Proof Certificates for Algebra and their Application to Automatic Geometry Theorem Proving

Loïc Pottier, Laurent Théry

Proof Certificates for Algebra and their Application to Automatic Geometry Theorem Proving

Benjamin Grégoire, Loïc Pottier, Laurent Théry

Marelle Project, INRIA Sophia Antipolis

Abstract. Integrating decision procedures in proof assistants in a safe way is a major challenge. In this paper, we describe how, starting from Hilbert's Nullstellensatz theorem, we combine a modified version of Buchberger's algorithm and some reflexive techniques to get an effective procedure that automatically produces formal proofs of theorems in geometry. The method is implemented in the Coq system but, since our specialised version of Buchberger's algorithm outputs explicit proof certificates, it could be easily adapted to other proof assistants.

Keywords: decision procedure, Nullstellensatz, geometry theorem proving, proof assistant

1 Introduction

Integrating decision procedures in proof assistants in a safe way is a major challenge. Many well-known and widely-used decision procedures exist but making them available in the context of a proof assistant is not so trivial: one has to certify the result of the procedure. This may explain, for example, why very few theorems of geometry have been formalised yet in the list compiled by Freek Wiedijk [28].

The integration can be made in several ways. Everything can be done inside the proof assistant. For example, one can write the procedure as a tactic, then, every time the tactic is called, a proof is built for the particular instance. Alternatively, if the system offers a programming language, one can write the entire procedure as a program inside the system and use standard program verification techniques to derive its correctness once and for all. In both cases, such an internal integration is usually very time consuming specially if the procedure is rather complex.

Another way to go is to use external programs. For example, one can take an existing implementation of the procedure and modifies it to output execution traces. Then, the proof assistant only needs to follow the information in the trace to build its own proof. An alternative is to use certificates instead of traces. Certificates do not contain all the execution path but just enough information to make it easy for the proof assistant to build a proof. Examples of such certificates are prime certificates that we have used [13] to get the primality of

large prime numbers, or algebraic certificates [17]. Verifying certificates is usually more difficult than checking traces but still an order of magnitude simpler than building internally the entire procedure. Verifying the certificates can be done either by tactics or by a verified program.

In this paper, we advocate the combination of an external program that generates certificates and a verified program that checks certificates for the particular case of deciding problems of the following kind:

$$\begin{array}{l}\forall X_1, \ldots, X_n \in R, \\ P_1(X_1, \ldots, X_n) = 0 \wedge \ldots \wedge P_s(X_1, \ldots, X_n) = 0 \\ \Rightarrow P(X_1, \ldots, X_n) = 0\end{array}$$

where R is a commutative ring without zero divisor and $P, P_1, \ldots, P_s$ are polynomials. This has been intensively studied during the 80s, specially by the computer algebra community. The main decision procedure used to solve these problems is Buchberger's algorithm [2]. A lot of efforts have been spent in improving this algorithm and trying to apply it to various domains of theorem proving. Without doubt, one of the most successful area of application is automatic geometry theorem proving (see for example [5, 20, 21, 26, 29, 22] for a survey). In the context of proof assistants, this has also recently drawn some attention. John Harrison [16] uses a basic implementation of Buchberger's algorithm to produce a certificate for elementary arithmetic (e.g. proving the Chinese remainder theorem) for the HolLight system [14]. Also, Amine Chaieb and Makarius Wenzel [3] use it in the context of the Isabelle system [24]. One of the authors [25] has also connected Coq with the state-of-the-art implementation of Gröbner bases algorithms.

In this paper, we explain how we manage to solve theorems of geometry proved by Wu's method: Desargues, Pascal, and 20 other theorems. Getting these theorems inside a proof assistant is a real challenge: the proof certificates we obtain for these theorems are quite large. Three key ingredients were essential:

- our implementation of Buchberger's algorithm that generates certificates does not try to compute the whole Gröbner basis. It just does enough work to solve the specific problem. In practice, this reduces drastically the time that is needed to generate certificates. Our approach is very pragmatic and lacks a more precise insight of what is exactly gained by progressively reducing the polynomials.
- our certificates are not just composed of a single polynomial identity but consist in *straight-line programs*. Such programs are composed of assignments only. No branching or loops are allowed. Straight-line programs are important tools used in algebraic complexity to prove optimal bounds [9]. Here, they appear to be also of great practical use.
- reflexive methods and the power of the reduction machine of the Coq system are used to verify these certificates efficiently.

A tactic in Coq is a piece of program that allows to prove automatically some specific kind of logical statements. It produces a proof term for this statement. The proof is then verified by Coq. If it is correct, the statement is validated

as a theorem. In this work, we have implemented and tested a new tactic for proving ideal membership. Its input is a statement with polynomial expressions as hypotheses and conclusion. Its output is a proof term that is automatically generated from a certificate (a list of polynomials lists). Another tactic has been developed for geometry. It reduces geometrical statements into their polynomial form. We can then compose the two tactics to prove geometrical statements.

The paper is organised as follows. In Section 2, we recall what the Nullstellensatz theorem and Gröbner bases are. In Section 3, we present our modified version of Buchberger's algorithm that generates certificates based on straight-line programs. In Section 4, we recall what reflexive methods in provers like Coq are and explain how they have been used in our context to produce short proof terms. Finally, in Section 5, we illustrate an application of our decision procedure to prove automatically some theorems of geometry taken from [5] and [6].

2 Nullstellensatz Theorem and Gröbner Basis

We seek to prove implications of the following form:

$$\begin{array}{l}\forall X_1, \ldots, X_n \in R, \\ P_1(X_1, \ldots, X_n) = 0 \wedge \ldots \wedge P_s(X_1, \ldots, X_n) = 0 \\ \Rightarrow P(X_1, \ldots, X_n) = 0\end{array}$$

where R is a commutative ring without zero divisor and $P_1, \ldots, P_s$ are polynomials. As a matter of fact, the general problem we want to solve is quantifier elimination in general rings and fields, but the general problem reduces easily to the problem we address here (see for example [23]). Hilbert's Nullstellensatz theorem shows how to reduce proofs of equalities on polynomials to algebraic computations (see for example [7] for the notions introduced in this section).

