Certified Computer Algebra

Henrik Persson
henrikp@cs.chalmers.se
Department of Computing Science,
Chalmers University of Technology and University of Göteborg,
S-412 96 Göteborg, Sweden.

Abstract

Computer algebra is about constructions and problems in algebra which can be efficiently computed or decided by mechanical procedures. A challenging point of importing computer algebra to type theory is that it require the formalisation of a considerable amount of non-trivial mathematics. The correctness of simple computer algebra algorithms can often depend on quite deep mathematical results, and since these results come from classical (and hence in general non-constructive) mathematics, it is far from clear that they can be imported to constructive type theory in a simple way.

In these notes, we will present two case-studies of computer algebra in type theory. We will emphasise two aspects of this development: that non-constructive arguments can be made *constructive*, and that the development can be done *abstractly* in type theory. We believe these two aspects are crucial for using type theory in practice since informal mathematics and computing science are in general non-constructive and abstractly presented. Furthermore, abstract formalisations are simpler to reuse and import to other theorem provers.

The first case-study is an integrated development of Buchberger's algorithm for computing Gröbner bases in Martin-Löf's type theory. The development is integrated in the sense that we prove constructively in type theory the existence of Gröbner bases and from this proof extract the algorithm. In the second case-study, we define the polynomials over an arbitrary ring R and a set of variables $\mathbf{X}$, and prove that they form a free R-algebra. A major part of the development is independent of the particular representation of polynomials. This is achieved by using universal properties to define and prove properties about functions.

Case-study 1: Buchberger's Algorithm[1]

In this case-study, we consider how Buchberger's algorithm for computing Gröbner bases can be developed within Martin-Löf's type theory. In particular, we aim for an *integrated* development of the algorithm: we want to prove, constructively in type theory, the existence of Gröbner bases and from such proofs extract the algorithms. Our main contribution is a reformulation of the standard theory of Gröbner bases which uses generalised inductive definitions. We isolate the main non-constructive part, a minimal bad sequence argument, and use the open induction principle [Rao88, Coq92] to interpret it by induction. This leads to short constructive proofs of Dickson's lemma and Hilbert's basis theorem, which are used to give an integrated development of Buchberger's algorithm. An important point of this work is that the elegance and brevity of the original proofs are maintained while the new proofs also have a direct constructive content. In the appendix we present a computer formalisation of Dickson's lemma and an abstract existence proof of Gröbner bases.

1.1 Introduction

This work is part of a project to develop computational algebra completely within Martin-Löf's type theory [NPS90], in an integrated fashion. Since the birth of the subject, algorithms in computational algebra have usually been *externally* developed: an algorithm is given and its correctness and termination is proved using classical logic. A possible reason for this approach is that classical abstract algebra is inherently non-constructive; e.g. existence-proof of primitive objects like prime and maximal ideals require Zorn's lemma, a highly non-constructive principle. This makes the other approach difficult, the *integrated* development, where an algorithm is extracted from a constructive existence proof. The notion of integrated and external programming logics was introduced by Girard [Gir86], for a comparison between the integrated and external approach to program development, see [Dyb90].

Gröbner bases together with an algorithm for computing them, was introduced by Buchberger [Buc65, Buc85]. It can be seen as a generalisation of the Euclidian algorithm for computing the greatest common divisor (gcd) of polynomials in several variables. In the case of polynomials in one variable, one can easily decide whether a polynomial f is in the ideal generated by the set of polynomials $F = \{f_1, \ldots, f_n\}$: just compute the gcd of F, since it generates the ideal of F, it is enough to check whether it divides f. This gives many algorithmic solutions to problems concerning polynomials in one variable.

[1] This is joint work with Thierry Coquand to appear in the Types'98 proceedings, LNCS, Springer-Verlag.

However, in the case of polynomials in several variables, this technique does not work. One problem is that F may not have a single generator of its ideal; hence one needs to define when a polynomial is divided by a set of polynomials. Another more important problem is that this technique is not complete: if F divides f then f is in the ideal of F, but f might be in the ideal of F even though F does not divide f. This is where Gröbner bases come into play; a Gröbner basis G of F is a finite set of polynomials which generates the same ideal and divides f if and only if f is in the ideal of F.

There are several thorough presentations of Gröbner bases and their applications [Buc85, Buc98, BW93, CLO97, Frö97]. However, these proofs are in general not constructive, which makes their development hard to translate into type theory. Our main contribution is a reformulation of the standard theory of Gröbner bases which replaces the non-constructive arguments by the use of generalised inductive definitions [Acz77]. A natural question is then whether the original elegant arguments are lost forever in this constructive framework? We show that this is not so by isolating the main non-constructive principle, a minimal bad sequence argument, and use the open induction principle [Rao88, Coq92] to interpret it by induction. This leads to short constructive proofs which follow the original arguments, and which have been formalised on computer.

In Section 1.2, we present a constructive existence proof of Gröbner bases for polynomial rings with field-coefficients. This proof uses a short constructive proof of Dickson's lemma [Dic13] which was extracted from a classical proof using open induction. During this work, we became aware of the work in [Thé98], where Théry presents a formal verification in Coq [HKPM97] of Buchberger's algorithm for computing Gröbner bases for polynomial rings with field coefficients. This formal proof is constructive, except for a classical proof of Dickson's lemma [Pot96]. A difference is that Théry's development is external; he starts with a program for Buchberger's algorithm and proves that it computes Gröbner bases, whereas we prove the existence of Gröbner bases constructively in such a way that Buchberger's algorithm is contained in the proof.

In Section 1.3, we present a short constructive proof of Hilbert's basis theorem, also extracted from a classical proof using open induction. This theorem can be used to prove termination of generalisations of Buchberger's algorithm for computing Gröbner bases for polynomials over principal ideal domains [BW93] and other algebraic structures [JL91]. Another approach was taken by Jacobsson and Löfwall [JL91] who proved constructively Hilbert's basis theorem by using Gröbner bases and a different definition of Noetherian.

This work is similar in spirit to Berger and Schwichtenberg [BS96], where an algorithm for computing the gcd of natural numbers is extracted from a classical existence proof. A difference is that they extract the proof automatically from a formal classical proof, whereas we manually rephrase an informal classical proof.

1.2 Gröbner Bases for Fields

In this section we assume K to be an arbitrary field with a decidable equality, e.g. the rationals $\mathbb{Q}$. We will consider the polynomial ring $K[X_1, \ldots, X_m]$, i.e. the set of polynomials in m variables with coefficients in K. A good suggestion on how to define polynomial rings in type theory can be found in [Jac95], where a formalisation in Nuprl [Con86] is described. We abbreviate monic monomials $X_1^{k_1} \cdots X_m^{k_m}$ as X^α, where $\alpha = (k_1, \ldots, k_m)$. We say that that X^α *divides* X^β if $\alpha \leq^m \beta$, where $(k_1, \ldots, k_m) \leq^m (l_1, \ldots, l_m) = k_1 \leq l_1 \,\&\cdots\&\, k_m \leq l_m$.

When the m is clear from the context, we will omit it and write $\leq$. Note that $X^\alpha \cdot X^\beta = X^{\alpha+\beta}$, where $\alpha + \beta = (k_1 + l_1, \ldots, k_m + l_m)$, and $\frac{X^\alpha}{X^\beta} = X^{\alpha-\beta}$, where $\alpha - \beta = (l_1 - k_1, \ldots, l_m - k_m)$ if X^β divides X^α. We define the *least common multiple of* α *and* β, $lcm(\alpha, \beta)$, as $(\max(k_1, l_1), \ldots, \max(k_m, l_m))$, if $\alpha = (k_1, \ldots, k_m)$ and $\beta = (l_1, \ldots, l_m)$.

We assume a total *compatible* well-founded order $\succ$ on the monomials, that is an order where $\alpha_1 \succ \alpha_2$ implies $\alpha_1 + \beta \succ \alpha_2 + \beta$. One possible order is: first order in terms of total degree, then monomials with equal total degree are ordered lexicographically. If f is a polynomial, $c_1 \neq 0$ and $c_1 X^{\alpha_1}$ is the highest monomial in f w.r.t. this order, we define $hd\ f = c_1 X^{\alpha_1}$. We define the *multi-degree* of such f, $mdeg(f)$, to be α_1.

Next, we define a reduction algorithm in the ring $K[X_1, \ldots, X_m]$:

Definition 1.2.1 A *reduction* of f after reducing by a set of non-zero polynomials $G = \{g_1, \ldots, g_n\}$, $RED(f; G)$, is defined by $\succ$-recursion on $mdeg(f)$:

$$
\begin{aligned}
RED(0; G) &= 0, \\
RED(f; G) &= \begin{cases} RED(f - \frac{hd\ f}{hd\ g_i} g_i; G), & \text{if } \exists g_i \in G.\, mdeg(g_i) \leq mdeg(f), \\ f, & \text{otherwise.} \end{cases}
\end{aligned}
$$

To make RED into a deterministic algorithm, one must decide on a strategy to choose the g_i in the above clause; for example to try g_1 first, then g_2, and so on. One problem is that the choice of this strategy might affect the result; consider e.g. $f = XY^2 + Y^2 + X$, $g_1 = XY - 1$ and $g_2 = Y^2 - 1$: if we always try to reduce by g_1 first, $RED(f; g_1, g_2) = X + Y + 1$, whereas if we always try g_2 first, $RED(f; g_1, g_2) = 2X + 1$. Another problem is that RED does not give a decision procedure for membership in ideals; e.g. $RED(X; Y^2 + X, Y) = X \neq 0$ for any strategy but $X \in \mathrm{Idl}(Y^2 + X, Y)$, where $\mathrm{Idl}(a_1, \ldots, a_n)$ is the ideal generated by $a_1, \ldots, a_n$. We say that a finite set is a Gröbner basis, if we can use RED as a decision procedure to decide the ideal it generates:

Definition 1.2.2 G is a *Gröbner basis* (for the ideal it generate), if $RED(f; G) = 0$ whenever $f \in \mathrm{Idl}(G)$.

