Quantifier Elimination over Algebraically Closed Fields in a Proof Assistant using a Computer Algebra System

David Delahaye [1]

CPR (CEDRIC)
CNAM
Paris, France

Micaela Mayero [2]

LCR (LIPN)
Université Paris Nord (Paris 13)
Villetaneuse, France

Abstract

We propose a decision procedure for algebraically closed fields based on a quantifier elimination method. The procedure is intended to build proofs for systems of polynomial equations and inequations. We describe how this procedure can be carried out in a proof assistant using a Computer Algebra system in a purely skeptical way. We present an implementation in the particular framework of Coq and Maple giving some details regarding the interface between the two tools. This allows us to show that a Computer Algebra system can be used not only to bring additional computational power to a proof assistant but also to enhance the automation of such tools.

Key words: Theorem Proving, Computer Algebra, Algebraically Closed Fields, Coq, Maple

1 Introduction

An Algebraically Closed Field (ACF) K is a field which has no proper algebraic extension, i.e. every algebraic extension is K itself. This means that every

[1] David.Delahaye@cnam.fr, http://cedric.cnam.fr/~delahaye/.
[2] mayero@lipn.univ-paris13.fr, http://www-lipn.univ-paris13.fr/~mayero/.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

non-constant polynomial of $K[X]$ has a root in K. With respect to the usual properties of a field, this adds the following condition:

$$\forall P \in K[X].\mathsf{deg}(P) > 0 \Rightarrow \exists x \in K.P(x) = 0 \tag{1}$$

where $\mathsf{deg}(P)$ denotes the degree of P.

As examples of ACFs, we can take the field of complex numbers, which is the algebraic closure (i.e. the algebraic extension which is algebraically closed) of the field of real numbers, or the field of algebraic numbers, which is the algebraic closure of the field of rationals. It can be shown that equation (1) is equivalent to what is known as the Fundamental Theorem of Algebra (FTA), also called D'Alembert's theorem[3] when proved over the field of complex numbers, which states that, given an ACF K, every polynomial of $K[X]$ of degree $n > 0$ has exactly n roots (which may not be distinct). From a mathematical point of view, this theorem has the nice and direct consequence that polynomials over $K[X]$ can be factorized and it is possible to solve polynomial equation and inequation systems of the following form:

$$\begin{cases} P_1(X) = 0, \ldots, P_n(X) = 0 \\ Q_1(X) \neq 0, \ldots, Q_m(X) \neq 0 \end{cases} \tag{2}$$

To solve this kind of system, we only have to factorize all the polynomials and to choose a common root of all P_i which is not a root of any Q_j. However, from a computational point of view, this process of factorization is not fully automatic in general. It can only be done in some particular cases such as over finite fields or in the field of complex numbers using, for instance, the Kneser constructive proof of equation (1) (see the FTA project [9]). Moreover, in practice, these methods turn out to be unsatisfactory: the former raises a problem of complexity (in general, the size of the splitting field of a polynomial is exponential in the degree), whereas the latter can produce arbitrary algebraic numbers as roots (typically, in [9], roots are pairs of limits of Cauchy sequences) and these are generally difficult to deal with.

In this paper, in order to solve systems such as (2), we consider an alternative method, called quantifier elimination due to the implicit existential quantifier (over the main variable X of the polynomials) we are trying to eliminate, which is mainly based on the idea of getting rid of the polynomial parts which do not contain the solution. This can be carried out by means of the notion of polynomial gcd, which is the main concept underlying this method and which makes it possible to simplify the system to be solved. Moreover, this method is a little stronger in the sense that it can deal with ACFs of any characteristic (not only 0), even if the latter must be known.

To integrate this method into a Deduction System (DS), several options can be considered (as described in [1]; some of these ideas are also discussed in [17] in a more general way to develop reliable algorithmic software). The

[3] Due to his first serious attempt at a proof of the FTA in 1746, even if the first proof is usually credited to Gauss in his doctoral thesis of 1799.

first could be to carry it out in a purely autarkic [4] way, i.e. the method is entirely coded in the DS and, in particular here, we have to define the notion of gcd (with possibly its proof of correctness). This choice may appear quite natural but it may also seem a little redundant with respect to the existence of Computer Algebra Systems (CAS) which are dedicated to this kind of computation and are probably more efficient. Another option is to benefit from the help provided by Computer Algebra (CA) using a CAS each time a symbolic computation is needed and verifying its correctness in a so-called skeptical style. Actually, we could also consider a believing approach which consists in trusting the CAS and assuming that the returned computations are correct, but such assumptions are a little too strong (CAS may have some bugs but more frequently, some side conditions may be omitted during computations) and we may find this option unsuitable.