It is easy to see that if a polynomial P in $R[X_1, \ldots, X_n]$ verifies $cP^r = \sum_{i=1}^{s} Q_i P_i$, with $c \in R$, $c \neq 0$, r a positive integer, and the Q_is in $R[X_1, \ldots, X_n]$, then P is zero whenever polynomials $P_1, ..., P_s$ are zero. The converse is also true when R is an algebraic closed field: the method is complete. So, proving our initial problem reduces into finding $Q_1, \ldots, Q_s$, c and r such that $cP^r = \sum_i Q_i P_i$. In this case, we call $(c, r, Q_1, \ldots, Q_s)$ the "certificate" of the statement we want to prove since it is straightforward to obtain a proof from this certificate.

In this work, we concentrate on the special case where $r = 1$, i.e. the problem of finding $Q_1, \ldots, Q_s$ and c such that $cP = \sum_i Q_i P_i$. The cases $r > 1$ can be tested by enumeration, or by using extra variables, as explained in [25] for example. In practice, almost all problems are solved with $r = 1$.

2.1 Division of Polynomials

An *ideal* $\mathcal{I}$ of a ring is an additive subgroup of the ring such that $a \times x \in \mathcal{I}$ whenever $a \in \mathcal{I}$. The ideal *generated* by a family of polynomials is the set of all linear combinations of these polynomials (with polynomial coefficients). A *Gröbner basis* of an ideal is a set of polynomials of the ideal such that their head

monomials (relative to a chosen order on monomials, e.g. lexicographic order, or degree order) generate the ideal of head monomials of all polynomials in the ideal. The main property of a Gröbner basis is that it provides a test for the membership to the ideal: a polynomial is in the ideal if and only if its euclidean *division* by the polynomials of the basis gives a zero remainder.

The division process is a generalisation of the division of polynomials in one variable: to divide a polynomial P by a polynomial $aX^\alpha - Q$ we write $P = aX^\alpha S + T$ where T contains no monomial that is multiple of X^α. Then we change P into $QS+T$ and repeat the process. When we reach a polynomial that is not divisible by $aX^\alpha - Q$, this is the remainder of the division. For example, suppose that we use the degree order on monomials. The head monomial of $x^2 - z$ is then x^2. In order to divide $x^4y + x^2 - 1$ by the polynomial $x^2 - z$, we rewrite x^2 into z everywhere. This leads to the polynomial $z^2y + z - 1$, which is not divisible by $x^2 - z$: it is the remainder of this division process. In order to divide a polynomial by a family of polynomials, we repeat this process with each polynomial of the family.

2.2 Gröbner Bases

In general, the remainder of a division of a polynomial by a family of polynomials depends on the order in which we use the polynomials of the family. With a Gröbner basis, this remainder is unique: this is a characteristic property of Gröbner bases. For example, dividing $x^2y^2 - y^4$ by the family $\{x^2 + 1, xy - 1\}$ gives the remainder $-y^2 - y^4$ if we divide by $x^2 + 1$ but gives $1 - y^4$ if we divide by $xy - 1$. Both remainders are irreducible: so the family is not a Gröbner basis. We can prove that the family $\{x^2 + 1, xy - 1, x + y, y^2 + 1\}$ is a Gröbner basis of the ideal generated by $\{x^2 + 1, xy - 1\}$. Any division of $x^2y^2 - y^4$ by this family will give the remainder 0. With a simple division algorithm, we can conclude that the polynomial $x^2y^2 - y^4$ is in the ideal generated by $\{x^2 + 1, xy - 1\}$.

Consider now polynomials of the ideal with a degree in y strictly lower than 2 such as $x^3 - y$. Obviously, dividing them by only the three first polynomials $\{x^2+1, xy-1, x+y\}$ of the basis is sufficient to give the remainder 0. This shows that on some particular cases it is not necessary to have a complete Gröbner basis in order to conclude. Also, half of the time for computing a Gröbner basis is usually spent in verifications that do not produce any new elements for that basis. In practice, the strategy that consists in checking the membership each time a new polynomial is added to the family gives an effective speed-up.

3 Buchberger's Algorithm and Certificates

The main method for computing Gröbner bases is Buchberger's algorithm. It consists in completing the initial family of polynomials by new polynomials in the same ideal built from the so-called *S-polynomials*. For a pair of polynomials (P, Q), its associated S-polynomial is the polynomial $t_1P - t_2Q$ where the terms t_1 and t_2 are chosen in such a way that the head monomials of t_1P and t_2Q are

identical, so they cancel out in the subtraction. The algorithm starts with the initial family and adds all the non-zero remainders of the S-polynomials for all pairs of elements of the family. If no new element has been added to the family, the algorithm terminates otherwise it repeats the same completion process with the new family. Dixon's lemma ensures that this iterative process eventually terminates. The resulting family is then a Gröbner basis.

We modify this algorithm in a simple way: each time a non-zero remainder of a S-polynomial is added to the family, we divide the polynomial P by it, and replace P by its remainder. If this remainder is zero, the algorithm terminates. Remembering all the divisions that has been done from the beginning gives a way of writing P as a linear combination of the original polynomials. Let us take a concrete example. Suppose we want to show that $P_1 = 0 \wedge P_2 = 0 \Rightarrow P = 0$ with $P_1 = x^2 + 1$, $P_2 = xy - 1$ and $P = x^3 - y$. Our initial family is then $\{P_1, P_2\}$ and we want to find a certificate c,Q_1 and Q_2 such that $cP = Q_1P_1 + Q_2P_2$. We first try to divide P by the family $\{P_1, P_2\}$ starting from left to right. P_1 divides P and the remainder is $R_1 = -x - y = P - xP_1$. R_1 is non-zero and irreducible by $\{P_1, P_2\}$, so we can start the completion. There is only one S-polynomial for the family $\{P_1, P_2\}$, which is $P_3 = x + y = yP_1 - xP_2$. It is irreducible by $\{P_1, P_2\}$, so we add $P_3 = x + y$ to the family $\{P_1, P_2\}$. We then try to divide R_1 by P_3, this gives $0 = R_1 + P_3$ so we can stop the completion. We have

$$0 = R_1 + P_3 = (P - xP_1) + (yP_1 - xP_2)$$

singling P out gives

$$P = (x - y)P_1 + xP_2$$

so the certificate is $c = 1$, $Q_1 = x - y$ and $Q_2 = x$. Note that in order to get the certificate, we have not computed the complete basis.