One can prove that $RED(f; G)$ is unique and independent of strategy when G is a Gröbner basis, see e.g. [BW93].

1.2.1 Construction of Gröbner Bases

Rather than first giving an algorithm which constructs Gröbner bases for ideals and then prove it correct, we will give a direct and *constructive* proof that for any finitely generated ideal, there exists a corresponding Gröbner basis. Since this proof is constructive, it will in particular contain an algorithm for constructing Gröbner bases.

To motivate the development, consider the example set $\{Y^2 + X, Y\}$; this is not a Gröbner basis as explained above, since $X \in \mathrm{Idl}(Y^2 + X, Y)$, but $RED(X; Y^2 + X, Y) \neq 0$. The first step is to find a systematic method to generate all such possible counter-examples.

Definition 1.2.3 Given two polynomials f and g, their *S-polynomial*, $spol(f, g)$, is defined as

$$spol(f, g) = \frac{X^\alpha}{hd\ f} \cdot f - \frac{X^\alpha}{hd\ g} \cdot g$$

where $\alpha = lcm(mdeg(f), mdeg(g))$.

S-polynomials give a practical characterisation of Gröbner bases:

Theorem 1.2.4 *$G = \{g_1, \ldots, g_t\}$ is a Gröbner basis if $RED(\mathrm{spol}(g_i, g_j); G) = 0$ for all $i < j \leq t$.*

Proof. In this case we prove that the set of elements f such that $RED(f; G) = 0$ is closed by addition. Since this set is clearly closed by multiplication by monomials, this will imply that we have $RED(f; G) = 0$ for all f in $\mathrm{Idl}(G)$, as desired.

We prove that if $RED(f; G) = RED(g; G) = 0$ then $RED(f + g; G) = 0$ by induction on $mdeg(f)$ and $mdeg(g)$. The only case which is not direct is if we have $mdeg(f) = mdeg(g)$, $RED(f; G) = RED(f - m_i g_i; G)$, $RED(g; G) = RED(g - m_j g_j; G)$, where $m_i = r_i X^{\alpha_i}, m_j = r_j X^{\alpha_j}$ are suitable monomials, and $mdeg(f - m_i g_i) \prec mdeg(f)$, $mdeg(g - m_j g_j) \prec mdeg(g)$. We can write

$$g_i = s_i X^{\beta_i} + h_i, \; g_j = s_j X^{\beta_j} + h_j$$

with $mdeg(h_i) \prec mdeg(g_i)$ and $mdeg(h_j) \prec mdeg(g_j)$. We have then

$$f + g - (r_i + \frac{r_j c_j}{c_i}) X^{\alpha_i} g_i = m spol(g_i, g_j) + f - m_i g_i + g - m_j g_j$$

for a suitable monomial m. By induction hypothesis the right hand side reduces to 0, hence so does $f + g$. $\square$

A naive approach to compute a Gröbner basis for a set is to add counter-examples (S-polynomials) to it until it satisfies the condition in Theorem 1.2.4. Quite surprisingly, this process will always terminate. The proof of this relies on a non-trivial combinatorial result known as *Dickson's lemma*: for any sequence of monic monomials $X^{\alpha_1}, X^{\alpha_2}, \ldots$, there will eventually be $i < j$ such that X^{α_i} divides X^{α_j}.

1.2.2 A Constructive Proof of Dickson's Lemma

In this subsection, we present a constructive proof of Dickson's lemma. The proof is a translation of the classical proof in Appendix A; the main non–constructive part, a minimal bad sequence argument, have been replaced by the open induction principle [Rao88, Coq92]. Dickson's lemma says that for any infinite sequence of n-tuples of natural numbers $\sigma_1, \sigma_2, \ldots$, there exists $i < j$ such that $\sigma_i \leq^n \sigma_j$, where $(a_1, \ldots, a_n) \leq^n (b_1, \ldots, b_n)$ if $\forall 0 < i \leq n.\, a_i \leq b_i$. Classically, a relation is called *well*[2] if it satisfies the condition above.

Remark 1.2.1 The proofs in this section only require $<$ to be a decidable relation which is well–founded on its underlying set A, with $a \leq b$ defined as $\neg(b < a)$. However, we will only instantiate the theorems for $<$ being the less-than relation on $\mathbb{N}$.

We want to express in type theory, extended with inductive definitions, what it means for a relation R over a set B to be well. To this end, we define $Good_R(b_0 \cdots b_m)$ to be $\exists i < j \leq m.\, b_i \, R \, b_j$. We use an inductive definition of *bar* [ML68] to express that for any infinite sequence $b_0 b_1 \cdots$, $Good_R(\sigma)$ will eventually hold for an initial segment σ of $b_0 b_1 \cdots$.

Definition 1.2.5 Given a set B and a predicate P over the lists of B, we define inductively when the predicate P *bars* σ, written $P \mid \sigma$:

[2] in previous work, the relation was required to be a quasi-order (well-quasi order).

$$\frac{P(\sigma)}{P \mid \sigma} \qquad \frac{\forall b.\, P \mid \sigma.b}{P \mid \sigma}$$

This is a generalised inductive definition [Acz77], which comes with a transfinite induction principle:

$$\frac{\begin{array}{c}\forall \rho.\, P(\rho) \Rightarrow \Psi(\rho), \\ \forall \rho.\, (\forall b.\, P \mid \rho.b) \;\Rightarrow\; (\forall b.\, \Psi(\rho.b)) \;\Rightarrow\; \Psi(\rho)\end{array}}{P \mid \sigma \;\Rightarrow\; \Psi(\sigma)}$$

Here $\sigma.b$ is the list σ extended with the element b. Intuitively, $P \mid \sigma$ means that P will eventually hold for any extension of σ. Classically, assuming the axiom of dependent choices, $Good_R \mid []$ is provable iff R satisfies the classical definition of well. This justifies us to define R to be well iff $Good_R \mid []$ is provable.

Lemma 1.2.6 *If* $\forall \sigma, \rho, \gamma.\, P(\sigma\rho) \Rightarrow P(\sigma\gamma\rho)$, *then* $\forall \sigma, \rho, \gamma.\, P \mid \sigma\rho \;\Rightarrow\; P \mid \sigma\gamma\rho$.

Proof. Immediate by induction on the proof of $P \mid \sigma\rho$. □

Given two relations R and S over sets A and B respectively, the product relation, $R \times S$, over $A \times B$ is defined as $(a, b)\,(R \times S)\,(a', b') = a\,R\,a' \;\&\; b\,S\,b'$. Following [Coq92], we define a predicate $M(\sigma)$, expressing that an initial sequence σ of pairs in $A \times B$ is *minimal w.r.t.* $<$, by recursion on σ:

$$\begin{array}{rcl} M([]) & = & \top, \\ M(\sigma.(x, b)) & = & M(\sigma) \;\&\; \forall y.\, y < x \Rightarrow \forall b.\, Good_{\leq \times R} \mid \sigma.(y, b). \end{array}$$

The predicate M will play the rôle of the minimal bad sequence in Appendix A.1; if $M(\sigma)$ holds, and ρ has a lexicographically smaller sequence of first components, then $Good_{\leq \times R} \mid \rho$ should hold.

Raoult's open induction principle can be expressed using these definitions:

Theorem 1.2.7 (Open Induction) *For any finite sequence* σ *of pairs in* $A \times B$, *if* $M(\sigma)$ *and* $\forall a, b.\, M(\sigma.(a, b)) \;\Rightarrow\; Good_{\leq \times R} \mid \sigma.(a, b)$, *then* $Good_{\leq \times R} \mid \sigma$.

Proof. Assume $M(\sigma)$ and $\forall a, b.\, M(\sigma.(a, b)) \;\Rightarrow\; Good_{\leq \times R} \mid \sigma.(a, b)$ holds. By induction on x, we prove $\forall x. \forall b.\, Good_{\leq \times R} \mid \sigma.(x, b)$: Assume $\forall y.\, y < x \Rightarrow \forall b.\, Good_{\leq \times R} \mid \sigma.(y, b)$. From this we directly obtain $M(\sigma.(x, b))$, and by hypothesis, $Good_{\leq \times R} \mid \sigma.(x, b)$. □

This theorem is a simplification of that in [Coq92] but it uses only generalised inductive definitions iterated once [Acz77]. It will interpret the argument: if $Good_{\leq \times R} \mid \sigma$ holds under the assumption that σ starts a minimal bad sequence, then $Good_{\leq \times R} \mid \sigma$ holds without this assumption as well. Therefore, the classical proof of Dickson's lemma in Appendix A can be interpreted as:

Lemma 1.2.8 *If* $Good_R \mid b_1 \cdots b_m$ *holds, then*

$$\forall x_1, \ldots, x_m.\, M((x_1, b_1) \cdots (x_m, b_m)) \;\Rightarrow\; Good_{\leq \times R} \mid (x_1, b_1) \cdots (x_m, b_m).$$

Proof. By induction on the proof of $Good_R \mid b_1 \cdots b_m$:

$Good_R(b_1 \cdots b_m)$: Then there exists a $i < j \leq m$ such that $b_i \, R \, b_j$. Now, by cases on the decidable $<$, we have either $\neg(x_i > x_j)$, that means $x_i \leq x_j$ so $(x_i, b_i) \, (\leq \times R) \, (x_j, b_j)$ holds, hence $Good_{\leq \times R}((x_1, b_1) \cdots (x_m, b_m))$. In the other case we have $x_i > x_j$, and from $M((x_1, b_1) \cdots (x_m, b_m))$ we directly obtain $M((x_1, b_1) \cdots (x_i, b_i))$, which entails $Good_{\leq \times R} \mid (x_1, b_1) \cdots (x_{i-1}, b_{i-1})(x_j, b_j)$, and by Lemma 1.2.6, we are done.

