightarrow_{f_2, x_0} x_0^3 \); both 0 and \( x_0^3 \) are irreducible modulo \( F \), so \( \rightarrow_F \) is not confluent.

The observation that \( \rightarrow_F \) is in general not confluent motivates the following

**Definition 2** (Gröbner basis). A set \( G \subseteq K[X]^k \) is a Gröbner basis iff \( \rightarrow_G \) is confluent.

Note that the notion of reduction, and hence also that of Gröbner basis, strongly depends on the implicitly fixed term order \( \preceq_{\text{tot}} \)!

If a set \( F \) is no Gröbner basis, it can be completed to one by successively considering all critical pairs and adding new elements to the set that ensure that all critical pairs have a common successor w.r.t. the reduction relation, just as in the well-known Knuth-Bendix procedure for general term rewrite systems. In the case of multivariate polynomials the algorithm is called Buchberger’s algorithm and has the nice property that it always terminates for any finite input \( F \).

**Definition 3** (S-polynomial). Let \( f,g \in K[X]^k \setminus \{0\} \), and assume \( \ell t(f) = s e_i \) and \( \ell t(g) = t e_j \). Then the S-polynomial of \( f \) and \( g \), written \( \text{spoly}(f,g) \), is defined as

\[ \text{spoly}(f,g) := \begin{cases} \frac{\ell c(f) s}{\ell c(f) t} \cdot f - \frac{\ell c(f) t}{\ell c(g) t} \cdot g & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \]

The S-polynomial of \( f \) and \( g \) is precisely the difference of the critical pair of \( f \) and \( g \), so it roughly corresponds to the smallest element where reduction modulo \( \{ f,g \} \) might diverge. In the usual Knuth-Bendix procedure one reduces the two constituents of a critical pair individually and then checks whether the normal forms are equal; in our case, it suffices to first compute their difference (i.e. the S-polynomial), then reduce the S-polynomial to normal form, and finally check whether the normal form is 0. This idea is summarized in the following
Theorem 4 (Buchberger, 1965). Let \( G \subseteq K[X]^k \). Then \( G \) is a Gröbner basis iff for all \( f, g \in G \), \( \text{spoly}(f, g) \to G^0 \).

Therefore, completing a finite set \( F \) to a Gröbner basis simply proceeds by

1. forming all S-polynomials of elements in the current set,
2. reducing them to normal form modulo the current set,
3. adding those normal forms that are not 0 to the current set (which means that they can be further reduced to 0 modulo the enlarged set, since we always have \( p \to p, 0 \)), and
4. repeating this procedure until all S-polynomials can be reduced to 0.

This procedure is called Buchberger’s algorithm, which is justified by

Theorem 5 (Buchberger, 1965). The procedure outlined above terminates after finitely many iterations, for any finite input set \( F \) and admissible term order \( \preceq_t \), and regardless in which order and how the S-polynomials are reduced.

Example 4. Let \( F, f_1, f_2, \preceq \) and \( \preceq_t \) be as in Example 3. We apply Buchberger’s algorithm to compute a Gröbner basis of \( F \).

We start with the S-polynomial of \( f_1 \) and \( f_2 \):

\[
\text{spoly}(f_1, f_2) = x_0 f_1 - x_1 f_2 = -x_0^2 x_1 \to f_2, x_0 x_1^3.
\]

\( f_3 = x_0^2 \) is irreducible modulo \( F \), so we must add it to \( F \): \( \tilde{F} = F \cup \{f_3\} \). This ensures that \( \text{spoly}(f_1, f_2) \) can be reduced to 0 modulo the enlarged set \( \tilde{F} \), but it also means that we have to consider \( \text{spoly}(f_1, f_3) \) and \( \text{spoly}(f_2, f_3) \) as well.

\[
\text{spoly}(f_1, f_3) = x_0^3 f_1 - x_1^2 f_3 = 0.
\]

Since the S-polynomial of \( f_1 \) and \( f_3 \) is 0, we do not have to augment \( \tilde{F} \) by a new polynomial.

\[
\text{spoly}(f_2, f_3) = x_0^3 f_2 - x_1 f_3 = x_0^4 \to f_3, x_0 0.
\]

Since the S-polynomial of \( f_2 \) and \( f_3 \) can be reduced to 0 modulo \( \tilde{F} \), and therefore all S-polynomials can be reduced to 0, \( \tilde{F} \) is a Gröbner basis of \( F \).

2.2 Submodules

We assume familiarity with the concept of a submodule of a module \( M \), as a subset of \( M \) that is closed under addition and under multiplication by arbitrary elements from \( M \). Regarding notation, we write \( \text{pmd}(F) \subseteq M \) for the submodule generated by the set \( F \subseteq M \). In our case, \( M \) is of course \( K[X]^k \); if \( k = 1 \), submodules are nothing else than ideals.

As can be easily seen, Buchberger’s algorithm preserves the submodule generated by the set in question, i.e. if Buchberger’s algorithm applied to \( F \) yields \( G \), then \( \text{pmd}(F) = \text{pmd}(G) \). Hence, we can conclude that every finitely generated submodule of \( K[X]^k \) has a finite Gröbner basis; this Gröbner basis is not unique, though: first of all it clearly depends on \( \preceq_t \), but even if \( \preceq_t \) is fixed, \( G \)
is a Gröbner basis (w.r.t. \( \preceq_t \)) and \( H \subseteq \text{pmdl}(G) \), then \( G \cup H \) is still a Gröbner basis of \( \text{pmdl}(G) \).

One important property of Gröbner bases \( G \) is that the unique normal form w.r.t. \( \to_G \) of an arbitrary polynomial \( p \) is 0 iff \( p \in \text{pmdl}(G) \). So, since normal forms are effectively computable, also the submodule membership problem is effectively decidable if the submodule \( N \) in question is given by a finite generating set \( F \); just compute a Gröbner basis \( G \) of \( F \) by Buchberger’s algorithm, reduce the polynomial in question to normal form w.r.t. \( G \), and check whether the result is 0.

**Example 5.** Continuing Example 4: \( x_0^5 \in \text{pmdl}(F) \), because \( x_0^5 \to f_3, x_0^2 0 \) and \( f_3 \) is contained in the Gröbner basis \( \tilde{F} \) computed from \( F \). However, \( x_0^5 \) is irreducible modulo \( F \) itself, illustrating that it is really crucial to perform the reduction modulo a Gröbner basis when deciding submodule membership, not just modulo any generating set.

### 3 Multivariate Polynomials

Since Gröbner bases are concerned with (vectors of) multivariate polynomials, we have to spend some words on the formalization of such polynomials in Isabelle/HOL.

The formal basis of multivariate polynomials are so-called polynomial mappings, originally formalized by Haftmann et al. [17], extended by Bentkamp [4], and now part of the AFP-entry Polynomials [32] in the development version of the Archive of Formal Proofs. A polynomial mapping is simply a function of type \( \alpha \Rightarrow \beta :: \text{zero} \) with finite support, i.e., all but finitely many arguments are mapped to 0. In Isabelle/HOL, as well as in the remainder of this paper, the type of polynomial mappings is called `poly_mapping` and written in infix form as \( \alpha \Rightarrow 0 \beta \), where \( \beta \) is tacitly assumed to belong to type class `zero`. Formally, `poly_mapping` is defined as

```isabelle
typedef (overloaded) (\alpha, \beta) poly_mapping = \{f::\alpha \Rightarrow \beta :: \text{zero}. \text{finite} \{x. f x \neq 0\}\}
```

The importance of type `poly_mapping` stems from the fact that not only polynomials, but also power-products (i.e. products of indeterminates, like \( x_0^3 x_1^2 \)) can best be thought of as terms of this type: in power-products, indeterminates are mapped to their exponents (with only finitely many being non-zero), and in polynomials, power-products are mapped to their coefficients (again only finitely many being non-zero). Hence, a scalar polynomial would typically be a term of type \( \chi \Rightarrow 0 \text{nat} \Rightarrow 0 \beta \), where \( \beta \) is tacitly assumed to belong to type class `zero`. Formally, `poly_mapping` is defined as

\[
\text{typedef (overloaded) } (\alpha, \beta) \text{ poly_mapping} = \{f::\alpha \Rightarrow \beta :: \text{zero}. \text{finite} \{x. f x \neq 0\}\}
\]

The importance of type `poly_mapping` stems from the fact that not only polynomials, but also power-products (i.e. products of indeterminates, like \( x_0^3 x_1^2 \)) can best be thought of as terms of this type: in power-products, indeterminates are mapped to their exponents (with only finitely many being non-zero), and in polynomials, power-products are mapped to their coefficients (again only finitely many being non-zero). Hence, a scalar polynomial would typically be a term of type \( \chi \Rightarrow 0 \text{nat} \Rightarrow 0 \beta \), with \( \chi \) being the type of the indeterminates and \( \beta \) being the type of the coefficients.