For our certificates, we are not going to express P as a combination of the initial family as in the previous example. A more effective way of presenting the certificate is to be a bit closer to the computation that is actually performed by the algorithm. The certificate is then composed of two parts. The first part, CR, is a list of polynomials. It gives the coefficients to express P as a combination of the initial family plus the extra polynomials that the computation of the partial Gröbner basis has added to the initial family in order to reduce P to 0. The second part, C, is a list of polynomials lists. Each subsist corresponds to one extra polynomial and explains why it belongs to the ideal generated by the initial family plus the polynomials that are added before. More formally, with an initial family $\{P_1, \ldots, P_s\}$ this gives:

$$\begin{aligned} CR &= [c_1, \ldots, c_{s+p}] \\ C &= [[a_{1\ s+1}, \ldots, a_{s\ s+1}], \\ &\quad \ldots \\ &\quad [a_{1\ s+p}, \ldots, a_{s\ s+p}, \ldots, a_{s+p-1\ s+p}]] \end{aligned}$$

where

$$\forall i \in [1; p], P_{s+i} = a_{1\ s+i}P_1 + \ldots + a_{s+i-1\ s+i}P_{s+i-1}$$

and

$$P = -(c_1 P_1 + \ldots + c_{s+p} P_{s+p})$$

For simplicity here, we have assumed that R is a field (hence $c = 1$) but we can easily extend the certificate format to the case where divisions are pseudo-divisions. As each new polynomial in C is defined with respect to the previous ones, C has a structure that is very similar to straight-line programs. Applied to our example, we get the following "program":

$$\begin{array}{l} P_1 := x^2 + 1; \\ P_2 := xy - 1; \\ P_3 := yP_1 + (-x)P_2; \\ P := -((-x)P_1 + 0P_2 + 1P_3); \end{array}$$

so the certificate is $CR = [-x, 0, 1]$ and $C = [[y, -x]]$. The main advantage of straight-line programs is that they allow the sharing of computations. This can change the exponential computing time into linear one (see [9] for an example of how straight-line programs can be used to find complexity bound).

In general, a straight-line program is an imperative program without loops, i.e. a sequence of assignments of expressions to variables, each expression depending on previously assigned variables:

$$\begin{array}{l} x_1 := f_1(); \\ x_2 := f_2(x_1); \\ x_3 := f_3(x_1, x_2); \\ ... \\ x_n := f_n(x_1, ..., x_{n-1}); \end{array}$$

where $f_1, f_2, ..., f_n$ are parametric procedures. In computer algebra, they are usually rational fractions, here just polynomials. A straight-line program can be viewed as a directed acyclic graph, i.e. a tree with shared sub-trees.

To illustrate how this change of complexity may occur in our particular context, let us consider the following contrived example. Let f_n be the nth Fibonacci number: $f_0 = 0, f_1 = 1, f_{n+2} = f_{n+1} + f_n$ and suppose that we want to prove that $X^{f_{n+1}} - 1 = 0 \wedge X^{f_n} - 1 = 0 \Rightarrow X - 1 = 0$. Computing the Gröbner basis of $\{X^{f_{n+1}} - 1, X^{f_n} - 1\}$ mimics Euclid's algorithm for gcd and the decomposition is

$$X - 1 = P_{n-2}(X^{f_{n+1}} - 1) + P_{n-1}(X^{f_n} - 1) \tag{1}$$

where the polynomials P_n are defined by $P_0 = 0, P_1 = 1, P_n = -X^{f_n} P_{n-1} + P_{n-2}$. The certificate with straight-line programs is

$$\begin{array}{ll} CR = & [0, \ldots, 0, 1] \\ C = & [[1, -X^{f_{n-1}}], \\ & [0, 1, -X^{f_{n-2}}], \\ & \ldots \\ & [0, \ldots, 0, 1, -X^{f_2}]] \end{array} \tag{2}$$

```
inideal(P,F){
   (* F = [P1,...,Pn] *)
   R := P; C := [];CR := LR; R := R1;
   SP:= Spolynomials(F,F);
   let (R1,LR) = divide(R,F) in
   while R <> 0 do (* stop if P divides to 0 by F *)
     if SP = [] then Fail
        (* the Gröbner basis is computed without reducing P to 0 *)
     else let (S,LS)::SP1 = SP in
          SP := SP1;
          let (D,LD) = divide(S,F) in
          if D <> 0 (* add a new polynomial to F *)
            then F := F + [D];   (* + denotes concatenation of lists *)
                 SP := SP + Spolynomials(F,[D]);
                 C := C + [merge(LD,LS)];
                 let (R1,LR) = divide(R,F) in (* reduce R by F *)
                 CR := merge(CR,LR);
                 R := R1;
   done;
   return(CR,C);
}
```

Fig. 1. Pseudo-code of the modified Buchberger's algorithm with certificate

Suppose that in order to check a polynomial equality, one first applies distributivity and then collects equal monomials. Let us compare the verification of the two certificates (1) and (2) in term of operations on monomials. In the first one, P_n has degree $f_{n+2} - 2$ and has f_n monomials, with coefficients 1 (n odd) or -1 (n even), then the verification of this equation requires $2f_n$ multiplications, and $3f_n$ additions. In the second one, each intermediate polynomial has form $X^{f_k} - 1$, then the verification only requires $2n - 4$ multiplications and $4n - 8$ additions. As $f_n \sim ((1 + \sqrt{5})/2)^n/\sqrt{5}$, there is an exponential factor between the two.