$\forall b.\, Good_R \mid (b_1 \cdots b_m b)$: Immediate by Theorem 1.2.7 and IH.

□

Corollary 1.2.9 (Dickson's lemma) *For all $n \in \mathbb{N}$, $Good_{\leq^n} \mid [\,]$.*

Proof. By induction on n: the case $n = 0$ is trivial. Otherwise, $n = m + 1$, and $Good_{\leq^m} \mid [\,]$ holds. We instantiate the development above with R being $\leq^m$, and by Lemma 1.2.8 we are done. □

Dickson's lemma implies the existence of Gröbner bases for any finitely generated (f.g.) ideal in $K[X_1, \ldots, X_m]$:

Theorem 1.2.10 *Every f.g. ideal $F = \mathrm{Idl}(f_1, \ldots, f_n)$ has a Gröbner basis G.*

Proof. We prove this using Dickson's lemma. Let $m_i = mdeg(f_i)$. Define $Bad(\sigma)$ as $\neg Good(\sigma)$. We can assume that $Bad(m_1 \cdots m_n)$ holds; if $m_i \leq m_j$ for $i < j$, we repeatedly reduce F by dividing f_j by f_i. The result follows from Dickson's lemma and the following lemma:

$$\begin{aligned} Good_{\leq} \mid m_1 \cdots m_k \;\Rightarrow\; & \forall f_1, \ldots, f_k.\, (\forall i.\, m_i = mdeg(f_i)) \;\Rightarrow \\ & Bad(m_1 \cdots m_k) \;\Rightarrow\; \exists G.\, G \text{ is a Gröbner basis for } f_1, \ldots, f_k, \end{aligned}$$

which is proved by induction on the proof of $Good_{\leq} \mid m_1 \cdots m_k$:

$Good(m_1 \cdots m_k)$: This contradicts with $Bad(m_1 \cdots m_k)$.

$\forall m.\, Good_{\leq} \mid m_1 \cdots m_k m$: Assume $f_1, \ldots, f_k$ with $mdeg(f_i) = m_i$ for all i. Consider the polynomials $RED(spol(f_i, f_j); f_1, \ldots, f_k)$ for $i \neq j$. If all are zero, $\{f_1, \ldots, f_k\}$ is already a Gröbner basis by Theorem 1.2.4. Otherwise, let f_{k+1} be the first polynomial $RED(spol(f_i, f_j); f_1, \ldots, f_k) \neq 0$. Then $m_l \not\leq mdeg(f_{k+1})$ for all $l \leq k$, so if $Bad(m_1 \cdots m_k)$, then $Bad(m_1 \cdots m_k mdeg(f_{k+1}))$. Hence, by IH, we have a Gröbner basis for $F \cup \{f_{k+1}\}$, and since f_{k+1} is in $\mathrm{Idl}(F)$, this is a Gröbner basis for F.

□

This is an integrated version of Buchberger's algorithm: while F is not a Gröbner basis, add normalised S-polynomials to F. Optimisations of the algorithm can be made by changing the proof, e.g. the order in which to compute the S-polynomials.

1.3 Hilbert's Basis Theorem

In this section we prove constructively Hilbert's Basis Theorem (HBT). This is used to prove termination of generalisations of Buchberger's algorithm for computing Gröbner bases for polynomials over principal ideal domains [BW93] and other algebraic structures [JL91].

In these more general cases, there are similar notions of reductions and ways of generating counter-examples and to decide whether the set is a Gröbner basis. As in the previous section, we need to prove that the process of adding counter-examples to the ideal ends. This follows from HBT, which concerns *Noetherian* rings. There are several classically equivalent definitions of Noetherian:

Definition 1.3.1 A ring R is *Noetherian*, if either of the following classically equivalent conditions holds:

1. every ideal in R is finitely generated,

2. there exists no infinite strictly increasing sequence of ideals,

3. for every infinite sequence $a_0, a_1 \ldots$ of elements in R, there exists an m such that $a_m \in \mathrm{Idl}(a_0, \ldots, a_{m-1})$.

HBT says that if R is a Noetherian ring, so is $R[X_1, \ldots, X_m]$.

1.3.1 A Constructive Proof of Hilbert's Basis Theorem

In this subsection, we present a constructive proof of HBT. Again, the proof is a translation of the classical proof in Appendix A; as in the previuos section, the open induction principle [Rao88, Coq92] can be used to replace the minimal bad sequence argument. Contrary to the constructive proofs of HBT previously known to us [Ric74, MRR88, JL91], the proof extracted does not require decidability of the equality in the ring or its ideals.

We want to express in type theory, extended with inductive definitions, the notion of Noetherian rings. To this end, we define $Good_R(a_0 \cdots a_m)$ to be $\exists k \leq m.\, a_k \in \mathrm{Idl}(a_0, \ldots, a_{k-1})$ for any ring R. The inductive definition of bar gives a good constructive definition of Noetherian: classically, assuming the axiom of dependent choices, $Good_R \mid []$ is provable iff R satisfies the second condition in Definition 1.3.1. This justifies us to define a ring R to be Noetherian iff $Good_R \mid []$ is provable.

Lemma 1.3.2 *If $Good_R \mid a_0 \cdots a_k\sigma$, then $Good_R \mid a_0 \cdots a_{k-1}(a_k + \sum_{i=0}^{k-1} r_i a_i)\sigma$.*

Proof. By induction on the proof of $Good_R \mid a_0 \cdots a_k\sigma$: If $Good_R(a_0 \cdots a_k\sigma)$, then it is direct since any $\sum_{i=0}^{k-1} r_i a_i$ is in $\mathrm{Idl}(a_0, \ldots, a_{k-1})$. Otherwise, we have $\forall a.\, Good_R \mid a_0 \cdots a_k\sigma a$, and the result follows by IH. □

To prove HBT, it is enough to prove that for any finitely generated R-module $R[X]$, if R is Noetherian, then $R[X]$ is Noetherian. In the rest of this section, we assume $R[X]$ to be a finitely generated R-module. To interpret the minimal bad sequence argument in Appendix A.2, we define a predicate $M(\sigma)$, expressing that the finite sequence σ of lists of R is *minimal*, and prove a corresponding open induction principle for it.

$$\begin{aligned} M([]) &= \top \\ M(\sigma.f) &= M(\sigma) \;\&\; \forall g.\, |g| < |f| \Rightarrow P(\sigma.g) \end{aligned}$$

where $|f|$ is the length of the list f, and $P(f_1 \cdots f_k) = Good_{R[X]} \mid \varphi(f_1) \cdots \varphi(f_k)$ where $\varphi(a_0 \cdots a_k) = a_0 + a_1 X + \cdots + a_k X^k$. If we write $\delta f = 0.f$, where $0.f$ is the list f with 0 put in front, then $\varphi(\delta f) = X\varphi(f)$. We use lists here rather than elements of $R[X]$ to avoid a decidable equality in R in the proof of HBT.

Theorem 1.3.3 (Open Induction) *For any finite sequence σ of lists of R, if $M(\sigma)$ and $\forall f.\, M(\sigma.f) \;\Rightarrow\; P(\sigma.f)$, then $P(\sigma)$.*

Proof. Assume σ such that $M(\sigma)$ and $\forall f.\, M(\sigma.f) \;\Rightarrow\; P(\sigma.f)$. We prove $\forall f.\, P(\sigma.f)$ by induction on the degree of f: Assume $\forall g.\, |g| < |f| \Rightarrow P(\sigma.g)$. This directly implies $M(\sigma.f)$, and by hypothesis, $P(\sigma.f)$. □

The classical proof of Hilbert's Basis Theorem in Appendix A.2 becomes:

Lemma 1.3.4 *Given $a_0, \ldots, a_m$ of R such that $Good_R \mid (a_0 \cdots a_m)$ holds, and a sequence $f_0, \ldots, f_m$ of lists of R, $M((f_0.a_0) \cdots (f_m.a_m)) \Rightarrow P((f_0.a_0) \cdots (f_m.a_m))$.*

Proof. By induction on the proof of $Good_R \mid a_0 \cdots a_m$:

$Good_R(a_0 \cdots a_m)$**:** Then there exists a $k \leq m$ such that $a_k \in \mathrm{Idl}(a_0, \ldots, a_{k-1})$, hence $a_k = r_0 a_0 + \cdots + r_{k-1} a_{k-1}$ for $r_0, \ldots, r_{k-1} \in R$. Let $g_i = f_i.a_i$. Either there exists $0 \leq i < k$ such that $|g_i| > |g_k|$. In that case, since we have $M(g_0 \cdots g_i)$, we obtain $P(g_0 \cdots g_{i-1} g_k)$ and by Lemma 1.2.6 repeatedly, $P(g_0 \cdots g_{i-1} g_i \cdots g_k \ldots g_m)$. In the other case, $|g_i| \leq |g_k|$ for all $i < k$, and we construct

$$g^* = g_k - \sum_{i=0}^{k-1} r_i(\delta^{|g_k|-|g_i|} g_i),$$

where summation of lists of equal lenght are just pointwise addition, and scalar multiplication is taken pointwise. Since $|g^*| < |g_k|$ and $M(g_0 \cdots g_k)$, we have $P(g_0 \cdots g_{k-1} g^*)$. Note that $\varphi(g^*) = \varphi(g_k) - \sum_{i=0}^{k-1} r_i(X^{|g_k|-|g_i|}\varphi(g_i))$, so by Lemma 1.3.2, $P(g_0 \cdots g_{k-1} g_k)$, and by Lemma 1.2.6, we are done.