In this paper, we focus on a skeptical approach to carry out the quantifier elimination method over ACFs (considering univariate polynomials) and, in particular, we describe how much easier it is to verify that a polynomial is a gcd (thanks to the cofactors and Bézout's theorem) than to compute it directly (with a certified function). To implement the method, we chose Coq [22] as a DS and Maple [4] as a CAS since an existing interface between these two tools [8] (designed by the authors) was an appropriate basis for an extension for the polynomial gcd. The method we consider is commonly accepted (it combines some well-known properties) and this paper provides two main novelties: the first consists in presenting the proof of the quantifier elimination problem in a purely constructive way so that an algorithm can be extracted; the second is to show that CASs can be used not only to bring computation to DSs (as in [12] or in [8]) but also to enhance the automation of such tools.

2 Quantifier elimination over ACFs

Before explaining the method we used, we need some preliminary lemmas mainly related to the polynomial gcd (for information, the corresponding proofs, which are rather well known, can be found in appendix A). In the following, the functions deg and gcd will denote respectively the degree of a polynomial and the monic gcd of two polynomials. Considering a polynomial $P(X)$, the notation P will denote, by default, the polynomial $P(X)$, except under a quantifier binding the variable say x, where it will denote the value of the corresponding function of $P(X)$ at x. We consider that we are implicitly in an ACF K and every polynomial will be in $K'[X]$, where $K' \subseteq K$ s.t. the equality to zero is decidable and, given two polynomials P_1 and P_2, $\mathsf{gcd}(P_1, P_2) \in K'$.

[4] The words "autarkic", "skeptical" and "believing" were coined by Henk Barendregt and Arjeh M. Cohen in [1].

2.1 Preliminary lemmas

Proposition 2.1 (Common roots) *Let P_1 and P_2 be two non-zero polynomials. Let $G = \mathsf{gcd}(P_1, P_2)$. We have: $\exists x.P_1 = 0 \wedge P_2 = 0$ iff $\exists x.G = 0$.*

Proposition 2.2 (Existence of a non-root value) *Let Q be a non-zero polynomial. We have: $\exists x.Q \neq 0$.*

Proposition 2.3 (Roots of relatively prime polynomials) *Let P and Q be two relatively prime non-zero polynomials (i.e. $\mathsf{gcd}(P, Q) = 1$). We have: $\exists x.P = 0 \wedge Q \neq 0$ iff $\exists x.P = 0$.*

Proposition 2.4 (Roots of quotient and gcd) *Let P and Q be two non-zero polynomials. Let $G = \mathsf{gcd}(P, Q)$ and P_1 be a polynomial s.t. $P = GP_1$. We have: $\exists x.P = 0 \wedge Q \neq 0$ iff $\exists x.P_1 = 0 \wedge G \neq 0$.*

The following proposition states that iterating the application of proposition 2.4 defines a well-founded induction scheme (the induction is made over the degree) and this will allow us to ensure the termination of our algorithm:

Proposition 2.5 (Degree decrease) *Let P and Q be two non-zero polynomials. Let $G = \mathsf{gcd}(P, Q)$ and P_1 be a polynomial s.t. $P = GP_1$. If P and Q are not relatively prime then $\mathsf{deg}(P_1) < \mathsf{deg}(P)$.*

2.2 Method

In the problem consisting in solving system (2), we will not consider constant polynomials, which can be easily dealt with before applying the method described below. Indeed, given $P_i = c$, where $c \in K'$, if $c = 0$ then this equality can be removed from the system, or else the system has no solution. Similarly, considering $Q_i = c$, with $c \in K'$, if $c = 0$ then the system has no solution, otherwise the equality can be removed.

The main theorem of quantifier elimination over ACFs is the following:

Theorem 2.6 (Quantifier elimination) *Let $P_{i,\ i=1..n}$ and $Q_{j,\ j=1..m}$ be non-constant polynomials. The problem to know if there exists x s.t.:*

$$P_1 = 0 \wedge ... \wedge P_n = 0 \wedge Q_1 \neq 0 \wedge ... \wedge Q_m \neq 0$$

is decidable.

Proof. If Φ denotes the proposition $\exists x. \bigwedge_{i=1}^{n} P_i = 0 \wedge \bigwedge_{i=1}^{m} Q_i \neq 0$, the idea is to prove $\Phi \vee \neg\Phi$ constructively (i.e. without using the excluded middle trivially) and using only well-founded induction schemes. In particular, this means that the previous proposition must be proved either by giving a proof of Φ or by giving a proof of $\neg\Phi$. Moreover, if some properties are used iteratively (providing equivalent propositions to be proved), we must ensure that the process terminates (looking for some decreasing measures). Thus, this will prove the theorem but it will also allow us to extract a correct, complete and

terminating procedure for our problem[5]. In the following, the *left* and the *right* cases denote that we prove $\Phi \vee \neg\Phi$ giving respectively a proof of Φ and a proof $\neg\Phi$.

Let $P = \mathsf{gcd}(P_{i,i=1..n})$ and $Q = \prod_{i=1}^{m} Q_i$ when respectively $n > 0$ and $m > 0$.

(i) $n = 0$, $m > 0$: we are in the *left* case (proposition 2.2).