**Remark 1.** Although the mathematical theory our formalization is concerned with clearly belongs to the field of Algebra, we did not follow HOL-Algebra’s approach to algebraic structures with explicit carrier sets, but instead based everything on `type classes`. In particular, the coefficient type of polynomials must belong to type class `field`, and the type of power-products must belong to the custom-made type class `graded_dickson_powerprod` – this already indicates that we developed the theory more abstractly than fixing the type of power-products to something like \( \chi \Rightarrow 0 \text{nat} \); see below for details.

\[\beta :: \text{zero}\] is a type-class constraint on type \( \beta \), stipulating that there must be a distinguished constant 0 of type \( \beta \). See [10] for information on type classes in Isabelle/HOL.
3.1 Power-Products

Instead of fixing the type of power-products to $\chi \Rightarrow_0 \text{nat}$ throughout the formalization, we opted to develop the theory slightly more abstractly: power-products can be of arbitrary type, as long as the type belongs to a certain type class that allows us to prove all key results of the theory. Said type class is called `graded.dickson_powerprod` and defined as

```plaintext
class graded.dickson_powerprod = cancel.comm.monoid.mult + dvd +
  fixes lcm :: "\alpha \Rightarrow \alpha \Rightarrow \alpha"
  assumes dvd.lcm: "s dvd (lcm s t)"
  assumes lcm.dvd: "s dvd u ==> t dvd u ==> (lcm s t) dvd u"
  assumes lcm.comm: "lcm s t = lcm t s"
  assumes times.eq.one: "s * t = 1 ==> s = 1"
  assumes ex.dgrad: "\exists d::\alpha \Rightarrow \text{nat}. dickson_grading d"
```

Several remarks on the above definition are in order. First of all, note that the base class of `graded.dickson_powerprod` is the class of cancellative commutative multiplicative monoids which in addition feature a divisibility relation (infix `dvd`). Furthermore, types belonging to the the class must also

- provide a function called `lcm` that possesses the usual properties of least common multiple,
- obey the law that a product of two factors can only be 1 if both factors are 1, i.e. 1 is the only invertible element, and
- admit a so-called Dickson grading.

Dickson gradings are a technicality we need for being able to represent power-products conveniently as functions of type `nat \Rightarrow_0 \text{nat}` (with an infinite supply of indeterminates) in actual computations, without having to having to care how many indeterminates actually appear in the computations. Gröbner bases normally only work with a fixed finite set of indeterminates, for otherwise the reduction relation $\rightarrow_F$ of Definition 1 is not terminating in general. Therefore, if we want to work with potentially infinitely many indeterminates, we need a means to ensure that only finitely many appear with non-zero exponent and non-zero coefficient in possibly infinite sets of polynomials – and this is exactly what a Dickson grading $d$ does. In fact, $d$ can best be thought of giving, for an abstract power-product $t$, the “index” of the highest indeterminate occurring in $t$ with non-zero exponent. So, for ensuring that only finitely many indeterminates appear in a set $F$ of polynomials, it suffices to stipulate the existence of some natural number $m$ such that $d(t) \leq m$ for all power-products $t$ appearing in $F$. Dickson gradings are called such because they ensure, by definition of `dickson.grading`, the Dickson property \cite{12} of sequences of abstract power-products:

```plaintext
lemma dickson.property:
  fixes s::"nat \Rightarrow \alpha"
  assumes "dickson.grading d" and "\forall i. d (s i) \leq d (s 0)"
  obtains i j where "i < j" and "(s i) dvd (s j)"
```

Thus, in any sequence $s$ of abstract power-products in which only finitely many indeterminates appear (second assumption of `dickson-property`), there exist indices $i < j$ such that $s_i$ divides $s_j$. In other words, the divisibility relation

\footnote{This representation is also suggested in \cite{17}.}
is some sort of *well-quasi order* on power-products. Dickson’s lemma, finally, states that such Dickson gradings indeed exist for type nat ⇒₀ nat: in a power-product of type nat ⇒₀ nat, just take the largest number that is not mapped to 0.

**Remark 2.** In the actual formalization, power-products are written additively rather than multiplicatively: the base class is cancel_comm_monoid_add, the monoid operation is +, the neutral element is 0, the least common multiple is called lcs (standing for “least common sum”) and the divisibility relation is called adds. The reason for doing so is a mere technicality: it better integrates with the existing type-class hierarchy of Isabelle/HOL, allowing us to reuse existing point-wise instantiations of the various group- and ring-related type classes by the function type ⇒.

Finally, in order to formalize Gröbner bases we need to fix an admissible order relation on power-products, as explained in Section 2. Since there are infinitely many of them we do not want to restrict the formalization to a particular one but instead parametrized all subsequent definitions, theorems and algorithms over such a relation through the use of a locale [2] (note that we have to provide both the reflexive and the strict version of the relation):

```isar
locale gd.powerprod = 
  linorder ord ord_strict
  for ord::"α ⇒ α::graded.dickson_powerprod ⇒ bool" (infix "⪯" 50)
  and ord_strict (infixl "≺" 50) +
  assumes one_min: "1 ⪯ t"
  assumes times_monotone: "s ⪯ t =⇒ s * u ⪯ t * u"
```

In addition, the formalization features three concrete orders on type nat ⇒₀ nat that are proved to be admissible: the purely lexicographic order lex, the degree-lexicographic order dlex, and the degree-reverse-lexicographic order drlex. Each of these orders can without any further ado be used in certified computations of Gröbner bases in Isabelle/HOL.

### 3.2 Polynomials

Having described our formalization of power-products, we now turn to polynomials. In fact, most definitions related to, and facts about, multivariate polynomials as objects of type α ⇒₀ β that are required by our Gröbner bases formalization were already formalized [17, 4]: addition, multiplication, coefficient-lookup (called lookup in the formal theories but denoted by the more intuitive coeff in the remainder) and support (called keys). These things are all pretty much standard, so we do not go into more detail here. We only emphasize that henceforth α is the type of power-products, i.e. is tacitly assumed to belong to type-class graded_dickson_powerprod.

What is certainly more interesting is the way how we represent vectors of polynomials: since we formulate the theory of Gröbner bases for free modules over polynomial rings over fields, i.e. for structures of the form $K[x_0, \ldots, x_n]^k$, we need to specify what the formal type of such structures is in our formalization. Going back to Section 2 one realizes that the best way (for our purpose) to represent vector-polynomials is as $K$-linear combinations of terms $t e_i$. Or, in

---

*Well-quasi orders and Dickson’s lemma have been formalized in Isabelle/HOL already [30], but we proved the lemma in our (slightly more general) setting from scratch.*
other words, as polynomial mappings mapping terms to coefficients. Terms, in
turn, can most conveniently be represented as pairs of power-products (like \( t \))
and component-indices (like \( i \)), without having to care what exactly the basis
elements \( e_i \) are. Therefore, the formal type of vector polynomials is \((\alpha \times \kappa) \Rightarrow_0 \beta\), where \( \alpha \) and \( \beta \) have their usual meaning as type of power-products and
coefficients, respectively, and \( \kappa \) is the type of component-indices.

Example 6. Let \( p \) be as in Example \(^1\). Then \( p \) is represented by the polynomial
mapping which maps the pair \((x_1^2, 0)\) to 1, \((x_0x_1, 0)\) to \(-1\), \((x_0, 1)\) to 2, \((1, 1)\) to 3,
and all other pairs to 0.

As Example \(^1\) shows, for representing a \( k \)-dimensional vector \( \kappa \) does not
need to have exactly \( k \) elements, but only \( at \) least \( k \) elements. This saves us
from introducing dedicated types for 1, 2, 3, \ldots dimensions, as we can use \( \text{nat} \)
throughout. Nonetheless, we do not fix \( \kappa \) to \( \text{nat} \), because in some situations
it is desirable to restrict definitions or theorems to one-dimensional (scalar)
polynomials, while still building upon concepts defined for vector-polynomials
and arbitrary \( \kappa \). This can be achieved easily by instantiating \( \kappa \) by the unit type \( \text{unit} \).

Remark 3. If \( \kappa \) is instantiated by \( \text{nat} \) we get similar problems as with infinitely
many indeterminates and, hence, must provide similar means for ensuring that
only finitely many components in (infinite) sets of infinite-dimensional polyno-
mials are non-zero. We omit the (not so intricate) details here.

Let us now turn to the extension \( \preceq_4 \) of \( \preceq \): since \( \preceq \) can be extended in many
different ways to \( \preceq_4 \), and we do not want to restrict ourselves to a particular
choice, we again employ a locale for parametrizing all subsequent definitions
and lemmas over any admissible instance of \( \preceq_4 \):

\[
\text{locale } \text{gd-term} = \\
\text{gd-powerprod ord ord-strict} + \\
\text{ord-term lin: linorder ord ord-term-strict} \\
\text{ord-strict (infix "\prec") 50} \\
\text{ord-term (infix "\preceq") 50} \\
\text{ord-term-strict (infix "\prec\preceq") 50} + \\
\text{assumes stimes-mono: "v \preceq_4 w \Rightarrow t \odot v \preceq_4 t \odot w"} \\
\text{assumes ord-termI: "fst v \preceq_4 fst w \Rightarrow snd v \leq snd w \Rightarrow v \preceq_4 w"}
\]

So, \( \text{gd-term} \) extends \( \text{gd-powerprod} \) by \( \preceq_4 \) and \( \prec_4 \), requires \( \kappa \) to be well-ordered by
\( \preceq \), and requires \( \preceq_4 \) to be a linear ordering satisfying the two axioms \( \text{stimes-mono} \)
and \( \text{ord-termI} \). \( \odot \) is defined as

\[
\text{definition stimes :: "} \alpha \Rightarrow (\alpha \times \kappa) \Rightarrow (\alpha \times \kappa) \text{" (infixl "\odot") 75} \\
\text{where "stimes t v = (t \odot fst v, snd v)"}
\]

i.e. it multiplies the power-product of its second argument with its first argument.
\( \text{fst} \) and \( \text{snd} \) are built-in Isabelle/HOL functions that access the first and
second, respectively, component of a pair. In the formalization, we prove two
interpretations of the locale: one for \( \text{POT} \) orders, and one for \( \text{TOP} \) orders.

In the context of the locale, i.e. with \( \preceq_4 \) as an implicit parameter avail-
able, we can now immediately define leading terms, leading power-products and
leading coefficients, just as described in Section \(^2\). In addition, we also define

\text{unit} contains only one single element, so \( \alpha \times \text{unit} \) is isomorphic to \( \alpha \).
the tail of a polynomial \( p \), \( \text{tail}(p) \), as a copy of \( p \) where, however, the leading coefficient of \( p \) is set to 0, i.e., in which only terms less than \( \text{tt}(p) \) appear.