$\forall a.\, Good_R \mid a_0 \cdots a_m a$**:** By IH, we get for any $a \in R$ and sequence $f_0, \ldots, f_m, f$ of lists of R, $M((f_0.a_0) \cdots (f_m.a_m)(f.a)) \;\Rightarrow\; P((f_0.a_0) \cdots (f_m.a_m)(f.a))$. This also holds if $f.a$ is replaced by the empty list, so by Theorem 1.3.3, $P(f_0 \cdots f_m)$.

□

Corollary 1.3.5 (Hilbert's Basis Theorem) $Good_R \mid [\,] \Rightarrow Good_{R[X_1,\ldots,X_m]} \mid [\,]$.

Proof. By induction on m and Lemma 1.3.4, since $R[X]$ is a ring if R is. □

1.4 Conclusions

This work shows how a non-trivial part of classical mathematics can be translated into constructive type theory by using the open induction principle. The constructive proofs share the elegance and brevity of the original proofs, and has a direct formalisation in type theory. This is shown in Appendix B below, where a computer formalisation of Dickson's lemma and an abstract existence proof of Gröbner bases is presented.

This can be seen as a general and integrated development of Buchberger's algorithm in a functional language. To be able to execute this program, one needs to formalise a polynomial ring over a field, the reduction function, and Theorem 1.2.4. In the next case-study below, we describe how the polynomial ring can be formalisation in an abstract way. For the remaining formalisation, one should be able to use the already existing work of Jackson [Jac95] and Théry [Thé98].

Case-study 2: Polynomial Ring

In this case-study, we develop an abstract formalisation of the polynomial ring $R[\mathbf{X}] = R[X_1, \ldots, X_n]$ over a ring R. We aim for the highest possible abstraction; that is, we want the development to be as independent of how polynomials are represented as possible. This is obtained by using the notion of universal properties which can be seen as abstract elimination rules. The proofs have been partially formalised in Agda [Coq98], which is a type checker for a version of Martin-Löf's type theory [NPS90].

2.5 Introduction

It is well-known that the polynomial ring $R[\mathbf{X}]$ forms a free R-algebra over $X_1, \ldots, X_n$ (see e.g. [Lan93]). This is a complete characterisation of $R[\mathbf{X}]$ in the sense that whenever we want to reason about a polynomial ring, we can just as well reason about the corresponding free R-Algebra. The power of the free R-Algebra comes from its universal property, which gives a kind of elimination-rule for polynomials. The universal property gives an abstract way to define functions over $R[\mathbf{X}]$ which otherwise would require access to the representation of $R[\mathbf{X}]$, for example the function which computes the sum of the coefficients in a polynomial. However, intensional functions over $R[\mathbf{X}]$ which depend on the representation can not be defined, for example the function which returns the third term in polynomial f cannot be defined since the notion of "third term" depends highly on the representation of $R[\mathbf{X}]$[3].

In his thesis [Jac95], Jackson presents a similar formalisation of the polynomial ring in Nuprl [Con86]. We believe our work improves his on two points. First, the structure of the development is more clear because of the abstraction-mechanisms in Agda, for example the iteration of the monoid copower construction below can easily be expressed in the language. Secondly, we are using the universal property, rather than the underlying representation, to define and reason about functions.

This development can then be instantiated with a concrete representation of polynomials. For example, we have used a representation where monomials are represented as association lists (lists of pairs) associating natural numbers (exponents) to variables, and polynomials are represented as association lists associating monomials to elements in the ring R (coefficients). That means that

$$
\begin{array}{rll}
X_1^{n_1} X_2^{n_2} X_3^{n_3} & \text{is represented as} & [(X_1, n_1), (X_2, n_2), (X_3, n_3)], \\
c_1 X_1^{n_1} X_2^{n_2} X_3^{n_3} + c_2 X_2^{m_2} & \text{is represented as} & [([(X_1, n_1), (X_2, n_2), (X_3, n_3)], c_1), \\
 & & ([(X_2, m_2)], c_2)].
\end{array}
$$

[3]Of course, the notion of "third term w.r.t. a particular order" is well-defined and the corresponding function can be defined.

In our development, such concrete representations are used as little as possible. This has several advantages: the proofs get simpler and cleaner which makes them easier to understand and quicker to type-check. The proofs are also easier to change, and should be easier to export to other theorem provers.

2.6 Monoid Copower

The monoid copower is an algebraic structure common to the two constructions

1. variables $X_1, \ldots, X_k$ and natural numbers $n_1, \ldots, n_k$
   $\Longrightarrow$ monomials $X_1^{n_1} \cdot \cdots \cdot X_k^{n_k}$,
2. monomials $m_1, \ldots, m_k$ and coefficients $c_1, \ldots, c_k$
   $\Longrightarrow$ polynomials: $c_1 m_1 + \cdots + c_k m_k$.

This construction is used twice to construct polynomials and is the only construction which require a concrete choice of representation. Hence, if we can express this construction abstractly in Agda, we can parametrise the whole development of polynomials on a choice of representation.

The idea to structure the development around the monoid copower is taken from [Jac95]. In abstract algebra such as [Lan93], the construction of the polynomial ring is essentially structured in this way. One difference is that they distinguish the two constructions above: the first is a free monoid construction over the set of variables, whereas the second is a free R-algebra construction over the monoid of monomials. Since these constructions usually have the same representation, this distinction leads to duplication of several definitions and proofs.

In the rest of these notes, by monoid we always mean Abelian monoid. If M and N are monoids, we write $M \Rightarrow N$ for the set of homomorphisms between M and N. We use the convention that if M is a monoid, we denote its carrier by M and its operator by $+_M$ or $*_M$ and the corresponding unit by 0_M or 1_M. Similarly, the components of a ring R are denoted by R, $+_R$, $*_R$, $-_R$, 0_R and 1_R. When the domain is clear from the context, we will often omit the index and write $0, 1, +, *$ and $-$ without the M or R.

Definition 2.6.1 $\coprod_{v \in V} M$ is a[4] *monoid copower* of a monoid M and a set V, if

1. $\coprod_{v \in V} M$ is a monoid,
2. there exists a homomorphism $\iota_v \in M \Rightarrow \coprod_{v \in V} M$ for all $v \in V$,
3. for any monoid H and family of homomorphisms $f_v \in M \Rightarrow H$, there exists a unique homomorphism $f' \in \coprod_{v \in V} M \Rightarrow H$ such that

$$\begin{tikzcd} M \arrow[r, "f_v"] \arrow[d, "\iota_v"'] & H \\ \coprod\limits_{v \in V} M \arrow[ur, "f'"'] & \end{tikzcd}$$

[4]one can show that the monoid copower is unique up to isomorphism, but formally we want to distinguish between different implementations.

commutes for all v.

With this definition, $\coprod_{v \in V} M$ is the copower of V in the category of monoids **Mon** in the sense of [ML71], that is a coproduct $\coprod_{x \in V} M_x$ in **Mon** where all the monoids M_x are equal to M.

To implement instances of this definition in type theory and Agda, we will require the underlying sets of all structures like M and V to be equipped with a decidable equivalence relation. This condition is not needed in classical mathematics, but we need it in order for several functions to be computable. Similarly, we will require that V is linearly ordered with a decidable relation $>$. This is not needed for the functions to be computable, but natural if we want them to be more efficient. With these requirements, it is possible to represent monomials and polynomials as association-lists with ordered keys. Such an order is anyhow required later if we want to have notions like the head-term of a polynomial.

The universal property gives a very powerful way of defining and proving properties about functions. For example, it entails the following induction principle for $\coprod_{v \in V} M$:

Theorem 2.6.2 *For any predicate φ over $\coprod_{v \in V} M$, if we can prove $\varphi(\iota_v(m))$ for all $v \in V$ and $m \in M$, and that $\varphi(a) \Rightarrow \varphi(b) \Rightarrow \varphi(a+b)$ for all $a, b \in \coprod_{v \in V} M$, then $\varphi(a)$ holds for any $a \in \coprod_{v \in V} M$.*

Proof. Define A to be the set of a for which $\varphi(a)$ holds, defined in type theory as $\sum(x \in \coprod_{v \in V} M).\, \varphi(x)$. Then A is a monoid by assumption ($\varphi(e)$ holds since $\iota_v(e) = e$ holds). By assumption we also have a homomorphism $i_v \in M \to A$ for all $v \in V$, so we can apply the universal property to obtain a unique f' making the diagram:

$$\begin{tikzcd} M \arrow[r, "i_v"] \arrow[d, "\iota_v"'] & A \\ \coprod\limits_{v \in V} M \arrow[ur, "f'"'] & \end{tikzcd}$$

commute for all $v \in V$. Now, the projection $\pi_1 \in A \to \coprod_{v \in V} M$ is a homomorphism, which implies that $\pi_1(f'a) = a$ for all a since $\pi_1 \circ f'$ must be equal to identity by the universal property. But then the second projection π_2 applied to $f'(a)$ gives a proof of $\varphi(\pi_1(f'(a)))$ which is equal to $\varphi(a)$ for any a.