We end this section by giving in Figure 1 the pseudo-code of the function `inideal` that generates our certificates. The function `divide` returns the remainder together with the list of quotients of the division. The function `Spolynomials` computes the S-polynomials of two families. For each of these S-polynomials, it also returns the monomials and polynomials used to compute them. The function `merge` adds terms of two lists of same rank, completing by zeros if needed. Our program is composed of 3500 lines of OCaml and 500 lines of Coq. It includes polynomial arithmetic for sparse polynomials and recursive polynomials, rational fractions, sub-resultant, gcd computation, and unbounded integer arithmetic. The OCaml code could easily be used as a standalone prover provided one adds a minimal parser/printer for polynomials.

4 Reflexive Method to Verify Large Certificates in Proof Assistant

In Coq system, each deduction step appears explicitly in the final proof term. Thus, each application of lemma (and in particular each rewriting step) is stored in the proof. This clearly prohibits the use of rewriting tactics to verify our certificates: proof terms would be too large. Fortunately, the Coq system integrates a programming language on which we can reason. For programs written in this language, symbolic evaluation is also possible via the reduction mechanism. More importantly, this reduction mechanism is integrated inside the logic: two terms are considered equal if their normal forms, i.e. the terms after evaluation, are structurally equal. So, reductions do not appear in proof terms. The reflexive method introduced by Allen et al. [1] takes advantage of this reduction mechanism in order to reduce drastically the size of proof terms. Note that if the reduction mechanism is particularly efficient, using reflexive methods can also reduce the time required to verify the proof. The reflexive method relies on the following remark:

- Let $P : A \to \mathsf{Prop}$ be a predicate over a set A.
- Assume we are able to write in the system a program c such that the following properties holds

  $$\mathsf{c\_spec} : \ \forall a,\ c\ a = \mathsf{true} \to P\ a$$

  In other words, for all value a, if the evaluation of the program c on a returns true then P is satisfied for a. This means that c is a semi decision procedure for the properties P and $\mathsf{c\_spec}$ is the lemma which expresses that the semi decision procedure is correct.

Now, assume that we have to prove $P\ a'$ for a specific a' and that, for this particular a', the system is able to reduce $c\ a'$ into true. In order to prove $P\ a'$, we can apply the lemma $\mathsf{c\_spec}$ to a', so we are left with $c\ a' = \mathsf{true}$ to prove. Since $c\ a'$ reduces to true, this proposition $c\ a' = \mathsf{true}$ is identical for the prover to the proposition $\mathsf{true} = \mathsf{true}$ which can be proved by the reflexivity of equality.

The Coq system is based on the Curry-Howard isomorphism. This means that proofs are represented by programs, propositions by types and valid proofs are well-typed programs. For example, our proof of $P\ a'$ is the program $\mathsf{c\_spec}\ a'\ (\mathsf{refleq}\ \mathsf{true})$. The typing derivation of the proof is:

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \mathsf{c\_spec}\ a' : c\ a' = \mathsf{true} \to P\ a'} \qquad \dfrac{\Gamma \vdash \mathsf{refleq}\ \mathsf{true} : \mathsf{true} = \mathsf{true} \quad \mathsf{true} = \mathsf{true} \equiv c\ a' = \mathsf{true}}{\Gamma \vdash \mathsf{refleq}\ \mathsf{true} : c\ a' = \mathsf{true}}[\textsc{Conv}]}{\Gamma \vdash \mathsf{c\_spec}\ a'\ (\mathsf{refleq}\ \mathsf{true}) : P\ a'}$$

Here the key point is the use of the Conv typing rule:

$$\dfrac{\Gamma \vdash t : T \qquad T \equiv U}{\Gamma \vdash t : U}[\textsc{Conv}]$$

which allows to view a program t of type T as a program of type U if T and U are equal modulo reduction (convertible). All the reduction steps that are necessary to check the convertibility of $\mathsf{true} = \mathsf{true}$ and $c\ a' = \mathsf{true}$ do not appear in the proof term. Naturally, if the reduction steps are not in the proof term, they are going to be performed during the checking phase. So the time needed to check a proof that uses the reflexive method will not only crucially depend on the efficiency of the reduction mechanism implemented by the prover but also on the efficiency of the semi-decision procedure. The compiled reduction mechanism of Coq has a very efficient strategy to reduce programs [11]. This makes the reflexive method very attractive in Coq.

Now that we have introduced the reflexive method, let us explain how it can be used to define a checker for the certificates defined in Section 3. For polynomial operations, we use the existing polynomial library of Coq [12]. This library has been developed for the reflexive `ring` tactic that proves equalities over an arbitrary ring structure. It defines two data-types:

- $\mathcal{E}$ represents the type of polynomial expressions, i.e. the free algebra;
- $\mathcal{P}$ represents the type of polynomials in Horner normal form.

Basic operations like addition or multiplication are defined on the type $\mathcal{P}$, thus it is easy to define a normalisation algorithm `norm` from $\mathcal{E}$ to $\mathcal{P}$ by structural recursion. Correctness of the basic operations is provided using an interpretation function $[\![\ ]\!]_\rho$ from $\mathcal{P}$ to an arbitrary ring R, where ρ is the valuation function that binds polynomial indeterminates to value in R. It is proved that each operator is correct with respect to the interpretation function. For example the specification of the addition is given by:

$$\forall \rho\ P_1\ P_2,\ [\![P_1 +_{\mathcal{P}} P_2]\!]_\rho = [\![P_1]\!]_\rho +_R [\![P_2]\!]_\rho$$

In a similar way, the correctness of the normalisation is defined using an interpretation function $[\![\ ]\!]^{\mathcal{E}}_\rho$ on polynomial expressions:

$$\forall \rho\ E,\ [\![E]\!]^{\mathcal{E}}_\rho = [\![\mathsf{norm}\ E]\!]_\rho$$

Thus, to prove that two ring expressions r_1 and r_2 in R are equal, it is sufficient to find two polynomial expressions E_1, E_2 and a valuation function ρ such that $[\![E_i]\!]^{\mathcal{E}}_\rho$ reduce to r_i. If the normalisation of E_1 and E_2 leads to the same Horner normal form then r_1 and r_2 are equal. This strategy is not necessarily the best one. The normalisation is defined by a naive structural recursion: in order to normalise $(X+Y)^{100} - (X+Y)^{100}$ the function first normalises twice the subterm $(X+Y)^{100}$ and then performs the subtraction. This is clearly not optimal.