Based on \( \otimes \) we introduce multiplication of a vector-polynomial by a coefficient \( c : \beta \) and a power-product \( t : \alpha \) in the obvious way: all coefficients are multiplied by \( c \), and all terms are multiplied by \( t \) via \( \otimes \). The resulting function is called \texttt{monom.mult} and caters for the multiplications needed in reduction (cf. Definition \[1\]) and S-polynomial (cf. Definition \[3\]).

Before we finish this section, we introduce three more notions related to power-products of type \( \alpha \times \kappa \):

\begin{verbatim}
definition dvd_term :: "\( (\alpha \times \kappa) \Rightarrow (\alpha \times \kappa) \Rightarrow \text{bool} \) (infix "dvd\(_{\tau}\)" 50)
  where "dvd_term \( u \) \( v \) \( \leftrightarrow \) (snd \( u \) = snd \( v \) \land (fst \( u \) dvd (fst \( v \))))"
\end{verbatim}

which means that a term \( s c \_j \) divides another term \( t \_j \) iff \( i = j \) and \( s \) divides \( t \).

Furthermore, we extend the linear order \( \preceq_\tau \) to a partial order \( \preceq_\rho \) on polynomials, whose strict version \( \prec_\rho \) is defined as

\begin{verbatim}
definition ord.strict.p :: "\( (\alpha \times \kappa) \Rightarrow \beta \Rightarrow (\alpha \times \kappa) \Rightarrow \beta \Rightarrow \text{bool} \) (infix "\prec\rho\" 50)
  where "p \( \prec\rho \) q \( \leftrightarrow \) (\exists v. coeff p v = 0 \land coeff q v \neq 0 \land
                   (\forall u. u \prec p \_u \rightarrow coeff p u = coeff q u))"
\end{verbatim}

and prove that \( \prec\rho \) is well-founded on sets of polynomials in which only finitely many indeterminates appear:

\begin{verbatim}
lemma ord.p.minimum:
  assumes "dickson.grading d" and "x \in \emptyset" and "0 \subseteq \text{dgrad.p.set} d n"
  obtains q where "q \in \emptyset" and "\forall y. y \prec p q \rightarrow y \notin \emptyset"
\end{verbatim}

d\text{grad.p.set}(d, m) gives the set \( F \) of all polynomials such that the Dickson grading \( d \) only attains values below \( m \) when applied to power-products appearing in \( F \). Hence, informally \( Q \subseteq \text{dgrad.p.set}(d, m) \) expresses that at most \( m \) indeterminates appear in \( Q \).

Finally, we also introduce the notion of a \textit{submodule} generated by a set \( B \) of polynomials as the smallest set that contains \( B \cup \{0\} \) and that is closed under addition and under \texttt{monom.mult}:

\begin{verbatim}
inductive_set pmdl :: "\( (\alpha \times \kappa) \Rightarrow \beta \Rightarrow (\alpha \times \kappa) \Rightarrow \beta \Rightarrow \text{set} \) for \( B \) where
  pmdl 0: "0 \in \text{pmdl} B"
  pmdl.plus: "a \in \text{pmdl} B \Rightarrow b \in B \Rightarrow a + \text{monom.mult} c \_t b \_b \in \text{pmdl} B"
\end{verbatim}

Submodules generalize the concept of ideals in rings, which are sets that are closed under addition and under multiplication by arbitrary elements of the ring.

\textit{Remark} 4. Although in the theory of Gröbner bases polynomials need to have coefficients in fields, we formulated all definitions and lemmas about polynomials for as general coefficient types as possible, often requiring only the ubiquitous type-class constraint \texttt{zero} (which we omit in this paper for better readability).

\textit{Remark} 5. Developing the theory for modules and submodules is a nice generalization of rings and ideals, and as mentioned before it is always possible to instantiate \( \kappa \) by \texttt{unit} when one wishes to state definitions/theorems for scalar polynomials only. But still, all types would then involve the redundant artifact \texttt{unit}, which would clutter the theory quite a bit. As a remedy, we actually do not fix the type \( \alpha \times \kappa \) in the formalization, but instead use some fresh type variable \( \nu \) which we only demand to be \textit{isomorphic} to \( \alpha \times \kappa \) via the morphisms
pair_of_term and term_of_pair (this happens once more in a locale). So, \( \nu \) can be instantiated by \( \alpha \) itself if \( \kappa \) is instantiated by unit, eventually yielding the ordinary polynomial ring \( \alpha \Rightarrow_0 \beta \).

4 Gröbner Bases and Buchberger’s Algorithm

From now on we tacitly assume, unless stated otherwise, that all definitions and theorems are stated in context gd_term (meaning that all parameters and axioms of gd_term are available for use, that \( \kappa \) belongs to type class wellorder, and that \( \alpha \) belongs to type class graded_dickson_powerprod), and that the type \( \beta \) of coefficients belongs to type class field.

4.1 Polynomial Reduction

Polynomial reduction is defined analogously to Definition 1:

definition red_single::
\[
((\alpha \times \kappa) \Rightarrow_0 \beta) \Rightarrow ((\alpha \times \kappa) \Rightarrow_0 \beta) \Rightarrow (\alpha \Rightarrow \text{bool})
\]
where
\[
\text{red_single } p \ q \ t \leftarrow (f \neq 0 \wedge \text{coeff } p \ (t \star lt \ f) \neq 0 \wedge q = p \ast \text{monom_mult } ((\text{coeff } p \ (t \star lt \ f)) / \text{lc } f) \ t \ f)
\]
definition red ::
\[
((\alpha \times \kappa) \Rightarrow_0 \beta) \text{set } \Rightarrow ((\alpha \times \kappa) \Rightarrow_0 \beta) \Rightarrow (\alpha \Rightarrow \text{bool})
\]
where
\[
\text{red } F \ p \ q \leftarrow (\exists f \in F. \exists t. \text{red_single } p \ q \ f \ t)
\]
definition is_red ::
\[
((\alpha \times \kappa) \Rightarrow_0 \beta) \text{set } \Rightarrow ((\alpha \times \kappa) \Rightarrow_0 \beta) \Rightarrow \text{bool}
\]
where
\[
\text{is_red } F \ a \leftarrow (\exists q. \text{red } F \ p \ q)
\]

\text{red_single}(p, q, f, t) expresses that polynomial \( p \) reduces to \( q \) modulo the individual polynomial \( f \), multiplying \( f \) by power-product \( t \). Likewise, \( \text{red}(F)(p, q) \) expresses that \( p \) reduces to \( q \) modulo the set \( F \) in one step; hence, \( \text{red}(F) \) is the actual reduction relation modulo \( F \), and \( (\text{red}(F))^* \) denotes its reflexive-transitive closure. In in-line formulas we will use the conventional infix notations \( p \rightarrow_F q \) and \( p \rightarrow^*_F q \) instead of the more clumsy \( \text{red}(F)(p, q) \) and \( (\text{red}(F))^*(p, q) \), respectively. \( \text{is_red}(F)(p) \), finally, expresses that \( p \) is reducible modulo \( F \).

After introducing the above notions, we are able to prove, for instance,

\text{lemma red.ord:} "\( \text{red } F \ p \ q \Rightarrow \Rightarrow_0 p \)"

which immediately implies that \( \rightarrow_F \) is well-founded. This justifies implementing a function \( \text{trd} \), which totally reduces a given vector-polynomial \( p \) modulo a finite list \( fs \) of polynomials and, thus, computes a normal form of \( p \) modulo \( fs \). Operationally, \( \text{trd}(fs, p) \) iterates over the terms appearing in the polynomial \( p \) in order (starting with the greatest one) and tries to reduce them modulo the polynomials in the list \( fs \); for each term, the first suitable \( f \in fs \) is taken (if any). After implementing \( \text{trd} \) in said way, it is possible to derive the following characteristic properties:

\text{lemma trd.red.rtrancl:} "(\text{red } (\text{set } fs))^* p (\text{trd } fs \ p)"
\text{lemma trd.irred:} "\neg \text{is_red } (\text{set } fs) (\text{trd } fs \ p)"

So, \( \text{trd} \) really computes some normal form of the given polynomial modulo the given list of polynomials. But recall from Section 2 that normal forms are in general not unique, i.e. the reduction relation modulo an arbitrary set \( F \) is in general not confluent.

\(^6\)Abusing notation, \( x \in xs \), for a list \( xs \), means that \( x \) is an element of \( xs \).
4.2 Gröbner Bases

The fact that $\rightarrow_F$ is not confluent for all $F$ motivates the definition of a Gröbner basis as a set that induces a confluent reduction relation:

\begin{verbatim}
definition is_Groebner_basis :: "((α × κ) ⇒₀ β) set ⇒ bool" where "is_Groebner_basis F ←→ is_confluent (red F)"
\end{verbatim}

where is_confluent is the predicate-analogue of CR from Abstract-Rewriting [31].