$\square$

Moreover, we have a powerful way to prove properties about operators on $\coprod_{v \in V} M$. We call an n-ary operator $f(x_1, \ldots, x_n)$ on $\coprod_{v \in V} M$ *multi-linear* if for any $i \leq n$, the function $a_i \mapsto f(a_1, \ldots, a_i, \ldots, a_n)$ is a homomorphism for any $a_i \in \coprod_{v \in V} M$. The following lemma will be very useful in section 2.7:

Lemma 2.6.3 *If $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ both are multi-linear operators on $\coprod_{v \in V} M$, then $f = g$ whenever one can prove*

$$f(\iota_{v_1}(m_1), \ldots, \iota_{v_n}(m_n)) = g(\iota_{v_1}(m_1), \ldots, \iota_{v_n}(m_n))$$

for all $v_i \in V$ and all $m_i \in M$.

Proof. By induction on n: If $n = 1$, then $f = g$ follows by uniqueness of the function f' defined from the universal property as

$$\begin{tikzcd} M \arrow[r, "h_v"] \arrow[d, "\iota_v"'] & H \\ \coprod_{v \in V} M \arrow[ur, "f'"'] & \end{tikzcd}$$

where $h_v(m) = f(\iota_v(m))$. If $n = k + 1$, then assume $a_1, \ldots, a_{k+1}$ and

$$f(\iota_{v_1}(m_1), \ldots, \iota_{v_{k+1}}(m_{k+1})) = g(\iota_{v_1}(m_1), \ldots, \iota_{v_{k+1}}(m_{k+1}))$$

for all $v_i \in V$ and all $m_i \in M$. From this, and the assumption that f and g are multi-linear, theorem 2.6.2 implies

$$f(\iota_{v_1}(m_1), \ldots, \iota_{v_k}(m_k), a) = g(\iota_{v_1}(m_1), \ldots, \iota_{v_k}(m_k), a)$$

for any a. By fixing a we can consider f and g as k-ary operators, so the induction hypothesis implies $f = g$. □

2.6.1 Association Lists

A simple instance of this is obtained by considering *association lists* with an ordering on the keys. Given an ordered set K of keys and a monoid V of values, the association-lists $\mathrm{AL}_{K,V}$ is defined as those lists of pairs of K and V, $\mathit{List}(K \times V)$, where the lists are sorted in their first components and the second components are non-unit. This restriction of the lists makes the desired equality to be the structural equality, i.e. two lists are equal if they have equal length and have equal components in all positions.

We will require a lookup function $l[k]$, taking an association lists l and a key k, and returning the value v such that (k, v) is in l, or the unit e_V of there are no such (k, v) in l. It is directly defined by recursion on l:

$$\begin{array}{rcl} ([\,])[k] & = & e_V \\ ((k', v) : l)[k] & = & \begin{cases} v, & \text{if } k' = k \\ l[k], & \text{otherwise.} \end{cases} \end{array}$$

We will also require a merge operation, $xs \mathbin{+\!\!+} ys$, which merges the association lists xs and ys into an association list which contains every pair $(k, v \cdot v')$ for $(k, v) \in xs$ and $(k, v') \in ys$, and every pair $(k, v) \in xs$ for which there are no v' such that $(k, v') \in ys$ and vice versa. It is defined by recursion on xs and ys:

$$\begin{array}{rcl} [\,] \mathbin{+\!\!+} ys & = & ys \\ xs \mathbin{+\!\!+} [\,] & = & xs \\ ((k_1, v_1) : xs) \mathbin{+\!\!+} ((k_2, v_2) : ys) & = & \begin{cases} (k_1, v_1) : xs \mathbin{+\!\!+} (k_2, v_2) : ys, & \text{if } k_1 < k_2 \\ (k_2, v_2) : (k_1, v_1) : xs \mathbin{+\!\!+} ys, & \text{if } k_1 > k_2 \\ (k_1, v_1 \cdot v_2) : xs \mathbin{+\!\!+} ys, & \text{if } k_1 = k_2 \text{ and} \\ & \quad v_1 \cdot v_2 \neq e_V \\ xs \mathbin{+\!\!+} ys, & \text{otherwise.} \end{cases} \end{array}$$

Theorem 2.6.4 *If M is a monoid and V is a linearly ordered set, then the set of association lists $\mathrm{AL}_{V,M}$ forms a monoid copower of M and V.*

To prove that $\langle \mathrm{AL}_{V,M}, +\!\!+, [\,] \rangle$ is a monoid, we use two general lemmas:

Lemma 2.6.5 *$M^V = V \to M$ has a monoid structure if V is a set and M is a monoid.*

Proof. We define the monoid to be $\langle R^V, =, \cdot, e \rangle$, where $f = g$ if $\forall x \in V.\, f(x) =_M g(x)$, $(f \cdot g)(x) = f(x) \cdot_M g(x)$, and $e(x) = e_M$. It is direct to check that this is a monoid if R is. □

```
FunMonoid (V::Set)(M::AbMonoid) :: AbMonoid = ...
```

Lemma 2.6.6 *If M is a set with an equality $=$, an operation $\cdot$, and a unit e, N is a monoid, and $f \in M \to N$ is a function which*

- *is injective,*
- *respects equality in M and N, and*
- *is a monoid-homomorphism w.r.t $\cdot$ and e, i.e.*
  - $f(e) = e_N$,
  - $f(a \cdot b) = f(a) \cdot_N f(b)$,

  then $\langle M, =, \cdot, e \rangle$ is a monoid.

Proof. Rather direct, by projecting the properties of N onto M using f. □

```
genMon (N::AbMonoid)
       (M::Set)(eqM::Rel M)(opM::BinOp M)(eM::M)
       (f::M->N.M)
       (oof::injective M N.M eqM N.(==) f)
       (respf::respEqFun M N.M eqM N.(==) f)
       (pf::(a,b::M)->N.(==) (f (opM a b)) (N.op (f a) (f b)))
       (qf::N.(==) (f eM) N.e)
  :: isAbMonoid M eqM opM eM = ...
```

Proof of theorem 2.6.4. By lemma 2.6.5 above, M^V is a monoid. To prove that $\langle \mathrm{AL}_{V,M}, +\!\!+, [\,] \rangle$ is a monoid, we use lemma 2.6.6 by constructing a function $f \in \mathrm{AL}_{V,M} \to M^V$:

$$f([\,])(v) = l[v],$$

and proving that this function is injective, respects equality and preserves $(+\!\!+)$:

$$\forall l_1, l_2 \in \mathrm{AL}_{V,M}.\, (\forall v \in V.\, l_1[v] = l_2[v]) \Rightarrow l_1 = l_2,$$

$$\forall l_1, l_2 \in \mathrm{AL}_{V,M}.\, l_1 = l_2 \Rightarrow (\forall v \in V.\, l_1[v] = l_2[v]),$$

$$\forall l_1, l_2 \in \mathrm{AL}_{V,M}.\, \forall v \in V.\, (l_1 +\!\!+ l_2)[v] = l_2[v] \cdot l_2[v],$$

by induction on l_1 and l_2. Finally, f respects $[\,]$ by definition: $\forall k \in V.\, f([\,])(k) = ([\,])[k] = e$.

Next we define $\iota_v(m) = [(v, m)]$ if $m \neq e$ and $\iota_v(m) = [\,]$ otherwise. It is direct to see that ι_v is a homomorphism for all v.

Finally for the universal property, assume H is a monoid with a family $f_v \in M \Rightarrow H$. Define $f' \in \mathrm{AL}_{V,M} \Rightarrow H$ by recursion:

$$\begin{array}{rcl} f'([\,]) & = & e_H, \\ f'((v, m) : l) & = & f_v(m) + f'(l). \end{array}$$

To prove that this is a homomorphism, we prove $f'(xs +\!\!+\, ys) = f'(xs) + f'(ys)$ by induction on the sum of the lengths of xs and ys. It is direct to check that $f'(\iota_v(m)) = f_v(m)$. To prove uniqueness, i.e. for any $g \in \mathrm{AL}_{V,M} \Rightarrow H$ such that $g(\iota_v(m)) = f_v(m)$, we have $\forall x \in \mathrm{AL}_{V,M}.\, f'(x) = g(x)$, we use induction on the list x. Note that the equality on ordered association-lists is just the structural equality on lists of pairs, and thus a decidable equivalence relation since it is defined from the decidable equivalence relations in M and V. $\square$

From now on, we will not use a particular representation. Instead we abstractly assume a Monoid Copower, and use the universal property to define and reason about functions. In the end, one can instantiate the development with this representation. This makes the rest of the notes independent of this particular representation, and hopefully less complicated.

2.7 The Free Monoid R-Algebra

This gives the structure of the polynomial ring. In the rest of this section, "algebra" refers to R-algebra. If M and N are algebras, we write $M \Longrightarrow N$ for the set of algebra-homomorphisms between M and N.

Definition 2.7.1 $R[M]$ is a *free monoid algebra of a monoid* M if R is a ring, and

1. $R[M]$ is an algebra,
2. there exists a homomorphism $\mathrm{inj} \in M \Longrightarrow R[M]$,
3. for any algebra H and homomorphism $f \in M \Rightarrow H$ (monoid homomorphism w.r.t. the multiplicative monoid in H), there exists a unique algebra homomorphism $f' \in R[M] \Longrightarrow H$ such that

$$\begin{tikzcd} M \arrow[r, "f"] \arrow[d, "\mathrm{inj}"'] & H \\ R[M] \arrow[ur, "f'"'] & \end{tikzcd}$$

commutes.