Implementing our checker on top of that library is straightforward. We first define a function `mult_l` that normalises each line of the certificate. In the syntax of Coq this looks like

```
Function mult_l (Le L: list P) : P :=
  match Le, L with
  | e::L'e, p::L' ⇒ e *P p +P mult_l L'e L'
```

```
  | _, _ ⇒ 0_P
  end.
```

Then a second function `compute_list` collects all the normalised polynomials that correspond to the lines of the certificates

```
Function compute_list (LL_e: list (list P)) (L:list P): list P :=
  match LL_e with
  | L_e::LL_e ⇒ compute_list LL_e ((mult_l L_e L)::L)
  | _ ⇒ L
  end.
```

Finally the checking function `check` tests the equality of the two normal forms:

```
Function check (L_e:list E) (p:E) (certif: list (list P) * list P) :=
  let (LL_e, L'_e) := certif in
  let L := map norm L_e in
  norm p =?=_P mult_l L'_e (compute_list LL_e L).
```

Note that all the functions we have defined for our checker are tail-recursive. In order to prove the correctness of the checker, we first define the property for a list to be composed of only zero polynomials:

```
Definition Allzero ρ (L: list P) := ∀P ∈ L, [P]_ρ = 0.
Definition Allzero_E ρ (L_e: list E) := ∀P ∈ L_e, [P]^E_ρ = 0.
```

We then show that the two functions `mult_l` and `compute_list` behave well with list of zero polynomials:

```
Lemma mult_l_spec: ∀ρ L_e L, Allzero ρ L → [mult_l L_e L]_ρ = 0.
Lemma compute_list_spec:
  ∀ρ LL_e L, Allzero ρ L → Allzero ρ (compute_list LL_e L).
```

Finally, we can derive the correctness of our checker

```
Lemma check_correct:
  ∀ρ L p certif, check L p certif = true → Allzero_E ρ L → [p]^E_ρ = 0.
```

Defining the checker and proving its correctness is straightforward. This is exactly what we wanted: the integration of a decision procedure from the prover side should be as seamless as possible.

5 Geometry Theorem Proving

In his book [5], Shang-Ching Chou proves 512 theorems of geometry mechanically. In this section, we show how we have been capable to prove some of the most difficult ones in Coq with our certificates. We also compare our results with other systems (HOL Light [14] and Macaulay2 [10]).

In order to turn geometry into algebra, points are represented by their coordinates, geometric predicates by polynomials based on determinants, scalar products and algebraic relations between trigonometric functions. For example, we define collinearity in Coq by:

```
Definition collinear (A B C:point):=
  (X A - X B) * (Y C - Y B) - (Y A - Y B) * (X C - X B) = 0.
```

and the fact two lines defined by two pairs of points are parallel:

```
Definition parallel (A B C D:point):=
  (X A - X B) * (Y C - Y D) = (Y A - Y B) * (X C - X D).
```

Figure 2 gives a summary of some of our experiments. The machine used for these benchmarks is a Linux PC with dual Intel Xeon 3.2Ghz processors with 33Gb of memory. Columns contain respectively:

1. The name of the theorem and in parenthesis the page in Chou's book where it is stated (when it exists);
2. The time in seconds for computing the certificate,
3. The time in seconds for verifying the certificate,
4. The size in number of characters of c and the Q_i when expanding the certificate into $cP = \sum_i Q_i P_i$.
5. The size in number of characters of the certificate,
6. The size of the certificate as a proof term (number of nodes),
7. The size of the certificate as a optimized straight-line program: every subterm of the proof term is shared (with *let* operator).

A more detailed presentation of the examples is available at

`http://www-sop.inria.fr/marelle/CertiGeo`

These results deserve some comments:

- The number of variables of the Gröbner bases computation is about 20 (the number of coordinates of the points) and the degree of the input polynomials is generally 2 (which is the general case: each ideal can be generated by polynomials of total degree less than two, provided we add extra variables).
- The time for computing a Gröbner base is very sensitive to the variable order. In general, a good choice is to have the variables of base points greater than the variables of constructed points, and to use a reverse-lexicographic term order. But this is not always the case. Only when computing with this naive order was prohibitive, we did try to find an better order. The names of such theorems are marked with a star in Figure 2.
- Certificates with straight-line programs are generally better than raw polynomials. For examples like Ceva's theorem, this makes a significant difference.

Theorems in geometry are not true in general, there are some non-degeneracy conditions: some particular points must not be collinear, some lines not parallel, and so on. From the algebra point of view, this means that we can only prove:

$$CP = \sum_i Q_i P_i$$

where C is a polynomial in some variables, which are parameters of the theorem. From a logical point of view, this means that the conclusion of the theorem