Before we are able to state and prove Theorem 4, we need S-polynomials:

\begin{verbatim}
definition spoly :: "((α × κ) ⇒₀ β) ⇒ ((α × κ) ⇒₀ β) ⇒ ((α × κ) ⇒₀ β)" where "spoly p q = (if snd (lt p) = snd (lt q) then let l = lcm (lp p) (lp q) in (monom_mult (1 / (lc p)) (l / (lp p)) p) - (monom_mult (1 / (lc q)) (l / (lp q)) q) else 0)"
\end{verbatim}

Theorem 4 states that a set $F$ is a Gröbner basis if the S-polynomials of all pairs of elements in $F$ can be reduced to 0 modulo $F$, i.e.

\begin{verbatim}
theorem Buchberger_thm_finite:
assumes "finite F"
assumes "∀ p q. p ∈ F =⇒ q ∈ F =⇒ (red F) ∗∗ (spoly p q) 0"
shows "is_Groebner_basis F"
\end{verbatim}

The finiteness constraint on $F$ could be weakened to an assumption involving Dickson gradings and dgrad_p_set, just as in ord-p-minimum. Our proof of Theorem Buchberger-thm-finite exploits various results about well-founded binary relations formalized in [31]. Thanks to that theorem, for deciding whether a finite set is a Gröbner basis it suffices to compute normal forms of finitely many S-polynomials and check whether they are 0. In fact, also the converse of Buchberger-thm-finite holds (quite trivially), so whenever one finds a non-zero normal form of an S-polynomial, the given set cannot be a Gröbner basis.

Another alternative characterization of Gröbner bases proved in the formalization is based on the divisibility of leading terms; this characterization is particularly useful for establishing the correctness of the algorithm for computing reduced Gröbner bases in Section 6:

\begin{verbatim}
lemma GB_alt_3_finite:
assumes "finite F"
shows "is_Groebner_basis F ←→ (∀ p ∈ pmdl F. p ≠ 0 −→ (∃ f ∈ F. f ≠ 0 ∧ lt f dvd lt p))"
\end{verbatim}

4.3 An Algorithm Schema for Computing Gröbner Bases

Theorem Buchberger-thm-finite not only yields an algorithm for deciding whether a given finite set $F$ is a Gröbner basis or not, but also an algorithm for completing $F$ to a Gröbner basis in case it is not. This algorithm, called Buchberger’s algorithm, is a classical critical-pair/completion algorithm that repeatedly checks whether all S-polynomials reduce to 0, and if not, adds their non-zero normal forms to the basis to make them reducible to 0; the new elements that are added to the basis obviously do not change the submodule generated by the basis, since reduction preserves submodule membership.

In our formalization, we do not directly implement Buchberger’s algorithm, but instead consider a more general algorithm schema first, of which both Buchberger’s algorithm and Faugère’s $F_4$ algorithm (cf. Section 5) are particular
instances. This algorithm schema is called \texttt{gb.schema.aux} and implemented by the following tail-recursive function:

\begin{verbatim}
function gb.schema.aux :: "((α × κ) ⇒ 0 β) list ⇒ 
    (((α × κ) ⇒ 0 β) × ((α × κ) ⇒ 0 β)) list ⇒ 
    ((α × κ) ⇒ 0 β) list" where

"gb.schema.aux bs ps = 
  (if ps = [] ∨ gen_whole_module bs then 
   bs 
   else 
    (let sps = sel bs ps; ps0 = ps -- sps; hs = compl bs ps0 sps in 
     gb_schema_aux (bs @ hs) (add_pairs bs ps0 hs)))"
\end{verbatim}

The first argument of \texttt{gb.schema.aux}, \texttt{bs}, is the so-far computed basis, and the second argument \texttt{ps} is the list of all pairs of polynomials from \texttt{bs} whose S-polynomials might not yet reduce to 0 modulo \texttt{bs}. Hence, as soon as \texttt{ps} is empty all S-polynomials reduce to 0, and by virtue of Theorem \texttt{Buchberger-thm-finite} the list \texttt{bs} constitutes a Gröbner basis. Moreover, if the current basis is detected to generate the whole module then it can as well be returned immediately without any further ado. \texttt{gen_whole_module} basically checks whether for each of the finitely many component-indices \texttt{i} appearing in \texttt{bs} there is a non-zero polynomial \texttt{b} in \texttt{bs} such that the component-index of \texttt{lt}(\texttt{b}) is \texttt{i} and \texttt{lp}(\texttt{b}) = 1. This parallels the scalar case, where a set of polynomials is known to generate the whole ring if it contains a non-zero constant polynomial.

The auxiliary function \texttt{add_pairs}, when applied to arguments \texttt{bs}, \texttt{ps0} and \texttt{hs}, returns a new list of pairs of polynomials which contains precisely (i) all pairs from \texttt{ps0}, (ii) the pair \texttt{(h,b)} for all \texttt{h ∈ hs} and \texttt{b ∈ bs}, and (iii) one of the pairs \texttt{(h1,h2)} or \texttt{(h2,h1)} for all \texttt{h1, h2 ∈ hs} with \texttt{h1} \neq \texttt{h2}. The auxiliary function \texttt{diff_list (infix "--")} is the analogue of set-difference for lists, i.e. it removes all occurrences of all elements of its second argument from its first argument. \texttt{append, infix @}, concatenates two lists.

The two functions \texttt{sel} and \texttt{compl} are additional parameters of the algorithm; they are not listed among the arguments of \texttt{gb.schema.aux} here merely for the sake of better readability. Informally, they are expected to behave as follows:

- If \texttt{ps} is non-empty, \texttt{sel(bs,ps)} should return a non-empty sublist \texttt{sps} of \texttt{ps}.
- \texttt{compl(bs,ps,sps)} should return a (possibly empty) list \texttt{hs} of polynomials such that (i) \texttt{0 ∉ hs}, (ii) \texttt{hs ⊆ pmdl(bs)}, (iii) \texttt{spoly(p,q) →_bs∪hs 0} for all \texttt{(p,q) ∈ sps}, and (iv) \texttt{¬lt(b) dvd', lt(h) for all b ∈ bs and h ∈ hs}.

Typically, concrete instances of \texttt{sel} do not take \texttt{bs} into account, but in any case it does not harm to pass it as an additional argument. Any instances of the two parameters that satisfy the above requirements lead to a partially correct procedure for computing Gröbner bases, since \texttt{compl} takes care that all S-polynomials of the selected pairs \texttt{sps} reduce to 0. However, the procedure is not only partially correct, but also terminates for every input; the argument roughly proceeds as follows:

- Assume the procedure did not terminate. Then, infinitely many non-zero polynomials \texttt{h} (originating from \texttt{compl}) are added to the basis \texttt{bs}.

\footnote{The “whole module” in this context corresponds to $K[x_0,\ldots,x_n]^k$, where $x_0,\ldots,x_n$ are the indeterminates and $k$ is the largest component-index appearing in \texttt{bs}.}
• The leading term of each of these polynomials is not divisible (w.r.t. dvd) by the leading term of any polynomial in the current basis.

• Therefore, the sequence of these polynomials violates the Dickson property of sequences of power-products – a contradiction. This argument also works if \( \kappa \) is instantiated by \( \text{nat} \) and \( \alpha \) by \( \text{nat} \Rightarrow \text{nat} \Rightarrow \text{0} \), because the sets of indeterminates and non-zero components appearing in the current basis \( bs \) in recursive calls of \( \text{gb.schema.aux} \) are finite and uniformly bounded.

Function \( \text{gb.schema} \), finally, calls \( \text{gb.schema.aux} \) with the right initial values:

\[
\text{definition gb.schema :: } \text{"}((\alpha \times \kappa) \Rightarrow \beta) \text{ list} \Rightarrow ((\alpha \times \kappa) \Rightarrow \beta) \text{ list}\text{"}
\]

where "\( \text{gb.schema bs = gb.schema.aux bs (add.pairs [()] [()] bs)} \)"

Remark 6. Not only Buchberger’s algorithm and \( F_4 \) are instances of \( \text{gb.schema} \), as briefly indicated above and discussed in more detail in Sections 4.4 and 5, but also Faugère’s \( F_5 \) algorithm [14] is an instance of it. \( F_5 \) is currently the most efficient method for computing Gröbner bases, but formalizing it in a proof assistant is a challenging task that is yet to be accomplished. \( F_5 \) computes Gröbner bases incrementally, i.e. for computing a Gröbner basis of \( m \) polynomials it calls itself recursively on the first \( m - 1 \) polynomials and then adds the \( m \)-th polynomial. \( \text{gb.schema} \) can handle such incremental computations as well, although this is not reflected in its (simplified) presentation in this paper.

4.4 Buchberger’s Algorithm

The function implementing the usual Buchberger algorithm, called \( \text{gb} \), can immediately be obtained from \( \text{gb.schema} \) by instantiating

• \( \text{sel} \) to a function that selects a single pair, i.e. returns a singleton list, and

• \( \text{compl} \) to a function that totally reduces \( \text{spoly}(p, q) \) to some normal form \( h \) using \( \text{trd} \), where \( (p, q) \) is the pair selected by the instance of \( \text{sel} \), and returns the singleton list \([h]\) if \( h \neq 0 \) and the empty list otherwise.

These instances of \( \text{sel} \) and \( \text{compl} \) can easily be proved to meet the requirements listed above, so we can finally conclude that \( \text{gb} \) indeed always computes a Gröbner basis of the submodule generated by its input:

\[
\text{theorem gb.isGB: is.Groebner.basis (set (gb bs))}
\]

\[
\text{theorem gb.pmdl: pmdl (set (gb bs)) = pmdl (set bs)}
\]

Gröbner bases have many interesting properties. One of them was briefly sketched at the end of Section 2: if \( G \) is a Gröbner basis, then a polynomial \( p \) is in the submodule generated by \( G \) iff the unique normal form of \( p \) modulo \( G \) is 0. Together with the two previous theorems this observation leads to an effective answer to the membership problem for submodules represented by finite lists of generators:

\[
\text{theorem in.pmdl.gb: } "p \in \text{pmdl (set bs)} \longleftrightarrow \{ \text{trd (gb bs) p} = 0 \}"
\]
4.5 Improving Efficiency: Buchberger’s Criteria

The key ingredient of \texttt{gb\_schema\_aux} is parameter \texttt{compl}, since it is precisely this function that has to ensure that all S-polynomials can be reduced to 0 modulo the enlarged list \texttt{bs\@hs}. However, computing the new polynomials \texttt{hs} can be very time-consuming, since it is usually accomplished by reducing the S-polynomials to some normal form, either employing \texttt{trd} or some other methodology (as in \texttt{F4}). Now, if the S-polynomial of some pair \((p, q)\) can be reduced to 0 modulo the current basis \texttt{bs} already, and this fact can somehow be predicted without actually doing the reduction, the whole expensive normal-form computation of \texttt{spoly}(p, q) could be avoided altogether; in this case, \((p, q)\) is called a useless pair. Therefore, one approach to improve the efficiency of \texttt{gb\_schema\_aux} is to detect as many useless pairs as possible and to immediately discard them without passing them along to the normal-form computation. Since all this happens in \texttt{compl}, and \texttt{compl} is a parameter of \texttt{gb\_schema\_aux}, many different strategies for detecting useless pairs can be implemented easily, without having to change the overall implementation of the algorithm schema.