Abstractly, we want to capture the fact that $\mathrm{AL}_{M,R}$ forms a free monoid algebra when M is a monoid and R is a ring. In fact, using the universal property for monoid copowers, we can prove the following theorem:

Theorem 2.7.2 *If $R = \langle R, +_R, -_R, \cdot_R, 0_R, 1_R \rangle$ is a ring and $M = \langle M, +_M, 0_M \rangle$ is a monoid, then any Monoid Copower $R[M] = \coprod_{m \in M} R$ of R and M forms a Free Monoid Algebra.*

Proof. First, we are supposed to prove that $R[M]$ is an algebra. By definition, $R[M]$ forms a monoid $\langle R[M], +_{R[M]}, 0_{R[M]}\rangle$. What is left to prove is:

1. $R[M]$ has an inverse $(-_{R[M]})$ w.r.t. $(+_{R[M]})$.

   This is constructed from the universal property in $R[M]$. Define functions $f_v(r) = \iota_v(-r)$, $g_v(r) = \iota_v(0)$ and $h_v(r) = \iota_v(r)$, then by universality in Def. 2.6.1, there exists corresponding functions (Inv), $(Zero)$ and (Id), such that the diagrams

$$\begin{tikzcd} R \arrow[r, "f_v"] \arrow[d, "\iota_v"'] & R[M] \\ R[M] \arrow[ur, "Inv"'] & \end{tikzcd} \quad \begin{tikzcd} R \arrow[r, "g_v"] \arrow[d, "\iota_v"'] & R[M] \\ R[M] \arrow[ur, "Zero"'] & \end{tikzcd} \quad \begin{tikzcd} R \arrow[r, "h_v"] \arrow[d, "\iota_v"'] & R[M] \\ R[M] \arrow[ur, "Id"'] & \end{tikzcd}$$

   all commutes. Now, $(-_{R[M]}) = Inv$ is the desired inverse for $(+)$. To see this, first note that $Zero(x) = 0$ and $Id(x) = x$ for all x. Define $+$ on functions as $(f+g)(x) = f(x) + g(x)$, the goal $\forall x \in R[M].\, x + (Inv(x)) = 0$ then follows from $Id + Inv = Zero$. This follows from the uniqueness in the universality of $Zero$, by proving $(Id+Inv)(\iota_v(r)) = Zero(\iota_v(r))$ for all v and r:

$$\begin{aligned} (Id + Inv)(\iota_v(r)) &= Zero(\iota_v(r)) = \iota_v(0), \\ Id(\iota_v(r)) + Inv(\iota_v(r)) &= \iota_v(0), \\ \iota_v(r) + \iota_v(-r) &= \iota_v(0), \\ \iota_v(r + (-r)) &= \iota_v(0). \end{aligned}$$

2. $R[M]$ has a scalar multiplication $\alpha \cdot a \in R[M]$, for $\alpha \in R$ and $a \in R[M]$. We define a more general operation $Scal_{v,r}(a) \in R[M]$ for $v \in M$, $r \in R$ and $a \in R[M]$. We fix arguments v and r and define $Scal_{v,r}$ by applying the universal property to the function $f_u(s) = \iota_{vu}(rs)$, making the diagram

$$\begin{tikzcd} R \arrow[r, "f_u"] \arrow[d, "\iota_u"'] & R[M] \\ R[M] \arrow[ur, "Scal_{v,r}"'] & \end{tikzcd}$$

   commute. By defining $\alpha \cdot_{R[M]} a = Scal_{1,\alpha}(a)$, one can use the universal property, similarly as for the inverse above, to prove that for all $\alpha, \beta \in R$ and $a, b \in R[M]$,

   - $\alpha \cdot (a + b) = \alpha \cdot a + \alpha \cdot b$,
     this follows directly since $Scal_{1,r}((-))$ is a homomorphism,
   - $(\alpha + \beta) \cdot a = \alpha \cdot a + \beta \cdot a$,
     by uniqueness of universal maps, it is enough to consider $a = \iota_u(r)$:

$$\begin{aligned} (\alpha + \beta) \cdot \iota_u(r) &= \iota_u((\alpha + \beta)r) = \iota_u(\alpha r + \beta r) = \iota_u(\alpha r) + \iota_u(\beta r) \\ &= \alpha \cdot \iota_u(r) + \beta \cdot \iota_u(r). \end{aligned}$$

- $(\alpha\beta) \cdot a = \alpha \cdot (\beta \cdot a)$,
  again, by uniqueness of universal maps, it is enough to consider $a = \iota_u(r)$:

$$(\alpha\beta) \cdot \iota_u(r) = \iota_u((\alpha\beta)r) = \iota_u(\alpha(\beta r)) = \alpha \cdot (\iota_u(\beta r)) = \alpha \cdot (\beta \cdot (\iota_u(r))).$$

- $1 \cdot a = a$, by universality, it is enough to consider $a = \iota_u(r)$:

$$1 \cdot \iota_u(r) = \iota_u(1 \cdot r) = \iota_u(r).$$

3. $R[M]$ has multiplication $*$, defined by fixing a and using the function $f_u(r) = Scal_{u,r}(a)$ and the universal property:

$$\begin{tikzcd} R \arrow[r, "f_u"] \arrow[d, "\iota_u"'] & R[M] \\ R[M] \arrow[ur, "Mul_a"'] & \end{tikzcd}$$

Multiplication is then defined as $a *_{R[M]} b = Mul_a(b)$, with a unit defined as $1_{R[M]} = \iota_0(1)$. We prove that

- $\langle R[M], *, 1\rangle$ is an Abelian monoid,
  (a) $*$ is associative: $a * (b * c) = (a * b) * c$,
  By lemma 2.6.3, since both $a * (b * c)$ and $(a * b) * c$ are multi-linear in a, b and c, this follows from

$$\begin{aligned} (\iota_u(r)) * ((\iota_v(s)) * (\iota_x(t))) &= (\iota_{u+(v+x)}(r(st))) = (\iota_{(u+v)+x}((rs)t)) \\ &= ((\iota_u(r)) * (\iota_v(s))) * (\iota_x(t)), \end{aligned}$$

  (b) $*$ is commutative: $a * b = b * a$,
  By lemma 2.6.3, since both $a * b$ and $b * a$ are multi-linear in a and b, this follows from

$$(\iota_u(r)) * (\iota_v(s)) = (\iota_{u+v}(rs)) = (\iota_{v+u}(sr)) = (\iota_v(s)) * (\iota_u(r)),$$

  (c) 1 is unit to $*$: $1 * a = a$,
  By lemma 2.6.3, since both $1 * a$ and a are multi-linear in a, this follows from

$$1 * (\iota_u(r)) = (\iota_0(1)) * (\iota_u(r)) = (\iota_{0+u}(1r)) = (\iota_u(r)).$$

- $*$ distributes over $+$: $a * (b + c) = a * b + a * c$,
  Direct, since Mul_a is a monoid homomorphism.
- $\cdot$ associates with $*$: $\alpha \cdot (b * c) = (\alpha \cdot b) * c$,
  By lemma 2.6.3, since both $\alpha \cdot (b * c)$ and $(\alpha \cdot b) * c$ are multi-linear in b and c, this follows from

$$\alpha \cdot (\iota_u(r) * \iota_v(s)) = \alpha \cdot (\iota_{u+v}(rs)) = \iota_{u+v}(\alpha(rs)) = \iota_u(\alpha r) * \iota_v(s).$$

Then we define $\mathrm{inj}(v) = \iota_v(1)$. Using the definition of multiplication, one can directly prove that this is a homomorphism into the multiplicative monoid of $R[M]$.

Finally, given an algebra N, and monoid homomorphism $f \in M \Rightarrow N$, into the multiplicative monoid of N, we are supposed to construct an algebra homomorphism $f' \in R[M] \Longrightarrow N$, making

$$\begin{tikzcd} M \arrow[r, "f"] \arrow[d, "\mathrm{inj}"'] & N \\ R[M] \arrow[ur, "f'"'] & \end{tikzcd}$$

commute. We use the universality of the Monoid Copower with the function $g_v(r) = r \cdot (f(v))$. This yields a monoid homomorphism $f' \in R[M] \Rightarrow N$ such that

$$\begin{tikzcd} M \arrow[r, "g_v"] \arrow[d, "\iota_v"'] & N \\ R[M] \arrow[ur, "f'"'] & \end{tikzcd}$$

commutes for all $v \in M$. Then in particular for all $v \in M$

$$f'(\mathrm{inj}(v)) = f'(\iota_v(1)) = g_v(1) = 1 \cdot (f(v)) = f(v).$$

Now we only need to verify that f' is an algebra homomorphism. By the Monoid Copower, it is a monoid homomorphism w.r.t. the additive monoid, so what is left to prove is:

- $f'(r \cdot x) = r \cdot f'(x)$,

  By universality, it is enough to prove $f'(r \cdot \iota_v(a)) = r \cdot f'(\iota_v(a))$ for all $v \in M$ and $a \in R$. This follows from the definition of $\cdot$ and f'.

- $f'(x * y) = f'(x) * f'(y)$,

  By lemma 2.6.3, since both $f'(x * y)$ and $f'(x) * f'(y)$ are multi-linear in x and y, this follows from

$$\begin{aligned} f'(\iota_u(x) * \iota_v(y)) &= f'(\iota_{u+v}(xy)) = (xy) \cdot f(u+v) \\ &= (xy) \cdot (f(u) * f(v)) = (x \cdot f(u)) * (y \cdot f(v)) \\ &= f'(\iota_u(x)) * f'(\iota_v(y)). \end{aligned}$$

- $f'(1) = 1$,

  since $f'(1) = f'(\iota_0(1)) = g_0(1) = 1 \cdot f(0) = 1 \cdot 1 = 1$, because f is a monoid homomorphism into the multiplicative monoid of N.

$\square$

The proof of this lemma has not been completely formalised in Agda. We have omitted several simple but tedious proofs, such as trivial algebraic equalities, and focused on the structure of the proof.