	Time (seconds)		Size (characters)		Size (nodes)	
Theorem	Computing	Verifying	Polynomials	Certificate	Term	SLP
Ceva (264)	181	2.5	538644	477414	266233	76669
Desargues (*)(269)	0.3	0.01	6359	4551	24527	4311
Feuerbach (199)	0.8	0.4	52569	16999	15585	5497
Pappus (*)(100)	1.3	0.2	2721	1934	29945	8031
Pascal_circle (*)	397	12	732982	864509	754290	183505
Pascal_circle2 (20)	91	2.6	10603	15128	312626	66154
Ptolemy (*)	1.1	0.5	1549	1556	26210	9129
Ptolemy_theo95 (142)	200	2.4	571931	571931	344278	73257
Pythagora	0.000	0.009	7	7	4	4
Simson (240)	0.3	0.2	1541	1238	15680	4919
Thales	0.03	0.1	5422	5169	3146	1323
bisectors	0.002	0.04	165	165	105	69
butterfly (119)	0.1	0.2	12116	11125	13661	3980
Euler circle	0.06	0.5	5532	2936	2795	1146
chords	0.002	0.04	639	642	568	282
altitudes	1.1	0.3	4947	5386	5295	1801
isosceles	0.001	0.01	10	10	3	3
medians	0.005	0.06	2910	2717	2284	1064
bisections	0.005	0.06	2577	2145	1911	831
Minh	0.07	0.1	3367	3616	3987	1881
SegmentsofChords	0.1	0.09	10375	9839	7476	2491
threepoints	0.11	0.13	2890	2587	2796	1105
fib(16)	0.003	0.8	15786	393	416	137
fib(17)	0.004	1.3	26059	423	465	151
fib(18)	0.004	2.4	40864	464	517	166
fib(22)	0.008	68	307720	630	753	225

Fig. 2. Times and sizes of some selected theorems

becomes a disjunction of the original conclusion and the degenerate cases. A work-around is to work with coefficients that are rational fractions in some variables $u_1, \ldots u_r$, called *parameters*. Polynomials are then not in $R[X_1, \ldots, X_n]$ anymore but in $R(u_1, \ldots, u_r)[X_1, \ldots, X_n]$. Let us take for example Desargues' theorem. It states that given two triangles (A, B, C) and $(A1, B1, C1)$ and a

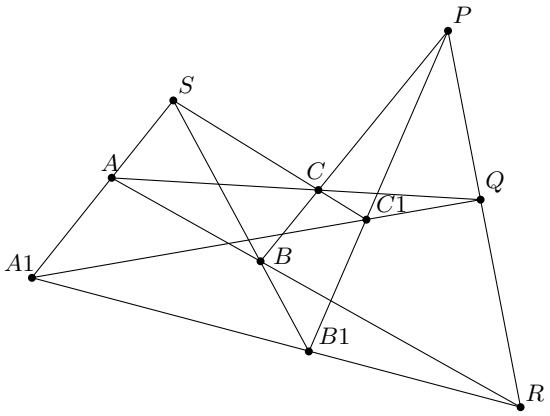

Fig. 3. Desargues'theorem

point S such that S, A, $A1$ are collinear, A, B, $B1$ are collinear, and S, C, $C1$ are collinear, we can deduce that the intersection R of (A, B) and $(A1, B1)$, the intersection Q of (A, C) and $(A1, C1)$, and the intersection P of (B, C) and $(B1, C1)$ are also collinear. All points are in the affine plane. Its statement and proof in Coq are:

```
Lemma Desargues: forall A B C A1 B1 C1 P Q R S:point,
  X S = 0 -> Y S = 0 -> Y A = 0 ->
  collinear A S A1 -> collinear B S B1 -> collinear C S C1 ->
  collinear B1 C1 P -> collinear B C P ->
  collinear A1 C1 Q -> collinear A C Q ->
  collinear A1 B1 R -> collinear A B R -> collinear P Q R
  ∨ X A = X B ∨ X A = X C ∨ X B = X C ∨ X A = 0
  ∨ collinear S B C
  ∨ parallel A C A1 C1 ∨ parallel A B A1 B1.
Proof.
geo_begin.
tzRpv 0%Z (X A::X B::Y B::X C::Y C::X A1::Y B1::Y C1::nil)
 (X B1::X C1::Y P::X P::Y Q::X Q::Y R::X R
 ::Y C1::Y B1::X A1::Y A1::Y C::X C::Y B::X B::nil).
Qed.
```

The theorem is proved by two tactics. The first one, `geo_begin`, transforms the statement in an algebraic one (disjunctions in conclusion become products, negations in hypothesis like $p \neq 0$ becomes $t*p = 1$ where t is a new variable and so on). The second one, `tdzRpv`, takes explicitly as arguments the parameters and the variables with which the certificate generator has to be called and checks back the resulting certificate. Without the extra conditions `X A = X B` $\vee \ldots \vee$ `parallel A B A1 B1`, the theorem is not true. In order to find these conditions, we try to prove the theorem using variables `X A, X B, Y B, X C, Y C, X A1, Y B1, Y C1` as parameters, i.e. allowing to multiply P with a polynomial in these variables. This succeeds and gives a coefficient c which has to be non zero. We take this c and factorise it (for example using MAPLE [4]). In this particular case, we get a product of 7 factors that we translate back as geometric conditions that are added to the goal as disjunctions. Finding these extra-conditions has to be done manually for the moment but with these extra conditions the theorem is proved automatically.

Trying to prove a general statement can give rise to extra conditions that have also to be proved. They correspond to denominators of all the fractions in the certificate being non-zero. When such a condition is not contradictory with the other hypotheses of the statement, it corresponds to an actual degenerated cases, so it is added to the original statement. Nevertheless, we have to be careful. Adding a condition that is contradictory with the other hypotheses would lead to a theorem that is trivially true but useless. For some examples, this detection of contradictions can be extremely costly.

Adding extra conditions has to be done for almost all theorems. This is not an easy task because selecting automatically which variables have to be put as parameters is not direct. As a matter of fact, there is a general method that we outline now. The set of coefficients c such that cP is in the ideal generated by $P_1, \ldots, P_s$ is itself an ideal. It defines an algebraic variety which represents the cases were the theorem is false. In order to completely describe this variety, one should compute its irreducible components. This can be done again with Gröbner bases computations. With the algebraic description of the variety of non-degenerate conditions, we will then be able to state the theorem correctly, even if some conditions are not easily expressible with the usual geometric predicates. For the theorems we have addressed so far, the heuristic method succeeds reasonably quickly, so we did not have to use the general machinery of irreducible decomposition of algebraic varieties yet.