In our formalization, the two instantiations of \texttt{compl} yielding Buchberger’s algorithm and the \texttt{F4} algorithm (cf. Section 5), respectively, incorporate two standard criteria, originally due to Buchberger, for detecting useless pairs: the product criterion (which is only applicable in the scalar case with \(\kappa = \text{unit}\)) and the chain criterion. In a nutshell, the product criterion states that if \(\gcd(lp(p), lp(q)) = 1\), then \(\texttt{spoly}(p, q) \rightarrow^* \{p, q\} 0\). The chain criterion states that if there is some \(r\) in the current basis \texttt{bs} satisfying (i) \(lp(r) \mid \text{lcm}(lp(p), lp(q))\), (ii) \(\texttt{spoly}(p, r) \rightarrow^{bs} 0\) and (iii) \(\texttt{spoly}(r, q) \rightarrow^{bs} 0\), then also \(\texttt{spoly}(p, q) \rightarrow^{bs} 0\), and hence the pair \((p, q)\) can be discarded. A more thorough account on Buchberger’s criteria can be found in [7]. In the formalization we of course prove that these two criteria are indeed correct, in the sense that they only discard useless pairs. It must be noted, though, that in general neither of them detects all useless pairs.

Remark 7. When testing the chain criterion on concrete input, lots of equality-checks between polynomials have to be performed. Because of that, \texttt{gb\_schema} automatically assigns a unique ID (of type \texttt{nat}) to every polynomial, allowing the chain criterion to only compare IDs rather than full polynomials. This small trick helped to significantly increase the efficiency of the implementation.

5 Faugère’s \texttt{F4} Algorithm

In Buchberger’s algorithm, in each iteration precisely one S-polynomial is reduced modulo the current basis, giving rise to at most one new basis element. However, as J.-C. Faugère observed in [13], it is possible to reduce several S-polynomials simultaneously with a considerable gain of efficiency (especially for large input). To that end, one selects some pairs from the list \texttt{ps}, reduces them modulo the current basis, and adds the resulting non-zero normal forms to the basis; in short, several iterations of the usual Buchberger algorithm are combined into one single iteration. This new algorithm is called \texttt{F4}.

The crucial idea behind \texttt{F4}, and the reason why it can be much faster than Buchberger’s algorithm, is the clever implementation of simultaneous reduction by computing the reduced row echelon form of certain coefficient matrices. Be-
fore we can explain how this works, we need a couple of definitions; let \( fs \) always be a list of vector-polynomials of length \( m \), and \( vs \) be a list of terms of length \( \ell \).

- **Keys_to_list** \((fs)\) returns the list of all distinct terms appearing in \( fs \), sorted descending w.r.t. \( \preceq \).
- **polys_to_mat** \((vs, fs)\) returns a matrix \( A \) (à la \([34]\)) of dimension \( m \times \ell \), satisfying \( A_{i,j} = \text{coeff}(fs_i, vs_j) \), for \( 0 \leq i < m \) and \( 0 \leq j < \ell \).
- **mat_to_polys** \((vs, A)\) is the “inverse” of **polys_to_mat**, i.e. if \( A \) is a matrix of dimension \( m \times \ell \) it returns the list \( gs \) of polynomials satisfying \( \text{coeff}(gs_i, vs_j) = A_{i,j} \) and \( \text{coeff}(gs_i, v) = 0 \) for all other terms \( v \) not contained in \( vs \).
- **row_echelon** \((A)\) returns the reduced row echelon form of matrix \( A \); it is defined in terms of **gauss_jordan** from \([34]\).

With these auxiliary functions at our disposal we can now give the formal definitions of two concepts whose importance will become clear below:

**definition** Macaulay_mat :: 
"\((α × κ) ⇒ β\) list ⇒ β :: field mat"

where "Macaulay_mat fs = polys_to_mat (Keys_to_list fs) fs"

**definition** Macaulay_red :: 
"\((α × κ) ⇒ β\) list ⇒ ((α × κ) ⇒ β :: field) list"

where "Macaulay_red fs = 
(let lts = map lt (filter (λf. f \(̸\) 0) fs) in 
filter (λf. f \(̸\) 0 ∧ lt f \(\notin\) set lts) 
(mat_to_polys (Keys_to_list fs) (row_echelon (Macaulay_mat fs))))"

Macaulay_mat \((fs)\) is called the **Macaulay matrix** of \( fs \). Macaulay_red \((fs)\) constructs the Macaulay matrix of \( fs \), transforms it into reduced row echelon form, and converts the resulting matrix back to a list of polynomials, from which it filters out those non-zero polynomials whose leading terms are not among the leading terms of the original list \( fs \).

**Example 7.** Let \( fs \) be the list \( fs = [x^3_1 - 5x^2_1 x_1 - 2, -4x^2_1 + 2x^2_1 + x^2_0 x_1, 2x^3_1 - x^2_0 + 1] \) of scalar polynomials, and let \( \preceq \) and \( \preceq \) be the lexicographic order with \( x_0 \prec x_1 \). The sorted list of terms (or, in this case, power-products) appearing in \( fs \) is \([x^3_1, x_0^2, x_0 x_1, x_1, x_0, 1]\). Hence, the Macaulay matrix of \( fs \) is

\[
\begin{pmatrix}
x^3_1 & x^2_1 & x^2_0 x_1 & x_0 & 1 \\
1 & 0 & -5 & 0 & -2 \\
-4 & 2 & 1 & 0 & 0 \\
2 & -1 & 0 & -1 & 4
\end{pmatrix}
\]

Row-reducing the Macaulay matrix yields

\[
\begin{pmatrix}
x^3_1 & x^2_1 & x^2_0 x_1 & x_0 & 1 \\
1 & 0 & 0 & -10 & 38 \\
0 & 1 & 0 & -19 & 72 \\
0 & 0 & 1 & -2 & 8
\end{pmatrix}
\]

from which we can extract the three polynomials \( h_1 = x^3_1 - 10x_0 + 38, h_2 = x^2_1 - 19x_0 + 72 \) and \( h_3 = x^2_0 x_1 - 2x_0 + 8 \). The leading term of \( h_1 \) is \( x^3_1 \), which is

\[\text{We use 0-based indexing of lists, vectors and matrices, just as Isabelle/HOL.}\]
also the leading term of one of the three (actually all three) original polynomials. So, \texttt{Macaulay\_red}(fs) returns the two-element list \([h_2, h_3]\).

\texttt{Macaulay\_red} is the key ingredient of the \(F_4\) algorithm, because the list it returns is precisely the list \(hs\) that must be added to \(bs\) in an iteration of \texttt{gb\_schema\_aux}. The only question that still remains open is which argument list \(fs\) it needs to be applied to; this question is answered by an algorithm called symbolic preprocessing, implemented by function \texttt{sym\_preproc} in our formalization. \texttt{sym\_preproc} takes two arguments, namely the current basis \(bs\) and the list of selected pairs \(sps\), and informally behaves as follows:

1. For each \((p, q) \in sps\) compute the two polynomials

   \[\text{monom\_mult}(1/\text{lcm}(lp(p), lp(q))/lp(p), p)\]

   \[\text{monom\_mult}(1/\text{lcm}(lp(p), lp(q))/lp(p), p)\]

   whose difference is precisely \(\text{spoly}(p, q)\) (unless \(\text{spoly}(p, q)\) is 0). Collect all these polynomials in an auxiliary list \(fs'\).

2. Collect all monomial multiples of each \(b \in bs\) that are needed to totally reduce the elements of \(fs'\) in a list \(fs''\). That means, for all \(b, f, g, h\) with \(b \in bs, f \in fs', f \rightarrow_{t}^b g\) and \(\text{red\_single}(g, h, b, t)\), the monomial multiple \(\text{monom\_mult}(1, t, b)\) of \(b\) must be included in \(fs''\).