2.8 $R[\mathbf{X}]$ is a free R-algebra

Given a way to construct monoid copowers, for example association lists as in section 2.6, we first construct the monomials Mon as the monoid copower $\coprod_{v \in V} \mathbb{N}$ for $V = X_1, \ldots, X_n$, and the monoid $\langle \mathbb{N}, +, 0 \rangle$. Then we construct the polynomials as the monoid copower $R[\text{Mon}] = \coprod_{m \in \text{Mon}} R$ for the monoid $\langle R, +, 0 \rangle$. By theorem 2.7.2, this is a free monoid R-algebra, and by composition a free R-algebra over $X_1, \ldots, X_n$.

2.9 Conclusions

We have given an abstract development of the polynomial ring, structured using universal properties and the notion of monoid copower which is iterated to first obtain monomials and then polynomials. This structuring is essentially the one given in textbooks on algebra such as [Lan93] and the abstractions mechanisms available in Agda is very suitable for expressing this development formally.

A problem with this formalisation is that the use of universal property to define functions sometimes leads to functions which are less efficient than those defined with a direct recursive definition. For example, when using the universal property to define the inverse operator on polynomials, and when using association lists as the monoid copower, $-([(c_1 m_1), \ldots, (c_k, m_k)])$ computes to $[(-c_1 m_1)] +\!\!+ \cdots +\!\!+ [(-c_k, m_k)]$. This function is of much worse complexity than the natural structural recursively defined inverse.

However, one can easily prove that these two inverses are extensionally equal, and hence all extensional properties proved for the inefficient inverse holds for the efficient inverse as well (and they can be directly proved in intensional type theory). We can consider these inefficient functions as a kind of prototype functions which are simple to define and prove theorems about. Furthermore, there are other possible representations of the monoid copower with an efficient binary operator $+\!\!+$, for example using trees instead of lists. In those cases the first algorithms obtained may have a satisfactory complexity.

We believe this work shows some of the benefits of finding the inherent structures of objects, and abstracting over such structures. In this particular case of polynomials, mathematicians have already discovered an inherent structure for us, but we feel this work motivates the search for structures and use of abstractions also in other areas.

We believe this abstract structured formalisation is on the right way towards a "final" formalisation of the polynomial ring which will survive the test of time. Still, there seems to be room for more improvements. One interesting possibility for improvement would be to use category theory explicitly in the formalisation. For example, the monoid copower and free monoid algebra could be defined as initial objects in certain categories. The use of some category theory might improve the structure of the formalisation while preserving the size of the formalisation, whereas too much may lead to new proof-obligations which makes the formalisation more complex.

Appendix A

Classical Proofs

A.1 A Classical Proof of Dickson's Lemma

Dickson's lemma has a short classical proof which uses a *minimal bad sequence argument* [NW63]:

Proposition A.1.1 (Dickson's lemma) *For all $n \in \mathbb{N}$, $\leq^n$ is well.*

Proof. By induction on n: If $n = 0$, it is trivial, so assume $n = m + 1$ and $\leq^m$ is well. We prove that if $<$ is well–founded and R is well, then $\leq \times R$ is also well, where $a \leq b$ is defined as $\neg(b < a)$. The proof is by contradiction: if $\leq \times R$ is not well, then there exists an infinite sequence $u_1, u_2, \ldots$ which is *bad*, i.e. for no $i < j$, $u_i \, (\leq \times R) \, u_j$. Using the axiom of dependent choices, we can construct a minimal bad sequence $v_1, v_2, \ldots$ in the following way: choose $v_1 = (x_1, w_1)$ with x_1 minimal among the first components of those pairs which starts a bad sequence. When $v_1, \ldots, v_k$ has been chosen, choose $v_{k+1} = (x_{k+1}, w_{k+1})$ with x_{k+1} minimal among the first components of those tuples continuing a bad sequence from $v_1, \ldots, v_k$. The existence of such minimal bad sequence is the main non–constructive part of the proof.

Since R is well by assumption, there exists $i < j$ such that $w_i \, R \, w_j$. But by construction, we must have $x_i \leq x_j$, otherwise $x_j w_i$ would continue a smaller bad sequence. Hence $(x_i, w_i) \, (\leq \times R) \, (x_j, w_j)$, which contradicts that $v_1, v_2, \ldots$ is a bad sequence. □

A.2 A Classical Proof of Hilbert's Basis Theorem

We give a short classical proof of HBT taken from [BW93], which uses two of the equivalent conditions in Definition 1.3.1:

Proposition A.2.1 *If R is a noetherian ring, then so is $R[X_1, \ldots, X_n]$.*

Proof. Since $R[X]$ is a ring if R is, it is enough to consider the case $n = 1$, the other cases follow by induction. Assume for a contradiction that I is an ideal of $R[X]$ and I is not finitely generated. Then I is not the zero ideal. We construct an infinite sequence $(f_i)_{i \in \mathbb{N}}$ of polynomials in I using the axiom of dependent choices:

1. f_0 is a non-zero polynomial in I of minimal degree.

2. f_{i+1} is a polynomial in $I \setminus \mathrm{Idl}(f_0, \ldots, f_i)$ of minimal degree.

It is clear that $\deg f_j \leq \deg f_k$ if $j < k$. We denote the (non–zero) head coefficient of f_i by a_i. Since R is noetherian, there exists an m such that $a_m \in \mathrm{Idl}(a_0, \ldots, a_{m-1})$, i.e. $a_m = r_0 a_0 + \cdots + r_{m-1} a_{m-1}$ where $r_0, \ldots, r_{m-1} \in R$. But then the polynomial

$$f^* = f_m - \sum_{i=0}^{m-1} X^{\deg f_m - \deg f_i} r_i f_i$$

must be in $I \setminus \mathrm{Idl}(f_0, \ldots, f_{m-1})$ since otherwise $f_m \in \mathrm{Idl}(f_0, \ldots, f_{m-1})$. But $\deg f^* < \deg f_m$, contradicting the minimality of f_m. □

Appendix B

Formal Proofs in Agda

Here we present some formal proofs in Agda [Coq98], a type-checker in the ALF-family for a variant of Martin-Löf's type theory. This type theory is very similar to Cayenne [Aug98], a functional programming language with dependent types.

B.1 Dickson's lemma in Agda

```
-- Theory of Dickson's lemma, takes a wellfounded relation gt and a well relation R
-- and proves leq x R (product of the two relations) to be a well relation.
--
package thDickson (A,B::Set)(gt::Rel A)(R::Rel B)
                  (wfgt::WF A gt)(dgt::decRel A gt)(gR::WR B R) =
 let
  leq (a,b::A) :: Set = Not (gt a b)
  leqxR (x,y::A*B) :: Set = (leq x.fst y.fst) & (R x.snd y.snd)

  GBarlR :: Pred (List (A*B)) = GRBar (A*B) leqxR
 in
 open OpenIndGoodRel (A*B) (fstR A B gt) leqxR (WFlem1 A gt wfgt (A*B) (fstR A B gt)
                     (Fst A B) (\(x,y::A*B) -> \(h::gt x.fst y.fst) -> h))
      use Min, open_ind
 in
 struct
 (:) (x::A*B)(l::List (A*B)) :: List (A*B) = @Cons x l
 sndL :: Fun (List (A*B)) (List B) = let {f (a::A*B) :: B  = a.snd} in map (A*B) B f

 lem0 (l::List (A*B))(a::A*B)(h1::ExistsL B (\(x::B) -> R x a.snd) (sndL l))(h::Min l)
   :: GBarlR (a:l) =
  let lem (vs::List (A*B))(h::ExistsL B (\(x1::B) -> R x1 a.snd) (sndL vs))(h1::Min vs)
          :: GBarlR (a:vs) =
       case vs of
        (Nil)  -> case h of { }
        (Cons a1 as) -> case h of
                          (Inl x) -> case dgt a1.fst a.fst of
                                       (No no) ->  @Base (@Inl (@Inl
                                                     (struct{fst :: leq a1.fst a.fst = no;
                                                             snd :: R a1.snd a.snd = x })))
                                       (Yes a') -> GRBarmon (A*B) leqxR (a:@Nil) as
```

```
                                                   (h1.fst a a') (a1:@Nil)
                         (Inr y) -> GRBarmon (A*B) leqxR (a:@Nil) as
                                            (lem as y h1.snd) (a1:@Nil)
  in lem l h1 h

 lem1 (us::List (A*B))(h1::GoodR B R (sndL us))(h::Min us) :: GBarlR us =
  case us of
   (Nil) ->  case h1 of { }
   (Cons a as) -> case h1 of
                   (Inl x) -> case as of
                               (Nil) ->  case x of { }
                               (Cons a' as') ->  lem0 (a':as') a x h.snd
                   (Inr y) -> GRBarmon (A*B) leqxR @Nil as
                                      (lem1 (append (A*B) @Nil as) y h.snd) (a:@Nil)

 keylem (us::List (A*B))(h::GRBar B R (sndL us)) :: (h1::Min us)-> GBarlR us =
  case h of
    (Base h1) -> lem1 us h1
    (Ind f) -> \(h1::Min us)->open_ind us h1 (\(u::A*B) -> keylem (u:us) (f u.snd))

 keylem_cor :: WR (A*B) leqxR = keylem @Nil gR @tt

-- This is the general version of Dickson's lemma.
--
Dickson (A::Set)(gt::Rel A)(wfgt::WF A gt)(dgt::decRel A gt)(n::Nat)
 :: WR (Vec A n) (VecRel A (NotR A gt) n) =
 case n of
  (Zero)  -> @Ind (\(a::Vec A @Zero)->
                      @Ind (\(a1::Vec A @Zero)-> @Base (@Inl (@Inl @tt))))
  (Succ n1) -> let package thD = thDickson A (Vec A n1) gt (VecRel A (NotR A gt) n1)
                                        wfgt dgt (Dickson A gt wfgt dgt n1)
               in thD.keylem_cor
```