(ii) $n > 0$, $m = 0$: the root is common to P_i i.e. it is equivalent (proposition 2.1) to showing if $\exists x.P = 0$ or not. If $P \neq 1$ then we are in the *left* case (definition of ACF); if not we are in the *right* case.

(iii) $n > 0$, $m > 0$:
- if $P = 1$ then we are in the *right* case.
- else let $G = \mathsf{gcd}(P, Q)$:
  - if $G = 1$ then it is equivalent to showing if $\exists x.P = 0$ or not (proposition 2.3).
  - else let P' be a polynomial s.t. $P = GP'$. It is equivalent to showing if $\exists x.P' = 0 \wedge G \neq 0$ or not (proposition 2.4; well-founded induction scheme due to proposition 2.5).

□

As said previously, we can extract, from the constructive proof of theorem 2.6, an algorithm which builds (if possible) a proof of:

$$\exists x.P_1 = 0 \wedge ... \wedge P_n = 0 \wedge Q_1 \neq 0 \wedge ... \wedge Q_m \neq 0$$

The algorithm is the following:

I. if $n = 0$ then go to III else compute $P = \mathsf{gcd}(P_{i,i=1..n})$.

II. if $m = 0$: apply proposition 2.1. It is equivalent to proving $\exists x.P = 0$:
   1. if $P \neq 1$ then terminate applying the definition of ACF
   2. else fail

III. if $m \neq 0$:
   1. compute $Q = \prod_{i=1}^{m} Q_i$
   2. if $n = 0$ then apply proposition 2.2
   3. else it is equivalent to proving $\exists x.P = 0 \wedge Q \neq 0$:
      a. if $P = 1$ then fail
      b. else compute $G = \mathsf{gcd}(P, Q)$:
         i. if $G = 1$ then apply proposition 2.3
         ii. else apply proposition 2.4 and re-apply the algorithm.

Actually, in practice, it is more efficient to apply step III to each Q_i individually rather than to the product of all Q_i, but the latter is only used to simplify the presentation of the algorithm.

[5] We do not recall here the ideas of Brouwer-Heyting-Kolmogorov's semantics which make this possible (giving a functional behavior to proofs) and the reader who may not be familiar with these notions is invited to refer to the abundant literature in this domain.

Remark 2.7 The method that has been described is fully constructive in the general theory of ACFs even if the method stops at the definition of ACF, which saves us from giving an explicit solution. Indeed, this definition is an axiom and cannot be realized (i.e. proved constructively) in the theory of ACFs in general. However, if the method remains constructive in general, it can be more informative in some particular cases, as for $\mathbb{C}$ for example, where the constructive proof of FTA can be used to build effective roots (even if these roots may be arbitrary algebraic numbers, which are often difficult to handle).

3 Integration in a DS using a CAS

Let us see how the procedure described in the previous section can be integrated into a DS and how a CAS can help for some computations (mainly for the gcd). We also present a specific implementation in the framework of Coq where Maple is called to perform the symbolic computations.

3.1 Polynomials and characteristic

To implement the method described in subsection 2.2, we must ensure that everything can be decided and more precisely, that a coefficient is zero or not. For example, this is necessary to compute the degree of a polynomial or the gcd of two polynomials (indeed, the degree of a polynomial is defined as the greatest natural number s.t. the corresponding coefficient is non-zero, whereas gcd algorithms, e.g. Euclid's algorithm, generally terminate with the zero polynomial). In our method, this is ensured by means of the assumption that we use polynomials over $K'[X]$ where $K' \subseteq K$ s.t. the equality to zero can be decided and the gcd of polynomials over $K'[X]$ remains in $K'[X]$. The former condition over K' deals with the case of the zero coefficient whereas the latter makes the recursive call of the algorithm (described in subsection 2.2) possible. Thus, in our implementation, we decided to use rational polynomials, i.e. polynomials with rational coefficients. However, as the characteristic may be non-zero, the latter must be given (it appears simply as a parameter in our theory) and then the equality to zero can be decided (the corresponding function is generically built w.r.t. the characteristic).

3.2 A skeptical approach

The idea of our approach is purely skeptical (see [1] for a general study of how to make DSs and CASs interact). This means that the quantifier elimination procedure is handled by the DS which may call a CAS for some computations and must verify the correctness of these computations. These verifications are intended to ensure that the logical consistency of the DS is not endangered by these external computations. Actually, this may look obvious but in this approach, it is not always necessary to prove the correctness of an

external computation if no property is required over this computation. For example, considering the algebraic expression $x^2 - 1$, if we want to factorize it in $(x+1)(x-1)$ using a CAS, there is no need, *a priori*, to verify that the result returned by the CAS is correct. In particular, we can use it without any verification if we do not make the assumption that this result is the factorization of $x^2 - 1$ (otherwise, we could introduce a potential inconsistency). However, if we want to ensure that the two expressions are equivalent (to replace the latter by the former in a proof, for example) then this equivalence has to be proved (typically using the appropriate ring properties).