3. Return the concatenation \(fs' @ fs''\) of \(fs'\) and \(fs''\).

The interesting part of symbolic preprocessing is Step 2, which can be accomplished without actually carrying out the reductions. In our formalization, it is implemented by the two functions \texttt{sym\_preproc\_addnew} and \texttt{sym\_preproc\_aux}, defined as

\begin{verbatim}
primrec sym.preproc.addnew :: "((α × κ) ⇒ β) list ⇒ (α × κ) list ⇒
((α × κ) list ⇒ (α × κ) ⇒ ((α × κ) ⇒ β) list × ((α × κ) ⇒ β) list)" where
  "sym.preproc.addnew [] vs fs _ = (vs, fs)"
  "sym.preproc.addnew (b # bs) vs fs v =
    if (lt b) dvd v then
      let f = monom.mult 1 (fst v / (lp b)) b in
      sym.preproc.addnew bs (merge.wrt (op ≻ v) vs (keys.to.list (tail f)))
      (insert.list f fs) v
    else
      sym.preproc.addnew bs vs fs v)"

function sym.preproc.aux :: "((α × κ) ⇒ β) list ⇒
((α × κ) list × ((α × κ) ⇒ β) list) ⇒
((α × κ) ⇒ β) list" where
  "sym.preproc.aux bs (vs, fs) =
    if vs = [] then
      fs
    else
      let v = Max (set vs); vs' = removeAll v vs in
      sym.preproc.aux bs (sym.preproc.addnew bs vs' fs v)"
\end{verbatim}

\texttt{sym\_preproc} calls \texttt{sym\_preproc\_aux} with the current basis \(bs\) and the pair \((vs, fs')\), where \(fs'\) is the result of Step 1 and \(vs\) is the sorted list of terms appearing in \(fs'\). A more detailed account on symbolic preprocessing can be found in [13].

Putting everything together, the function \texttt{f4\_red} is obtained as
definition f4.red::"((α × κ) ⇒_0 β :: field) list ⇒
((α × κ) ⇒_0 β × (α × κ) ⇒_0 β) list ⇒ ((α × κ) ⇒_0 β) list"
where "f4.red bs sps = Macaulay_red (sym_preproc bs sps)"

and proved to be a feasible instance of parameter \textit{compl} in particular, the leading terms of the polynomials in \(hs = f4\textunderscore red(bs, sps)\) are not divisible by the leading terms of the polynomials in \(bs\), and indeed all S-polynomials originating from pairs in \(sps\) are reducible to 0 modulo the enlarged basis \(bs@hs\):

\begin{verbatim}
lemma f4_red_not_dvd:
assumes "h ∈ set (f4.red bs sps)" and "b ∈ set bs" and "b ≠ 0"
shows "¬ lt b dvd (lt h)"
\end{verbatim}

\begin{verbatim}
lemma f4_red_spoly_reducible:
assumes "set sps ⊆ set bs × set bs" and "(p, q) ∈ set sps"
shows "red (set (bs @ (f4_red bs sps))))" (spoly p q) 0"
\end{verbatim}

Eventually, the resulting instance of \textit{gb\_schema} which implements Faugère’s \(F_4\) algorithm is called \textit{f4}.

Summarizing, the simultaneous reduction of several S-polynomials boils down to the computation of the reduced row echelon form of Macaulay matrices over the coefficient field \(K\). These matrices are typically very big, very rectangular (i.e. have much more columns than rows) and extremely sparse. Therefore, if \(F_4\) is to outperform Buchberger’s algorithm, such matrices must be stored efficiently (possibly even involving some sort of compression), and the computation of reduced row echelon forms must be highly optimized; we again refer to [13] for more information. Furthermore, the superiority of \(F_4\) over Buchberger’s original algorithm only takes effect when the problem instances are sufficiently large as to outweigh the overhead stemming from all the matrix constructions in \(F_4\); for small problems (\(\leq 5\) indeterminates, moderate degrees), Buchberger’s algorithm is typically still faster.

6 Reduced Gröbner Bases

As mentioned in Section 2, Gröbner bases are not unique, even if the term order \(\succeq_1\) is fixed. One can, however, impose stronger constraints on generating sets of submodules than giving rise to a confluent reduction relation, which do ensure uniqueness; the resulting concept is that of \textit{reduced Gröbner bases}.

The central idea behind reduced Gröbner bases is \textit{auto-reducedness}: a set \(B\) of polynomials is auto-reduced iff no \(b ∈ B\) can be reduced modulo \(B\setminus\{b\}\). A reduced Gröbner basis, then, is simply an auto-reduced Gröbner basis of non-zero monic\footnote{A polynomial is called \textit{monic} iff its leading coefficient is 1.} polynomials:

\begin{verbatim}
definition is_reduced_GB :: "((α × κ) ⇒_0 β :: bool) list ⇒ bool"
where "is_reduced_GB B ⇐ is_Groebner_basis B ∧ is_auto_reduced B ∧
is_monic_set B ∧ B ∉ B"
\end{verbatim}

After having defined reduced Gröbner bases as above, one can prove with moderate effort that, upon existence, they are indeed unique for every submodule (of course only modulo the implicitly fixed ordering \(\succeq_1\)):
Besides uniqueness, one can furthermore also prove existence of reduced Gröbner bases. The proof we give in the formalization is even constructive, in the sense that we formulate an algorithm which auto-reduces and makes monic a given set (or, more precisely, list) of polynomials, and, therefore, when applied to some Gröbner basis returns the reduced Gröbner basis of the submodule it generates. Said algorithm proceeds in three steps:

1. First, all polynomials of the input list whose leading terms are divisible by the leading term of any other polynomial in the input, are removed, and so are all occurrences of 0.

2. Next, every remaining polynomial is totally reduced modulo the others (employing function \texttt{trd}) and replaced by the result of this process.

3. Finally, the polynomials are made monic by dividing through their respective leading coefficients.

The function combining these three steps is called \texttt{comp_red_monic_basis} and possesses the following two key properties:

\begin{itemize}
  \item \textbf{lemma} \texttt{comp_red_monic_basis.is_reduced_GB}:
    \begin{itemize}
      \item \texttt{assumes} "is_Groebner_basis (set bs)"
      \item \texttt{shows} "is_reduced_GB (set (comp_red_monic_basis bs))"
    \end{itemize}
  \item \textbf{lemma} \texttt{comp_red_monic_basis.pmdl}:
    \begin{itemize}
      \item \texttt{assumes} "is_Groebner_basis (set bs)"
      \item \texttt{shows} "pmdl (set (comp_red_monic_basis bs)) = pmdl (set bs)"
    \end{itemize}
\end{itemize}

So, by combining functions \texttt{gb} (or \texttt{f4}) and \texttt{comp_red_monic_basis}, we obtain a certified function for computing reduced Gröbner bases from any given list of polynomials, and can moreover conclude that every finitely-generated submodule of $K[X]^k$ has a unique reduced Gröbner basis:

\begin{itemize}
  \item \textbf{theorem} \texttt{exists_unique.reduced_GB.finite}:
    \begin{itemize}
      \item \texttt{assumes} "finite F"
      \item \texttt{shows} "∃! G. is_reduced_GB G ∧ pmdl G = pmdl F"
    \end{itemize}
\end{itemize}

\textbf{Example 8.} The Gröbner basis $\tilde{F} = \{x_1^2, x_0x_1 + x_0^2, x_0^3\}$ computed in Example 4 is already the reduced Gröbner basis of the ideal it generates. On the other hand, $G = \{x_0^2 - 2x_0x_1 + 1, x_1 - x_0\}$ is a Gröbner basis (w. r. t. the degree-lexicographic ordering with $x_0 < x_1$) but no reduced Gröbner basis, because the first element is reducible modulo the second one.

Remark 8. Auto-reduction can already be applied during the computation of Gröbner bases and may lead to a significant speed-up of the algorithm. Incorporating auto-reduction into \texttt{gb_schema} is possible future work.

\section{Gröbner Bases of Syzygy Modules}

Given a list $bs = [b_0, \ldots, b_{m-1}]$ of $m$ vector-polynomials, one could ask oneself what the polynomial relations among the elements of $bs$ are. In other words,
one wants to find all $m$-component vectors $s = (s_0, \ldots, s_{m-1})^T \in K[X]^m$ such that
\[ \sum_{i=0}^{m-1} s_i b_i = 0. \]

In the literature, such a vector $s$ of polynomials is called a syzygy \footnote{In the formalization the type of the polynomials is $(\chi \Rightarrow_{\text{nat}} \Rightarrow_{\text{nat}} \beta)$, because fixing $\kappa$ to nat turns out to be convenient here.} of $bs$, and as one can easily see the set of all syzygies of a list $bs$ forms a submodule of $K[X]^m$.

**Example 9.** If $m = 2$ and $b_0, b_1 \in K[X] \setminus \{0\}$, then a non-trivial syzygy is obviously given by \((b_1, -b_0)^T\), because $b_1 b_0 + (-b_0) b_1 = 0$. More generally, each list of scalar polynomials with at least two non-zero elements admits non-trivial syzygies of the above kind.

As it turns out, it is not difficult to compute Gröbner bases of syzygy modules. We briefly outline how it works in theory; so, assume that $bs$ is the $m$-element list $[b_0, \ldots, b_{m-1}]$ of polynomials in $K[X]^T$.

1. Add further components to the $b_i$, for all $0 \leq i < m$, such that $b_i$ becomes $(0, \ldots, 0, 1, 0, \ldots, 0, b_i)^T$, where the 1 occurs precisely in the $i$-th component. Note: these vectors have $\geq m + 1$ components, since after the newly introduced $m$ components come all of $b_i$’s existing components. Call the resulting list $bs'$.

2. Compute a Gröbner basis $gs$ of $bs'$ w.r.t. a POT-extension $\preceq_t$ of some admissible order $\leq$ on power-products.

3. From $gs$ extract those elements of the form $(s_0, \ldots, s_{m-1}, 0, \ldots, 0)^T$, and restrict them to $m$-dimensional vectors $(s_0, \ldots, s_{m-1})^T$. These vectors constitute a Gröbner basis w.r.t. $\preceq_t$ of the syzygy module of $bs$.