B.2 Abstract Existence proof of Gröbner bases

Below is a formal proof of the existence of Gröbner bases for ideals in an arbitrary polynomial ring (`Poly`). We do not need to assume any properties of a polynomial ring at this level. The development is general and captures the reasoning in Section 1.2; for instance the assumption `bars::GBar @Nil` can be instantiated by the formal proof of Dickson's lemma above. Some parts of the proof terms were omitted to improve the presentation, these are denoted by "...".

```
package GB0  (Poly::Set)
             (Z::Poly)
             (plusP, timesP::BinOp Poly)
             ((==)::Rel Poly)
             (P::Poly->Pred (List Poly)) =
 let LP :: Set = List Poly
     ForallLP :: Pred Poly -> Pred LP = ForallL Poly
     ExistsLP :: Pred Poly -> Pred LP = ExistsL Poly
     Good::Pred (List Poly) = GoodP Poly P
```

```
    GBar :: Pred LP = Bar Poly Good
    Bad :: Pred LP = NotP LP Good
    (:) (f::Poly)(fs::LP) :: LP = @Cons f fs
in
open pkIdeal Poly (==) plusP timesP Z use Ideal, eqI, eqvI, congI, ilem1, ilem2
in
struct
package GB1 (spols::LP -> LP)
            (GB::Pred LP)
            (RED::Poly -> LP -> Poly)
            (deceq::decRel Poly (==))
            (ispol::(fs::LP)->ForallLP (\(g::Poly)-> Ideal fs g) (spols fs))
            (iRED::(f::Poly)->(fs::LP)->Ideal (f:fs) (RED f fs))
            (iREDP::(g::Poly)->(gs::LP)->P (RED g gs) gs -> RED g gs == Z)
            (Pprop::(f::Poly)->(fs::LP)->Exists Poly (\(g::Poly) ->
                                         eqI (f:fs) (g:fs) & Not (P g fs)))
            (gbchar::(gs::LP)->ForallLP (\(g'::Poly) -> RED g' gs == Z)
                                     (spols gs) -> GB gs)
            (bars::GBar @Nil)
 = open eqvI use ire, isy, itr
   in
   struct
   lem1 (fs::LP) :: Or (ExistsLP (\(h :: Poly) -> Not (RED h fs == Z)) (spols fs))
                       (ForallLP (\(h :: Poly) ->  RED h fs == Z) (spols fs))
         = existsLlem1 Poly (\(h :: Poly) -> RED h fs == Z)
                            (\(x :: Poly) -> deceq (RED x fs) Z) (spols fs)

   badlem (gs::LP)(g::Poly)(h1::Not (RED g gs == Z))(b::Bad gs)::Bad ((RED g gs):gs) =
    \(x :: Good ((RED g gs):gs)) -> case x of
                                        (Inl x') -> h1 (iREDP g gs x')
                                        (Inr y) -> b y

   remEq1 (fs::LP)::ForallLP (\(g' :: Poly) -> eqI ((RED g' fs):fs) fs) (spols fs) =
    let l1 (sfs::LP)
           (f1::ForallLP (\(g::Poly)-> Ideal fs g) sfs)
         :: ForallLP (\(g'::Poly) -> eqI ((RED g' fs):fs) fs) sfs =
         case sfs of
          (Nil) -> @tt
          (Cons a as) ->  ...
    in l1 (spols fs) (ispol fs)

   GBof (F,G::LP) :: Set = eqI G F & GB G

   thm (fs::LP)(gb::GBar fs)(b::Bad fs) :: Exists LP (GBof fs) =
    case gb of
     (Base h) ->  elimN0 (Exists LP (GBof fs)) (b h)
     (Ind f) ->
       case lem1 fs of
         (Inl x) ->  let l1 (sgs::LP)
                            (f2::ExistsLP (\(g'::Poly) -> Not (RED g' fs == Z)) sgs)
                            (f3::ForallLP (\(g'::Poly) -> eqI ((RED g' fs):fs) fs) sgs)
                            :: Exists LP (GBof fs) =
                               case sgs of
```

```
                             (Nil) ->  case f2 of { }
                             (Cons a as) ->  ...
                  in l1 (spols fs) x (remEq1 fs)
      (Inr y) ->  struct fst :: LP = fs
                         snd :: GBof fs fs = struct fst :: eqI fs fs = ire fs
                                                    snd :: GB fs = gbchar fs y

badprop (f::Poly)(fs::LP)(b::Bad fs)(np::Not (P f fs)) :: Bad (f:fs) =
  \(x :: Good (f:fs)) -> case x of
                            (Inl x') -> np x'
                            (Inr y) -> b y

exbad (fs::LP) :: Exists LP (\(gs::LP) -> Bad gs & eqI fs gs)
  = case fs of
        (Nil) ->  struct fst :: LP =  @Nil
                         snd :: Bad fst & (eqI @Nil fst) = ...
        (Cons a as) ->   ...

cor (fs::LP) :: Exists LP (GBof fs) =  let gs :: LP = (exbad fs).fst
                                           bad :: Bad gs = (exbad fs).snd.fst
                                           eq :: eqI fs gs = (exbad fs).snd.snd
                                           th :: Exists LP (GBof gs) = ...
                                       in ...
```

Bibliography

[Acz77] P. Aczel. An Introduction to Inductive Definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North-Holland Publishing Company, 1977.

[Aug98] L. Augustsson. Cayenne - a language with dependent types. Technical report, Department of Computing Science, Chalmers University of Technology, 1998. Homepage: http://www.cs.chalmers.se/˜ augustss/cayenne/.

[BS96] U. Berger and H. Schwichtenberg. The greatest common divisor: a case study for program extraction from classical proofs. In *Proceedings of the Workshop TYPES '95, Torino, Italy, June 1995*, number 1158 in Lecture Notes in Computer Science. Springer-Verlag, 1996.

[Buc65] B. Buchberger. *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal (German)*. PhD thesis, University of Innsbruck, 1965.

[Buc85] B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Multidimensional systems theory*, pages 184–232. Reidel Publ. Co., 1985.

[Buc98] B. Buchberger. Introduction to Gröbner bases. In B. Buchberger and F. Winkler, editors, *Gröbner bases and applications*, pages 3–31. Cambridge University Press, 1998.

[BW93] T. Becker and V. Weispfenning. *Gröbner bases*, volume 141 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1993. In cooperation with H. Kredel.

[CLO97] D. Cox, J. Little, and D. O'Shea. *Ideals, varieties, and algorithms*. Undergraduate Texts in Mathematics. Springer-Verlag, New York, second edition, 1997.

[Con86] R. L. Constable et al. *Implementing Mathematics with the NuPRL Proof Development System*. Prentice-Hall, Englewood Cliffs, NJ, 1986.

[Coq92] Th. Coquand. Constructive topology and combinatorics. In *proceeding of the conference Constructivity in Computer Science, San Antonio, LNCS 613*, pages 28–32, 1992.

[Coq98] C. Coquand. The homepage of the Agda type checker. Homepage: http://www.cs.chalmers.se/˜ catarina/Agda/, 1998.

[Dic13] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Am. J. Math.*, 35:413–422, 1913.

[Dyb90] P. Dybjer. Comparing integrated and external logics of functional programs. *Science of Computer Programming*, 14:59–79, 1990.

[Frö97] R. Fröberg. *An introduction to Gröbner bases.* John Wiley & Sons, 1997.

[Gir86] J-Y Girard. Linear logic and parallelism. In M. Venturini Zilli, editor, *Mathematical Models for the Semantics of Parallelism*, number LNCS 280, pages 166–182. Springer-Verlag, September 1986.

[HKPM97] G. Huet, G. Kahn, and C. Paulin-Mohring. The Coq proof assistant: A tutorial. Technical report, Rapport Technique 204, INRIA, 1997.

[Jac95] P. B. Jackson. *Enhancing the Nuprl proof development system and applying it to computational abstract algebra.* PhD thesis, Cornell University, 1995.

[JL91] C. Jacobsson and C. Löfwall. Standard bases for general coefficient rings and a new constructive proof of Hilbert's basis theorem. *J. Symbolic Comput.*, 12(3):337–371, 1991.

[Lan93] S. Lang. *Algebra.* Addison-Wesley, 3rd edition edition, 1993.

[ML68] P. Martin-Löf. *Notes on Constructive Mathematics.* Almqvist & Wiksell, 1968.

[ML71] S. Mac Lane. *Categories for the working mathematician.* Springer-Verlag, 1971.

[MRR88] R. Mines, F. Richman, and W. Ruitenburg. *A course in constructive algebra.* Universitext. Springer-Verlag, New York, 1988.

[NPS90] B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's Type Theory. An Introduction.* Oxford University Press, 1990.

[NW63] C. Nash-Williams. On well-quasi-ordering finite trees. *Proceedings of the Cambridge Philosophical Society*, 59:833–835, 1963.

[Pot96] Loïc Pottier. Dixon's lemma. URL: ftp://www.inria.fr/safir/pottier/MON, 1996.

[Rao88] J-C. Raoult. Proving open properties by induction. *Information processing letters*, 29:19–23, 1988.

[Ric74] F. Richman. Constructive aspects of noetherian rings. In *Proc. AMS 44*, pages 436–441, 1974.

[Thé98] L. Théry. Proving and computing: A certified version of the Buchberger's algorithm. In *Proceeding of the 15th International Conference on Automated Deduction, Lindau, Germany, LNAI 1421*, 1998.