In our procedure, the main symbolic computation we use is the polynomial gcd and this is exactly what we want to be performed by a CAS. We must also point out that this computation must be verified as being exactly the gcd since several lemmas of section 2 used in the procedure require this result to be a gcd and not only a divider. To do so, the idea is to use the converse of Bézout's theorem (which is stated in appendix A to prove some of the preliminary lemmas of section 2), i.e.:

Theorem 3.1 (Bézout's theorem, converse) *Let P, Q and G be three non-zero polynomials. If G divides P and Q, and if there exist two polynomials A and B s.t. $AP + BQ = G$ then G is the gcd of P and Q.*

With this theorem, in addition to the gcd, we need to ask to the CAS to also return the two quotients P_1 and Q_1 corresponding respectively to the division of P and Q by G, as well as the two cofactors A and B of the Bézout relation. Thus, in the DS, it remains to prove the following equalities:

$$\begin{cases} P = P_1 G \\ Q = Q_1 G \\ G = AP + BQ \end{cases}$$

This additional information provided by the CAS can be seen as certificates which are intended to help the DS to prove the correctness of the computation. However, some of them are little more than simple certificates, for example, the quotient P_1 is also used in the procedure itself (see propositions 2.4 and 2.5 in section 2).

We can notice that the previous equalities are much easier to prove than to compute the gcd in an autarkic way (i.e. directly in the DS) because we have to write the gcd function in the DS, but more difficult, we also have to build its correctness proof. In our approach, we can potentially benefit from the use of more clever algorithms (than Euclid's algorithm, in some particular cases where the coefficients are integers or rationals, for example) provided by the CAS whose correctness does not need to be proved (we verify that each call is correct but we do not ensure the global correctness of the external function which would be as hard to prove as the algorithm is sophisticated).

3.3 *Using* Coq *and* Maple

As a concrete application, we chose to implement our decision procedure using Coq [22] as a DS and Maple [4] as a CAS. This choice was motivated by the existence of an interface between the two tools [8] (designed by the authors and available as a Coq user contribution), which was a good basis for any extension required by the realization of our method. This interface is intended to bring, to Coq, computations from Maple on algebraic expressions over a field in a skeptical way. The possible properties (mainly equalities required to be verified by the skeptical mode) are automatically handled by a tactic, called `field` [7] (also designed by the authors), which may generate some side conditions (over the inverses) to be proved by the user. To carry out our method, it has been necessary to develop an univariate polynomial theory in Coq and the Coq/Maple interface has been extended to deal with the polynomial gcd. This operation obviously returns the gcd but also the quotients and the cofactors (required, as said previously, to prove that it is the gcd). The validation of the gcd implies proving the equalities given in section 3.2, which is automatically done using the polynomial operations and comparing the polynomial coefficients (we use the tactic `field`). In this way, the call to Maple is quite transparent for the Coq user.

Another point which motivated our choice, was that Coq owns an elaborated toplevel tactic language (see [6]). With this high-level language, it is not necessary to go into the details of the implementation of Coq (which is necessary if you intend to write your tactic with the full-programmable meta-language[6]) and our method has been built very intuitively (w.r.t. the procedure given in section 2.2). This possibility is obviously important for the developer but also for any user who does not get lost when taking a look at the code. The whole development (containing the updated Coq/Maple interface, the quantifier elimination tactic as well as some examples) will be soon available as a new Coq contribution. For information, this development works with Coq 8.0 (the latest version of Coq) and Maple V (and should also work with newer versions of Maple).

4 Some examples

To show some examples, we use the tool that has been implemented in the framework of Coq/Maple and briefly described in subsection 3.3. As an ACF, we consider a construction of $\mathbb{C}$ (where the elements are pairs of reals), which we call `C` (the property of ACF has been assumed but could be proved later). The constants `C0` and `C1` denote respectively the neutral elements 0 and 1 of `C`. Rational constants of `C` are built by means of the function `Ccte` which, given

[6] The meta-language must be distinguished from the logic language (where the theorems are expressed and proved) and aims at providing a means to write programs which can build (automatically) proofs.

a natural number n, returns the expression `(C1+...+C1)/C1`, n times, where `+` and `/` are respectively the addition and the division in `C`. The univariate polynomial theory (with rational coefficients), we developed and called `pol`, requires a canonical representation of polynomials, which are encoded as lists of coefficient/power sorted decreasingly w.r.t. the power. Thus, a polynomial is formed using the constructor `PList` which takes, as arguments, the coefficient/power list and a proof that the list is correctly sorted. Given a value, the function `UPeval` allows us to evaluate a polynomial.