**Example 10.** Consider the three scalar polynomials $b_0 = x_0 x_1 - x_2$, $b_1 = x_0 x_2 - x_1$ and $b_2 = x_1 x_2 - x_0$, and the list $bs = [b_0, b_1, b_2]$. According to Step 1 we construct $bs'$ as

$$bs' = \left[ \begin{array}{c} 1 \\ 0 \\ 0 \\ b_0 \\ 0 \\ b_1 \\ 0 \\ b_2 \end{array} \right], \left[ \begin{array}{c} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{array} \right], \left[ \begin{array}{c} 0 \\ 0 \\ -b_1 \\ b_0 \\ 0 \\ 0 \\ b_0 \\ b_1 \end{array} \right], \left[ \begin{array}{c} 0 \\ 0 \\ -b_2 \\ 0 \\ 0 \\ -b_2 \\ 0 \\ 0 \end{array} \right].$$

Next, we compute a (non-reduced) Gröbner basis $gs$ of $bs'$ w.r.t. the POT-extension of the degree-reverse-lexicographic order with $x_0 < x_1 < x_2$:

$$gs = \left[ \begin{array}{c} 1 \\ 0 \\ 0 \\ b_0 \\ 0 \\ b_1 \\ x_2 \\ -x_1 \\ x_0 \\ x_0^2 - x_2 \end{array} \right], \left[ \begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ x_2 \\ x_1 - x_1 x_2^2 \\ x_1 - x_1 x_2^2 \\ x_1 - x_1 x_2^2 \\ x_1 - x_1 x_2^2 \\ x_1 - x_1 x_2^2 \\ x_1 - x_1 x_2^2 \end{array} \right], \left[ \begin{array}{c} 0 \\ 0 \\ 0 \\ -x_1 \\ -x_1 \\ -x_1 x_1 \\ -x_0 x_1 \\ -x_0 x_1 \\ -x_0 x_1 \\ -x_0 x_1 \end{array} \right].$$
So, according to Step 3 the four-element list

\[
\text{syz} = \left\{ \begin{array}{c}
\begin{pmatrix}
- b_1 \\
0
\end{pmatrix},
\begin{pmatrix}
- b_2 \\
0
\end{pmatrix},
\begin{pmatrix}
0 \\
- b_2
\end{pmatrix},
\begin{pmatrix}
x_1 - x_1 x_2^2 \\
x_1^2 x_2 - x_2^2
\end{pmatrix}
\end{array} \right\}
\]

constitutes a Gröbner basis of the syzygy module of \(bs\).

In the formalization, function \texttt{init\_syzygy\_list} takes care of the first step. The second step can of course be accomplished by any function for computing Gröbner bases, like \texttt{gb} or \texttt{f4}; there is nothing special about it concerning syzygies. The last step, finally, is implemented by function \texttt{filter\_syzygy\_basis}:

\begin{definition}
\texttt{filter\_syzygy\_basis} :: "nat ⇒ (α × nat) ⇒ α \texttt{β list} ⇒ (α × nat) ⇒ α \texttt{β list}"
\text{where} "filter\_syzygy\_basis \texttt{m gs} = \texttt{gs} \setminus \texttt{keys g ⊆ {0..<m}}"
\end{definition}

The correctness of the three-step algorithm sketched above is established by the following two lemmas:

\begin{lemma}
\texttt{pmdl\_filter\_syzygy\_basis}:
\text{assumes} "distinct \texttt{bs}" and "\texttt{is\_Groebner\_basis (set \texttt{gs})}"
\text{shows} "\texttt{pmdl (set (filter\_syzygy\_basis (length \texttt{bs}) \texttt{gs}))} = \texttt{syzygy\_module\_list \texttt{bs}}"
\end{lemma}

\begin{lemma}
\texttt{filter\_syzygy\_basis\_isGB}:
\text{assumes} "distinct \texttt{bs}" and "\texttt{is\_Groebner\_basis (set \texttt{gs})}"
\text{shows} "\texttt{is\_Groebner\_basis (set (filter\_syzygy\_basis (length \texttt{bs}) \texttt{gs}))}"
\end{lemma}

\texttt{syzygy\_module\_list} is an auxiliary constant that gives the set of syzygies of its argument. The first lemma states that the result of the algorithm indeed generates the syzygy module of the input list \(bs\), and the second lemma states that the result is even a Gröbner basis. Neither of the lemmas is confined to a particular function for computing the Gröbner basis \(gs\).

Besides a Gröbner basis of the syzygy module of \(bs\), the Gröbner basis \(gs\) computed in Step 2 carries further useful information:

- Projecting \(gs\) onto the last component(s), i.e. removing those components that were added in Step 1, yields a Gröbner basis of the original list \(bs\).
- Each element in \(gs\) possesses the property that its first \(m\) components are the cofactors, w.r.t. \(bs\), of its last component(s).

Both these claims are proved in the formalization.

\textbf{Example 11.} Continuing Example \[10\] we find that

\[
[b_0, b_1, b_2, x_1^2 - x_0^2, x_0^2 - x_2^2, x_0 - x_3^3]
\]

constitutes a Gröbner basis of \(bs\), and that

\[
\begin{align*}
x_1^2 - x_0^2 &= -x_1 b_1 + x_0 b_2 \\
x_0^2 - x_2^2 &= x_2 b_0 - x_0 b_2 \\
x_0 - x_3^3 &= -x_1 b_0 - x_0 x_1 b_1 + (x_0^2 - 1) b_2.
\end{align*}
\]
8 Code Generation

The algorithms about (reduced) Gröbner bases and syzygies formalized in Isabelle/HOL can be turned into actual Haskell/OCam/Scala/SML code by means of Isabelle’s Code Generator [15]. The formalization contains a couple of sample computations, both by Buchberger’s algorithm and by the $F_4$ algorithm. One of the main reasons why we deem our formalization elegant is that setting up a computation is particularly easy: one does not have to care about the (number of) indeterminates featuring in a computation, as indeterminates are simply indexed by natural numbers, and it does not matter how many of them appear in a computation; the same is true for the dimension $k$ of the module under consideration, since in any case $k$ can just be instantiated by $\text{nat}$. In particular, it is not necessary to a-priori introduce dedicated types for univariate/bivariate/trivariate/… polynomials.

Table 1 shows the performance of the code generated from our certified algorithms on some benchmark problems, on a standard desktop computer. For comparison, Théry [33] reports 2 seconds for Cyclic 5 and 30 minutes for Cyclic 6, which on our computer reduces to 0.5 seconds and 2 minutes, respectively. We identified two main sources of inefficiency in our implementation, which explain the huge performance differences:

- In computations, power-products and polynomials are represented as associative lists. This representation is feasible but not optimal, since certain invariants could additionally be encoded in the representing type and exploited in code equations. For instance, one could require the keys in associative lists representing polynomials to be distinct and sorted descending w.r.t. the chosen ordering $\preceq_t$; then, most basic operations (addition, subtraction, lt, etc.) could be implemented much more efficiently than is currently the case. Although doable in principle, this is challenging since Isabelle does not support dependent types, which makes it difficult to encode parametric invariants in types (parametric because arbitrary term orders shall be supported). Improving the representation of power-products and polynomials is ongoing work, but first experiments indicate that such an improvement would indeed reduce computation times drastically: Buchberger’s algorithm only takes 40 minutes for Cyclic 6 then.

- Computations of Gröbner bases over $\mathbb{Q}$ often suffer from a so-called coefficient swell, i.e. the numerators and denominators of coefficients grow extremely large. For instance, in Cyclic 6 the largest denominators occurring during the computation have more than 200 digits. Hence, in order to handle such big numbers efficiently, one should use native types of rational numbers in the target languages (like $\text{Ratio.ratio}$ in OCaml, which is used in [33]). Isabelle’s Code Generator, however, constructs its own type of rational numbers as pairs of (native) integers. Experiments show that this setup is slower by a factor of about 20 compared to OCaml’s $\text{Ratio.ratio}$ when adding rational numbers with 200-digit denominators.

So, we have good reason to believe that Cyclic 6 can be solved by our certified implementation of Buchberger’s algorithm in roughly the same amount of time.

---

12 This is also proposed in [17].
as by Théry’s implementation in Coq+OCaml, once the two issues listed above have been sorted out. Still, the computation times are much slower than in state-of-the-art computer algebra systems like Maple [25] or Mathematica [36], which return the reduced Gröbner basis of Cyclic$_6$ in a split second.

Moreover, our implementation of the F$_4$ algorithm is in most cases considerably slower than Buchberger’s algorithm, although in Section 5 we claimed that it should be the other way round. One reason for this phenomenon certainly lies in the dense representation of matrices in [34] as IArrays of IArrays. As indicated in Section 5, a much more efficient representation of matrices in conjunction with optimized algorithms for computing reduced row echelon forms would be necessary to outperform Buchberger’s algorithm. Such representations and algorithms have not been formalized in Isabelle yet. Indeed, our motivation to consider F$_4$ was mainly academic interest, to demonstrate it can be formalized with moderate effort if only the algorithm schema gb_schema is formulated in a sufficiently general way, to establish the connection between the computation of Gröbner bases and Macaulay matrices, and to lay the foundation for more efficient Gröbner-basis computations by computer-certified state-of-the-art algorithms (like F$_5$ [14]) in the future.

**Example 12.** The Gröbner basis of the syzygy module of Example 10 can be computed in Isabelle/HOL as

```
value [code]
"syzygy_basis_drlex [Vec 0 0 (X * Y - Z), Vec 0 0 (X * Z - Y), Vec 0 0 (Y * Z - X)]"
```

which returns

```
[Vec 0 0 (- X * Z + Y) + Vec 0 1 (X * Y - Z),
 Vec 0 0 (- Y * Z + X) + Vec 0 2 (X * Y - Z),
 Vec 1 (- Y * Z + X) + Vec 0 2 (X * Z - Y),
 Vec 0 0 (Y - Y * Z ^ 2) + Vec 0 1 (Y ^ 2 * Z - Z) + Vec 0 2 (Y ^ 2 - Z ^ 2)]
```

X, Y and Z are predefined constructors of scalar polynomials, representing the first three indeterminates$^{13}$ and Vec$_0$(i, p) turns the scalar polynomial p into a vector of polynomials by setting the i-th component to p and all others to 0.