We have compared our method with two systems: MACAULAY2 [10], a system dedicated to algebraic geometry which is very efficient in Gröbner bases computations and HOL LIGHT [14], a proof assistant that has a tactic for geometry theorem proving using Gröbner bases computation [15] [18] . For MACAULAY2, in several cases, e.g. Pascal's theorem for the circle, it was not able to check ideal membership (time over $1000s$) because it fails to compute the whole Gröbner basis while our method succeeds (in 397 seconds). For HOL LIGHT, Figure 4 gives a more extensive comparison. All theorems are general instances of the ones presented in Figure 2. Since our version of HOL LIGHT was running in in-

terpreted mode, times have been divided by a factor of 4 to compensate the fact that Coq is using native code (4 is the average ratio between interpreted versus compiled code in Ocaml). In HOL Light, geometrical theorems are proven by refutation, i.e. the conclusion of the theorem is negated and the contradiction $1 = 0$ is proved by showing that 1 is in the Gröbner basis. Since this method differs from ours (the benefit of dividing the conclusion by the partial Gröbner basis during completion is lost), each theorem with hypothesis H, generic case C and particular cases $CP_1, \ldots, CP_n$ is proved using two equivalent formulations:

(1) the particular cases are negated in hypotheses: $H \wedge \neg CP_1 \wedge \ldots \wedge \neg CP_n \Rightarrow C$
(2) the conclusion is negated: $H \wedge \neg CP_1 \wedge \ldots \wedge \neg CP_n \wedge \neg C \Rightarrow 1 = 0$.

The idea is that the first version benefits from dividing the conclusion while the second one exactly mimics HOL Light behaviour.

	Times (seconds)			
	Coq (1)	HOL Light (1)	Coq (2)	HOL Light (2)
Feuerbach	94	>2000	>2000	>2000
Ptolemy	3.7	>800	20	>800
Ceva	4.8	28	5	28
Minh	0.8	1.9	27	1.9
Butterfly	24	20	25	21
Pappus	0.9	1	1	1
Euler circle	20	0.2	1	0.3
Pascal circle	50	6	11	6
Simson	89	118	7.3	121
Desargues	117	28	32	29
Threepoints	3	75	2.4	75

	Times (seconds)	
	Coq	HOL Light
Feuerbach or	1127	>2000
Ceva or	26	27
Pappus or	51	0.2
Desargues or	>500	11
Pascal circle or	44	8

Fig. 4. Comparison with HOL Light

Let us now comment on these results:

- the first block of lines of the first table contains theorems for which refutation is slower than our method. For these theorems, our tactic is faster than HOL Light.
- the second block contains theorems for which refutation is faster. For these theorems, our tactic and HOL Light are similar.

– the second table contains theorems written as $H \Rightarrow C \vee CP_1 \vee \ldots \vee CP_n$. In this case, our method behaves very badly: the polynomial representing the conclusion is rather big, so dividing it to 0 takes a long time.

There is no clear winner between our method and the refutation one. However it seems that when the theorem deals with euclidean geometry and not only projective one, our method is better. In average, our tactic is faster than the one in HOL Light and it has the extra benefit of generating a certificate that can be easily verified.

6 Conclusion

This paper addresses an issue that is rarely taken into account by the automated theorem proving community: can we really trust the tools we are using to prove theorems? For proof assistants, this question is central. Proof assistants are systems where mathematical knowledge is added progressively: new facts are derived from previously proved ones. Current systems usually come with libraries that contain thousands of theorems. It is then crucial for these libraries to be built with the highest degree of confidence.

Nullstellensatz theorem and Gröbner bases algorithms are well-known ingredients to automatically prove theorems but how can they be put into action if, for example, one would like to build a library of geometrical facts that can be easily certified even by other proof assistants? The main contribution of this paper is to propose an effective way to do this. It is very easy to device a solution that works only on small examples. Before what is proposed in this paper, we had several non-conclusive attempts:

1. First, we have developed a certified implementation of Buchberger's algorithm that can be run within Coq [27]. This implementation was still an order of magnitude slower than usual implementations that can be found in computer algebra systems. Furthermore, any modification of this implementation usually requires a non-trivial proving effort to re-establish its correctness. Computations like the one needed for Ceva's theorem could not be performed inside Coq.
2. We have also tried to use efficient programs that computes Gröbner bases [8] using standard techniques of effective algebra [25]. These techniques make it possible to use the program as a black box but requires to add extra variables to the problem in order to compute the Q_i. The complexity of Gröbner bases being very sensitive to the number of variables, examples like Pascal were clearly out of reach.
3. We have also experimented with the simple form of certificates that only contains the coefficients of the linear combination as in the example (1) of page 6. Unfortunately, for some examples like Feuerbach, checking the certificates was taking much more time than generating them.

The notion of certificate as straight-line programs is a key aspect of this work. We agree that the only insight we could get is the fib example that shows that

straightline program captures some cancellation. Still, we could not characterise when this actually happens. But it gives an explicit interface between the computation that is done externally of the proof assistant and the proof checking that is done inside the proof assistant. It also provides a very compact way of writing certificates. This means for example that transferring all the theorems of Table 2 into another proof assistant is straightforward: one just needs to develop his own trusted version of the checker. Note that in that case the time for generating the certificate becomes irrelevant: having Ceva's theorem would only require the time to check the certificate, i.e. a couple of seconds.

The system we have developed has made it possible to get very quickly a library of standard theorems of geometry within Coq. For building the library, we intensively use our secure decision procedure for ideal membership. Each theorem was first stated in its full generality. Then, our tactic that turns the problem in a membership problem and calls the decision procedure was tried. Most of the time, Coq failed to fully accept the certificate returned by the decision procedure. Some polynomials have to be proved non-zero. So we add them as extra conditions. As explained before, this was done manually. Once the extra conditions added, the tactic was tried again and this time the certificate was accepted. The theorem with its extra-conditions could then be stored in the Coq database. What was important in this experiment was to show that our decision procedure that is much slower than state-of-the-art Gröbner implementations could still be used for proving interesting theorems. We have encountered very few examples (Pascal's theorem for conics is one of them) where our decision procedure fails for a clear lack of computing power. This method of proving is of course very sensitive to the way theorems are stated. This is well known. Only one theorem has resisted our attempts to turn it into a Coq theorem, it is Morley's theorem (this theorem is in [5] but its statement involved extra points).