In our examples, we consider the following polynomials (to simplify, we have chosen to give only integer coefficients, rather than rational coefficients; however, in these examples, we often deal with rational polynomials since the quotients of the division by the gcd as well as the cofactors of the Bézout relation, necessary to ensure the correctness of the computation as described in subsection 3.2, are often rational polynomials):

$$P_1 = 3X^2 + X + 2$$
$$P_2 = 3X^3 + 10X^2 + 5X + 6$$
$$Q_1 = 2X - 1$$
$$Q_2 = 2X^2 + 5X - 3$$

which are supposed to be defined as the Coq constants `P1`, `P2`, `Q1` and `Q2`. More precisely and for information, `P1` is, for example, defined as follows:

```
Definition P1 : pol C := PList C ((Ccte 3, 2) :: (Ccte 1, 1) ::
                                  (Ccte 2, 0) :: nil) sorted_p1.
```

where `::`/`nil` are the list constructors and `sorted_p1` is a proof (previously built) that the list of powers (i.e. `2`, `1` and `0`) is decreasingly sorted.

In the following, we suppose that the ACF axiomatization, the univariate polynomial theory, the Coq/Maple interface and the quantifier elimination tactic, called `qelim`, have been loaded in the Coq toplevel. We also suppose that the previous polynomials `P1`, `P2`, `Q1` and `Q2` have already been defined.

4.1 Some simple cases

Here, we give some examples illustrating some identified cases of the algorithm described in subsection 2.2.

4.1.1 Case II: $n \neq 0$ and $m = 0$

We propose to prove the following proposition:

$$\exists x. P_1 = 0 \wedge P_2 = 0$$

In this case, we have $\mathtt{gcd}(P_1, P_2) = 3X^2 + X + 2$ and by proposition 2.1, it is equivalent to showing $\exists x. 3x^2 + x + 2 = 0$ (which is trivially proved by the

definition of ACF).

In Coq, we have:

```
Coq < Goal exists x : C, UPeval x P1 = C0 /\ UPeval x P2 = C0.
1 subgoal

  ============================
   exists x : C, UPeval x P1 = C0 /\ UPeval x P2 = C0
```

Once the definitions of P1 and P2 unfolded, we apply the quantifier elimination tactic (qelim):

```
Unnamed_thm < unfold P1, P2.
1 subgoal

  ============================
   exists x : C,
     UPeval x (PList C ((Ccte 3, 2) :: (Ccte 1, 1) ::
                        (Ccte 2, 0) :: nil) sorted_p1) = C0 /\
     UPeval x (PList C ((Ccte 3, 3) :: (Ccte 10, 2) :: (Ccte 5, 1) ::
                        (Ccte 6, 0) :: nil) sorted_p2) = C0

Unnamed_thm < qelim.
Proof completed.
```

In the same way, we could also test the dual situation where $n = 0$ and $m \neq 0$ (case 2) using, for example, Q_1 and Q_2 to prove $\exists x.Q_1 \neq 0 \wedge Q_2 \neq 0$.

4.1.2 Case b: recursive call

In the following example, we want to get a recursive call of the algorithm (case ii of b). This can be obtained when trying to prove:

$$\exists x.P_2 = 0 \wedge Q_2 \neq 0$$

In this case, we have $\mathsf{gcd}(P_2, Q_2) = X + 3 = G$ and by proposition 2.4, it is equivalent to proving $\exists x.P_{2q} = 0 \wedge G \neq 0$, where P_{2q} is s.t. $P_2 = P_{2q}G$. We have to re-apply the algorithm: we compute $\mathsf{gcd}(P_{2q}, G) = 1$ and by proposition 2.3, it is equivalent to showing $\exists x.P_{2q} = 0$ (which is trivially proved by the definition of ACF).

In Coq, as previously, after having unfolded the definitions of P2 and Q2, we can apply the tactic qelim:

```
Coq < Goal exists x : C, UPeval x P2 = C0 /\ UPeval x Q2 <> C0.
Unnamed_thm < unfold P2, Q2; qelim.
Proof completed.
```

4.1.3 Case a: failure

Finally, to illustrate the completeness of our tactic, we can give an example which fails (in the case a). This situation can occur when trying to prove the following proposition:

$$\exists x. P_1 = 0 \wedge Q_1 = 0 \wedge Q_2 \neq 0$$

Here, P_1 and Q_1 are relatively prime and it is not possible to find common roots (actually, the condition over Q_2 is irrelevant since we cannot solve the constraint between P_1 and Q_1).

In Coq, we have:

```
Coq < Goal exists x : C, UPeval x P1 = C0 /\ UPeval x Q1 = C0 /\
Coq <                    UPeval x Q2 <> C0.
Unnamed_thm < unfold P1, Q1, Q2; qelim.
Toplevel input, characters 739-744
> unfold P1, Q1, Q2; qelim.
>                    ^^^^^
Error: Tactic failure "This system has no solution!"
```

where the error is raised by the tactic `qelim`, which fails to be applied (since the statement to be proved is false).

4.2 Another example

A more sophisticated example could be inspired by a geometrical problem. For instance, let us consider two curves and a line defined by the following polynomials:

$$quartic = X^4 + X^3 + X^2 + X$$
$$cubic = X^3 + X^2 + X + 1$$
$$line = X + 1$$

The problem is to know if there exist points which are on the two curves but not on the the line, i.e. can we prove the following proposition:

$$\exists x. x^4 + x^3 + x^2 + x = 0 \wedge x^3 + x^2 + x + 1 = 0 \wedge x + 1 \neq 0$$

Informally, we know that there are 3 points which are solutions of $x^4 + x^3 + x^2 + x = 0$ and $x^3 + x^2 + x + 1 = 0$: i, $-i$ and -1. The solution -1 does not satisfy $x + 1 \neq 0$ but the two other complex solutions do. Thus, this problem has a solution.