9 Conclusion

We hope we could convince the reader that the work described in this paper is an elegant, generic and executable formalization of an interesting and important mathematical theory in the realm of commutative algebra. Even though other formalizations of Gröbner bases in other proof assistants exist, ours is the first in Isabelle/HOL, and the first featuring the F$_4$ algorithm and Gröbner bases

$^{13}$More such constructors can be added on-the-fly when needed.
for modules. Besides, our work also gives an affirmative answer to the question whether multivariate polynomials à la [17] can effectively be used for formalizing theorems and algorithms in computer algebra.

Our own contributions to multivariate polynomials make up approximately 8600 lines of proof, and the Gröbner-bases related theories make up another 16700 lines of proof (3000 lines of which are about Macaulay matrices and the $F_4$ algorithm), summing up to a total of 25300 lines. Most proofs are intentionally given in a quite verbose style for better readability. The formalization effort was roughly eight person-months of full-time work, distributed over two years of part-time work. This effort is comparable to what the authors of other formalizations of Gröbner bases theory in other proof assistants report.

It is worth noting that all formalizations of Gröbner bases in existence restrict themselves to the basics of the theory. Indeed, browsing through the ample literature on the subject one quickly realizes that a lot more properties of Gröbner bases, ways of computing them, generalizations, and intriguing applications could be added to the corpus of formal mathematics in the future; examples include Gröbner bases over coefficient rings that are not fields, elimination orders and their applications, converting between different term orders, and non-commutative Gröbner bases. We plan to contribute to this endeavor by formalizing the very recent approach of computing Gröbner bases by transforming Macaulay matrices (or generalized Sylvester matrices) into reduced row echelon form [35] in Isabelle/HOL. This approach has similarities to the $F_4$ algorithm but only computes the reduced row echelon form of one big matrix, instead of doing this repeatedly in every iteration of a critical-pair/completion algorithm. Besides, formalizing said approach also necessitates proving upper bounds on the degrees of polynomials that may appear in a Gröbner basis, which in turn allows us to draw conclusions concerning the theoretical complexity of algorithms for computing Gröbner bases. All this is ongoing work.

References

[1] Adams, W.W., Loustaunau, P.: An Introduction to Gröbner Bases, Graduate Studies in Mathematics, vol. 3. AMS (1994)

[2] Ballarin, C.: Tutorial to Locales and Locale Interpretation. In: Lambán, L., Romero, A., Rubio, J. (eds.) Contribuciones Científicas en Honor de Mirian Andrés Gómez. pp. 123–140. Servicio de Publicaciones de la Universidad de La Rioja (2010), https://isabelle.in.tum.de/dist/Isabelle2016/doc/locales.pdf, part of the Isabelle documentation

[3] Bancerek, G., Byliński, C., Grabowski, A., Korniłowicz, A., Matuszewski, R., Naumowicz, A., Pąk, K., Urban, J.: 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’15). LNAI, vol. 9150, pp. 261–279. Springer (2015)

[4] Bentkamp, A.: Expressiveness of Deep Learning. Archive of Formal Proofs (2016), http://isa-afp.org/entries/Deep_Learning.html, formal proof development
[5] Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program De-
velopment – Coq’Art: The Calculus of Inductive Constructions. Texts in
Theoretical Computer Science. An EATCS Series, Springer (2004)

[6] Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des
Restklassenrings nach einem nulldimensionalen Polynomial. Ph.D. thesis,
Mathematisches Institut, Universität Innsbruck, Austria (1965)

[7] Buchberger, B.: A Criterion for Detecting Unnecessary Reductions in the
Construction of Gröbner Bases. In: Ng, E.W. (ed.) Symbolic and Algebraic
Computation (Proceedings of EUROSAM’79, Marseille, June 26-28).
Lecture Notes in Computer Science, vol. 72, pp. 3–21. Springer (1979)

[8] Buchberger, B.: 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(1),
65–75 (2004)

[9] Buchberger, B., Jebelean, T., Kutsia, T., Maletzky, A., Windsteiger, W.: Theorema 2.0: Computer-Assisted Natural-Style Mathematics. Journal of
Formalized Reasoning 9(1), 149–185 (2016)

[10] Chaieb, A., Wenzel, M.: 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’07). LNCS, vol. 4573, pp. 27–39.
Springer (2007)

[11] Craciun, A.: Lazy Thinking Algorithm Synthesis in Gröbner Bases The-
ory. Ph.D. thesis, Research Institute for Symbolic Computation, Johannes
Kepler University Linz, Austria (2008)

[12] Dickson, L.E.: Finiteness of the Odd Perfect and Primitive Abundant
Numbers with n Distinct Prime Factors. American Journal of Mathematics
35(4), 413–422 (1913)

[13] Faugère, J.C.: A New Efficient Algorithm for Computing Gröbner Bases
($F_4$). Journal of Pure and Applied Algebra 139(1), 61–88 (1999)

[14] Faugère, J.C.: A New Efficient Algorithm for Computing Gröbner Bases
without Reduction to Zero ($F_5$). In: Mora, T. (ed.) Proceedings of IS-
SAC’02. pp. 61–88. ACM Press (2002)

[15] Haftmann, F.: Code Generation from Isabelle/HOL Theories, [http://
isabelle.in.tum.de/dist/Isabelle2017/doc/codegen.pdf](http://isabelle.in.tum.de/dist/Isabelle2017/doc/codegen.pdf) part of the Is-
abelle documentation

[16] Haftmann, F.: Haskell-like Type Classes with Isabelle/Isar, [http://
isabelle.in.tum.de/dist/Isabelle2017/doc/classes.pdf](http://isabelle.in.tum.de/dist/Isabelle2017/doc/classes.pdf) part of the Is-
abelle documentation

[17] Haftmann, F., Lochbihler, A., Schreiner, W.: Towards Abstract and Exec-
utable Multivariate Polynomials in Isabelle (2014), [http://www.risc.
jku.at/publications/download/risc_5812/IWS14.pdf](http://www.risc.
jku.at/publications/download/risc_5812/IWS14.pdf) contributed talk at
Isabelle Workshop 2014, associated with ITP 2014, Vienna, Austria
[18] Harrison, J.: Complex quantifier elimination in HOL. In: Boulton, R.J., Jackson, P.B. (eds.) TPHOLs 2001: Supplemental Proceedings, pp. 159–174. Division of Informatics, University of Edinburgh (2001), http://www.informatics.ed.ac.uk/publications/report/0046.html

[19] Immler, F., Maletzky, A.: Gröbner Bases Theory. Archive of Formal Proofs (2016), http://isa-afp.org/entries/Groebner_Bases.html formal proof development

[20] Immler, F., Maletzky, A.: Gröbner Bases of Modules and Faugère’s $F_4$ Algorithm (2018), http://www.risc.jku.at/people/amaletzk/CICM2018.html compatible with Isabelle2017

[21] Jorge, J.S., Guilas, V.M., Freire, J.L.: Certifying properties of an efficient functional program for computing Gröbner bases. Journal of Symbolic Computation 44(5), 571–582 (2009)

[22] Kaufmann, M., Manolios, P., Moore, J.S.: Computer-Aided Reasoning: An Approach. Kluwer Academic Publishers (2000)

[23] Kreuzer, M., Robbiano, L.: Computational Commutative Algebra 1. Springer (2000)

[24] Maletzky, A.: Computer-Assisted Exploration of Gröbner Bases Theory in Theorema. Ph.D. thesis, RISC, Johannes Kepler University Linz, Austria (2016)

[25] Maplesoft: Maple 2018 (2018), https://www.maplesoft.com/products/maple/

[26] Medina-Bulo, I., Palomo-Lozano, F., Ruiz-Reina, J.L.: A Verified COMMON LISP Implementation of Buchberger’s Algorithm in ACL2. Journal of Symbolic Computation 45(1), 96–123 (2010)

[27] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL—A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)

[28] Persson, H.: An Integrated Development of Buchberger’s Algorithm in Coq. Tech. Rep. 4271, INRIA Sophia Antipolis (2001)

[29] Schwarzweller, C.: Gröbner Bases – Theory Refinement in the Mizar System. In: Kohlhase, M. (ed.) Mathematical Knowledge Management. LNAI, vol. 3863, pp. 299–314. Springer (2006)

[30] Sternagel, C.: Well-Quasi-Orders. Archive of Formal Proofs (2012), http://isa-afp.org/entries/Well_Quasi_Orders.html formal proof development

[31] Sternagel, C., Thiemann, R.: Abstract Rewriting. Archive of Formal Proofs (2010), http://isa-afp.org/entries/Abstract-Rewriting.html formal proof development

[32] Sternagel, C., Thiemann, R., Maletzky, A., Immler, F.: Executable Multivariate Polynomials. Archive of Formal Proofs (2010), http://isa-afp.org/entries/Polynomials.html formal proof development

27
[33] Théry, L.: A Machine-Checked Implementation of Buchberger’s Algorithm. Journal of Automated Reasoning 26(2), 107–137 (2001)

[34] Thiemann, R., Yamada, A.: Matrices, Jordan Normal Forms, and Spectral Radius Theory. Archive of Formal Proofs (2015), http://isa-afp.org/entries/Jordan_Normal_Form.html, formal proof development

[35] Wiesinger-Widi, M.: Gröbner Bases and Generalized Sylvester Matrices. Ph.D. thesis, RISC, Johannes Kepler University Linz, Austria (2015)

[36] Wolfram Research, Inc.: Mathematica 11.1 (2017). http://www.wolfram.com/mathematica/