Looking at the problem of testing ideal membership from the perspective of generating small certificates is also very intriguing. We plan to further work on our generator. We believe it is possible to greatly improve both the efficiency and the compactness of the certificates that are generated. We also plan to apply similar ideas of compact certificates to other techniques that are used in geometry theorem proving. In that respect, Ritt-Wu's decomposition algorithm [6] seems a natural candidate.

7 Acknowledgements

This work was supported by the ANR Galapagos. Yves Bertot helped us with his expertise of the GeoGebra system [19]. We thanks the referees for their constructive comments on the first version of this paper.

References

1. Stuart F. Allen, Robert L. Constable, Douglas J. Howe, and William E. Aitken. The semantics of reflected proof. In *LICS*, pages 95–105. IEEE Computer Society, 1990.

2. Bruno Buchberger. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. volume 41 of *Journal of Symbolic Computation*, 2006.
3. Amine Chaieb and Makarius Wenzel. Context Aware Calculation and Deduction. In *Calculemus/MKM*, volume 4573 of *LNCS*, pages 27–39. Springer-Verlag, 2007.
4. Bruce W. Char, Gregory J. Fee, Keith O. Geddes, Gaston H. Gonnet, and Michael B. Monagan. A tutorial introduction to MAPLE. 2(2):179–200, June 1986.
5. Shang-Ching Chou. *Mechanical geometry theorem proving*. Kluwer Academic Publishers, Norwell, MA, USA, 1987.
6. Shang-Ching Chou and Xiao-Shan Gao. Ritt-Wu's Decomposition Algorithm and Geometry Theorem Proving. In *CADE*, volume 449 of *LNCS*, pages 207–220. Springer-Verlag, 1990.
7. David Eisenbud. Commutative algebra: with a view toward algebraic geometry. Graduate Texts in Mathematics. Springer-Verlag, 1999.
8. Jean-Charles Faugère. A new efficient algorithm for computing Gröbner bases (f4). volume 139 of *Journal of Pure and Applied Algebra*, pages 61–88, 1999.
9. Marc Giusti, Joos Heintz, Jose Enrique Morais, Jacques Morgenstern, and Luis Miguel Pardo. Straight-line programs in geometric elimination theory. *Journal of Pure and Applied Algebra*, 124(1/3):101 – 146, 1998.
10. Daniel R. Grayson and Michael E. Stillman. Macaulay2. page http://www.math.uiuc.edu/Macaulay2/, 2009.
11. Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In *International Conference on Functional Programming 2002*, pages 235–246. ACM Press, 2002.
12. Benjamin Grégoire and Assia Mahboubi. Proving equalities in a commutative ring done right in coq. In *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005, Proceedings*, volume 3603 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2005.
13. Benjamin Grégoire, Laurent Théry, and Benjamin Werner. A computational approach to Pocklington certificates in type theory. In *FLOPS'06*, volume 3945 of *LNCS*, pages 97–113. Springer-Verlag, 2006.
14. John Harrison. HOL Light: A tutorial introduction. In *FMCAD'96*, volume 1166 of *LNCS*, pages 265–269. Springer-Verlag, 1996.
15. John Harrison. Complex quantifier elimination in HOL. In *TPHOLs 2001: Supplemental Proceedings*, pages 159–174. Division of Informatics, University of Edinburgh, 2001. Published as Informatics Report Series EDI-INF-RR-0046.
16. John Harrison. Automating elementary number-theoretic proofs using Gröbner bases. In *CADE 21*, volume 4603 of *LNCS*, pages 51–66. Springer-Verlag, 2007.
17. John Harrison. Verifying nonlinear real formulas via sums of squares. In *TPHOLs'2007*, volume 4732 of *LNCS*, pages 102–118. Springer-Verlag, 2007.
18. John Harrison. Practical logic and automated reasoning. page 414. Cambridge University Press, 2009.
19. Markus Hohenwarter and Judith Preiner. Dynamic Mathematics with GeoGebra. *Journal of Online Mathematics*, 7:ID 1448, March 2007.
20. Deepak Kapur. Geometry theorem proving using hilbert's nullstellensatz. In *SYMSAC '86: Proceedings of the fifth ACM symposium on Symbolic and algebraic computation*, pages 202–208. ACM, 1986.
21. Deepak Kapur. A refutational approach to geometry theorem proving. *Artificial Intelligence*, 37(1-3):61–93, 1988.

22. Deepak Kapur. Automated Geometric Reasoning: Dixon Resultants, Gröbner Bases, and Characteristic Sets. In *Automated Deduction in Geometry*, volume 1360 of *LNCS*, pages 1–36. Springer-Verlag, 1996.
23. Georg Kreisel and Jean Louis Krivine. *Elements of Mathematical Logic (Model Theory)*. Studies in Logic and the Foundations of Mathematics. North-Holland, 1967.
24. Lawrence C. Paulson. Isabelle: A generic theorem prover. *Journal of Automated Reasoning*, 828, 1994.
25. Loïc Pottier. Connecting Gröbner Bases Programs with Coq to do Proofs in Algebra, Geometry and Arithmetics. In *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and The 7th International Workshop on the Implementation of Logics*, number 418. CEUR Workshop Proceedings, 2008.
26. Judit Robu. Geometry Theorem Proving in the Frame of the Theorema Project. Technical Report 02-23, RISC Report Series, University of Linz, Austria, September 2002. PhD Thesis.
27. Laurent Théry. A Machine-Checked Implementation of Buchberger's Algorithm. *Journal of Automated Reasoning*, 26(2), 2001.
28. Freek Wiedijk. *Formalizing 100 Theorems*. `http://www.cs.ru.nl/~freek/100`.
29. Wen-Tsun Wu. On the decision problem and the mechanization of theorem-proving in elementary geometry. In *Automated Theorem Proving - After 25 Years*, pages 213–234. American Mathematical Society, 1984.