In Coq, if we suppose that the corresponding polynomials `quartic`, `cubic` and `line` have already been defined, we have:

```
Coq < Goal exists x : C, UPeval x quartic = C0 /\
Coq <                    UPeval x cubic = C0 /\ UPeval x line <> C0.
```

```
Unnamed_thm < unfold quartic, cubic, line; qelim.
Proof completed.
```

5 Related work

Quantifier elimination is a very prolific topic in Computer Algebra. Many studies have been carried out, mainly over Real Closed Fields (RCF) but also over ACFs. Regarding ACFs and w.r.t. our approach, these studies are generally in the particular context of complex numbers (assuming the null-characteristic) or do not aim at being implemented in a DS (and obviously do not consider a CAS to do so). For example, we can refer to the work of John Harrison [11] who developed a decision procedure for the first order algebraic theory of $\mathbb{C}$ in the HOL proof assistant in order to prove the FTA. Hervé Perdry, in his PhD thesis [18], gives a simple algorithm inspired from a kind of cylindric algebraic decomposition. Finally, Doug Ierardi [14] gives a fast and parallel decision procedure for ACFs.

Regarding RCFs, historically, the first quantifier elimination algorithm was given by Alfred Tarki [21] (see also the work of Abraham Seidenberg [20]). The major problem when dealing with RCFs is the complexity of the algorithm. Thus, we can find abundant literature aiming at improving the efficiency of the quantifier elimination in these fields. Chronologically, we have the work of Kreisel-Krivine [15], Collins [5], Heintz-Roy-Solernò [13], Renegar [19], who gave a double exponential algorithm, and more recently, the best algorithm (w.r.t. worst-case complexity) for this problem appeared in [2]. Some of these algorithms have been implemented in some theorem provers. For example, John Harrison [10] did this in HOL using a variant of the Kreisel-Krivine algorithm. In Coq also, Assia Mahboubi and Loïc Pottier [16] deal with the univariate polynomials having a degree at most 3 using the Tarski-Seidenberg principle described in [3].

6 Conclusion

6.1 Summary

In this work, several goals have been achieved. First, we have proved the decidability of the first order theory of ACFs (for the univariate case) in an original way, i.e. using a purely constructive style (without excluded middle and applying well-founded induction schemes). From this proof, we have been able to extract a quantifier elimination algorithm for the first order theory of ACFs. Next, we have described a general method to carry out this algorithm in a DS using a CAS for some computations (essentially the gcd) in a purely skeptical way. Finally, we have presented an implementation that we developed in the framework of Coq and Maple, as well as some examples that can be dealt with. If importing computations from a CAS to enhance the

computational power of a DS can be considered as a routine these days, this implementation shows concretely how CAS computations can be also used to enhance the power of automation of DSs.

6.2 Future work

As future work, we would like to deal with multivariate polynomials. However, our procedure cannot be directly extended to do so. Indeed, for example, even if we consider multivariate polynomials of $K[X,Y]$ as recursive univariate polynomials of $K[X][Y]$, our procedure cannot be recursively reused because although K is a field, $K[X]$ is not. Thus, to generalize our work to the multivariate case, we have to deal with all the variables simultaneously. An initial idea could be to transform the problem into one of solving a system of polynomial equations (the inequations are converted into equivalent equations using the Rabinowitsch trick, i.e. $x \neq y \equiv \exists z.(x-y)z+1=0$). Then we could use, for instance, Gröbner bases (whose computation can be also imported from Maple) to get equivalent triangular systems, which can be solved (by substitution), or Hilbert's *Nullstellensatz* (using, more appropriately, the *weak Nullstellensatz* corollary stating that I is a proper ideal over $K[X_1,\ldots,X_n]$ iff there exists a common zero for all polynomials in I) building the ideal generated by the polynomials of the system and showing that it is a proper ideal.

Another extension of this work could be to consider RCFs. Actually, a major part of our algorithm could be reused (almost everything except when concluding with the definition of ACF or with proposition 2.2) to fit with a quantifier elimination for RCFs. On those fields, we could also extend the algebra dealing with the symbol $>$. Such an extension would be interesting not so much for the method itself (as said in section 5, the literature is really prolific in this domain) but always in the way of carrying it out in a DS using a CAS. This allows us to understand how, in the particular framework of Coq and Maple, this work would be rather orthogonal to [16] even if this aims at dealing with the same problem.

Acknowledgements. We would like to thank Thierry Coquand who helped us in the integration of the procedure in a DS using a CAS (in particular, for dealing with the gcd proofs by means of Bézout's theorem). Thanks also to Hervé Perdry whose many discussions regarding quantifier elimination over RCFs inspired us in the design of this method for ACFs.

References

[1] Henk Barendregt and Arjeh M. Cohen. Electronic Communication of Mathematics and the Interaction of Computer Algebra Systems and Proof Assistants. *Journal of Symbolic Computation (JSC)*, 32(1/2):3–22, July/August

2001.

[2] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. On the Combinatorial and Algebraic Complexity of Quantifier Elimination. *Journal of the ACM.*, 43(6):1002–1045, November 1996. Extended abstract in Proc. 35th Symposium of foundation of Computer Science, (1994).

[3] Jacek Bochniak, Michel Coste, and Marie-Françoise Roy. *Real Algebraic Geometry*, volume 36 of *Ergebnisse der Mathematik und ihrer Grenzgebiete. 3. Folge / A Series of Modern Surveys in Mathematics.* Springer-Verlag, 1998. ISBN 3-540-64663-9.

[4] Bruce W. Char, Keith O. Geddes, Gaston H. Gonnet, Benton L. Leong, Michael B. Monagan, and Stephen M. Watt. *The* Maple V *Language Reference Manual.* Springer-Verlag, New York, 1991. ISBN 0387976221.

[5] George E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Second GI Conference on Automata Theory and Formal Languages*, volume 33, pages 134–183. Springer-Verlag LNCS, 1976.

[6] David Delahaye. A Tactic Language for the System Coq. In *Proceedings of Logic for Programming and Automated Reasoning (LPAR), Reunion Island (France)*, volume 1955, pages 85–95. Springer-Verlag LNCS/LNAI, November 2000. http://cedric.cnam.fr/~delahaye/publications/LPAR2000-ltac.ps.gz.

[7] David Delahaye and Micaela Mayero. `Field`: une procédure de décision pour les nombres réels en Coq. In *Journées Francophones des Langages Applicatifs, Pontarlier (France)*. INRIA, Janvier 2001. http://cedric.cnam.fr/~delahaye/publications/JFLA2000-Field.ps.gz.

[8] David Delahaye and Micaela Mayero. Dealing with Algebraic Expressions over a Field in Coq using Maple. *Journal of Symbolic Computation (JSC)*, 2005. To appear.

[9] Herman Geuvers, Freek Wiedijk, Jan Zwanenburg, Randy Pollack, and Henk Barendregt. The "Fundamental Theorem of Algebra" Project, 2000. http://www.cs.kun.nl/gi/projects/fta/.

[10] John Harrison. *Theorem Proving with the Real Numbers.* Springer-Verlag, 1998. ISBN 3-540-762566-6.

[11] John Harrison. Complex Quantifier Elimination in HOL. In Richard J. Boulton and Paul B. Jackson, editors, *TPHOLs 2001: Supplemental Proceedings*, pages 159–174. Division of Informatics, University of Edinburgh, 2001. Published as Informatics Report Series EDI-INF-RR-0046. Available on the Web at http://www.informatics.ed.ac.uk/publications/report/0046.html.

[12] John Harrison and Laurent Théry. A Skeptic's Approach to Combining HOL and Maple. *Journal of Automated Reasoning*, 21:279–294, 1998.

[13] Joos Heintz, Marie-Françoise Roy, and Pablo Solernò. Sur la complexité du principe de Tarski-Seidenberg. *Bull. Soc. math. France*, 118:101–126, 1990.

[14] Doug Ierardi. Quantifier Elimination in the Theory of an Algebraically-closed Field. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 138–147. ACM Press, 1989.

[15] Georg Kreisel and Jean-Louis Krivine. *Éléments de logique mathématique: théorie des modèles.* Dunod, 1964.

[16] Assia Mahboubi and Loïc Pottier. Élimination des quantificateurs sur les réels pour Coq. In *Journées Francophones des Langages Applicatifs, (France).* INRIA, Janvier 2002.

[17] Kurt Mehlhorn. The Reliable Algorithmic Software Challenge RASC. In *Computer Science in Perspective*, volume 2598 of *Lecture Notes in Computer Science*, pages 255–263. Springer-Verlag, 2003.

[18] Hervé Perdry. *Aspects constructifs de la théorie des corps valués.* PhD thesis, Université de Franche-Comté, Décembre 2001.

[19] James Renegar. On the Computational Complexity and Geometry of the First Order Theory of the Reals. *Journal of Symbolic Computation (JSC)*, 13(3):255–352, 1992.

[20] Abraham Seidenberg. A New Decision Method for Elementary Algebra. *Annals of Mathematics*, 60:365–374, 1954.

[21] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, 1951. Previous version published as a technical report by the RAND Corporation, 1948; prepared for publication by J. C. C. McKinsey.

[22] The Coq Development Team. *The Coq Proof Assistant Reference Manual Version 8.0.* INRIA-Rocquencourt, January 2005. http://coq.inria.fr/doc-eng.html.

A Proofs

In the following and according to our convention, the notation $P(x_i)$ will denote, by default, the product of the polynomials $P(X)$ and $x_i(X)$, whereas under a quantifier x, it will denote the product of $P(x)$ and $x_i(x)$, i.e. the values of the corresponding functions of $P(X)$ and $x_i(X)$ at x; however, a local exception will be made in the proofs of propositions 2.2 to 2.4, where it will denote the value of the corresponding function of $P(X)$ at x_i.

The main theorem that we need in our proofs is Bézout's theorem (we do not give the proof of this theorem which can be found in any Algebra book):

Theorem A.1 (Bézout's theorem) *Let P and Q be two non-zero polynomials. Let $G = \mathsf{gcd}(P, Q)$. There exist two polynomials A and B, s.t. $AP + BQ = G$.*

Proposition 2.1 (Common roots) *Let P_1 and P_2 be two non-zero polynomials. Let $G = \mathsf{gcd}(P_1, P_2)$. We have: $\exists x.P_1 = 0 \wedge P_2 = 0$ iff $\exists x.G = 0$.*

Proof. If P_1 or P_2 is a constant polynomial then $G = 1$ and the theorem trivially holds; otherwise, we have:

- $\Rightarrow$: Let x_0 be a root of P_1 and P_2. We have:
  $P_1 = (X - x_0)Q_1$
  $P_2 = (X - x_0)Q_2$
  Using Bézout's theorem (theorem A.1), there exist A and B s.t.:
  $G = AP_1 + BP_2 = A(X - x_0)Q_1 + B(X - x_0)Q_2 = (X - x_0)(AQ_1 + BQ_2)$
  i.e. x_0 is also a root of G.
- $\Leftarrow$: Let x_0 be a root of G. We have $G = (x - x_0)G_1$ and
  $P_1 = GQ_1 = (X - x_0)G_1Q_1$
  $P_2 = GQ_2 = (X - x_0)G_1Q_2$
  i.e. x_0 is also a root of P_1 and P_2.

□

Proposition 2.2 (Existence of a non-root value) *Let Q be a non-zero polynomial. We have: $\exists x.Q \neq 0$.*

Proof.

- $\mathsf{deg}(Q) = 0$: $Q = a \neq 0$, where a is a constant, and any x fits.
- $\mathsf{deg}(Q) > 0$: we know that $\exists x.1 + Q = 0$ (by definition of ACF since $\mathsf{deg}(Q) = \mathsf{deg}(1 + Q) > 0$). Let x_0 be a value verifying this proposition, i.e. s.t. $1 + Q(x_0) = 0$. Then x_0 is s.t. $Q(x_0) = -1 \neq 0$.

□

Proposition 2.3 (Roots of relatively prime polynomials) *Let P and Q be two relatively prime non-zero polynomials (i.e. $\mathsf{gcd}(P, Q) = 1$). We have: $\exists x.P = 0 \wedge Q \neq 0$ iff $\exists x.P = 0$.*

Proof.

- $\Rightarrow$: Let x_0 be s.t. $P(x_0) = 0$ and $Q(x_0) \neq 0$. Trivially, we have $P(x_0) = 0$.
- $\Leftarrow$: We have that $\mathsf{gcd}(P, Q) = 1$. Using Bézout's theorem (theorem A.1), there exist A and B s.t. $AP + BQ = 1$. Let x_0 be s.t. $P(x_0) = 0$. We have $B(x_0)Q(x_0) = 1$ which implies that $Q(x_0) \neq 0$.

□

Proposition 2.4 (Roots of quotient and gcd) *Let P and Q be two non-zero polynomials. Let $G = \mathsf{gcd}(P, Q)$ and P_1 be a polynomial s.t. $P = GP_1$. We have: $\exists x.P = 0 \wedge Q \neq 0$ iff $\exists x.P_1 = 0 \wedge G \neq 0$.*

Proof.

- $\Rightarrow$: Let x_0 be s.t. $P(x_0) = 0$ and $Q(x_0) \neq 0$. Let P_1 and Q_1 be s.t.
  $P = GP_1$ and $Q = GQ_1$.
  As $Q(x_0) \neq 0$, we have $G(x_0) \neq 0$ and then $P_1(x_0) = 0$.
- $\Leftarrow$: Let x_0 be s.t. $P_1(x_0) = 0$ and $G(x_0) \neq 0$.
  As $P_1(x_0) = 0$, we have $P(x_0) = 0$. Using Bézout's theorem (theorem A.1),

we have $B(x_0)Q(x_0) = G(x_0) \neq 0$ which implies $Q(x_0) \neq 0$.

□

Proposition 2.5 (Degree decrease) *Let P and Q be two non-zero polynomials. Let $G = \mathsf{gcd}(P, Q)$ and P_1 be a polynomial s.t. $P = GP_1$. If P and Q are not relatively prime then $\mathsf{deg}(P_1) < \mathsf{deg}(P)$.*

Proof. If P and Q are not relatively prime then we have $\mathsf{deg}(G) > 0$ and trivially $\mathsf{deg}(P_1) \leq \mathsf{deg}(P) - 1$. □