A formal quantifier elimination for algebraically closed fields

Cyril Cohen, Assia Mahboubi

A formal quantifier elimination for algebraically closed fields

Cyril Cohen, Assia Mahboubi

INRIA Saclay – Île-de-France
LIX École Polytechnique
INRIA Microsoft Research Joint Centre
[Cyril.Cohen|Assia.Mahboubi]@inria.fr

Abstract. We prove formally that the first order theory of algebraically closed fields enjoy quantifier elimination, and hence is decidable. This proof is organized in two modular parts. We first reify the first order theory of rings and prove that quantifier elimination leads to decidability. Then we implement an algorithm which constructs a quantifier free formula from any first order formula in the theory of ring. If the underlying ring is in fact an algebraically closed field, we prove that the two formulas have the same semantic. The algorithm producing the quantifier free formula is programmed in continuation passing style, which leads to both a concise program and an elegant proof of semantic correctness.

1 Introduction

Quantifier elimination is a standard way of proving the decidability of first order theories. In this paper, we investigate the formalization of quantifier elimination, and decidability for the first order theory of *algebraically closed fields*, inside the Coq proof assistant [4]. The work does not address the problem of implementing a fast proof producing decision procedure. Our motivation is to enrich an existing hierarchy of algebraic structures [8] with a decidability result. This allows in particular to justify case analysis of the validity of first-order statements, in the context of a constructive formal development. In this work, we follow the proof given in standard references [2]. Yet we diverge from the usual expositions of the algorithm using continuation passing style to rephrase and prove quantifier elimination in a more elegant way.

Our work can be read online at the following URL : `http://perso.crans.org/cohen/closedfields`[1].

The article is composed of three parts. First, we reduce quantifier elimination in discrete structures to the elimination of a single existential quantifier. We also build the boolean decision procedure resulting from quantifier elimination. Then, we establish an algebraic caracterisation of any existential formula with a single quantifier. Finally, we show how to compute a quantifier free formula from this caracterisation, using a continuation passing style formula transformation.

[1] It can be run using Coq v8.2 and SSReflect v1.2

2 Quantifier elimination and decidability

2.1 Preliminaries

In this section we recall some standard definition, essentially following [11], and introduce some notations needed in the sequel.

Syntax : Signature, Terms, Formulas. In all what follows, we consider *signatures* of the form: $\Sigma = \mathcal{C} \cup \mathcal{F} \cup \mathcal{R}$, formed of a finite set $\mathcal{C}$ of constant symbols, a finite set $\mathcal{F}$ of functions symbols with arity, and a finite set $\mathcal{R}$ of relations symbols with arity. Given such a signature Σ and countable set of variable $\mathcal{V}$, *terms* are inductively defined as: variables in $\mathcal{V}$ and constants in $\mathcal{C}$ are terms, other terms being of the form $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$ is function with arity n and $t_1 \ldots t_n$ are terms. A term is *closed* if no variables occur in it. We write $\mathcal{T}(\Sigma, \mathcal{V})$ for terms, and $\mathcal{T}(\Sigma)$ for closed terms.

The *atomic formulas* of a signature Σ are the strings $t_1 = t_2$ where t_1, t_2 are any terms, and $R(t_1, \ldots, t_n)$ where $R \in \mathcal{R}$ is a relation with arity n. The *first order language* of Σ is the set of all first order formulas with these atoms. First order formulas of Σ are recursively defined by: atomic formulas are first order formulas, other formulas being of the form $\neg f$, $f_1 \wedge f_2$, $f_1 \vee f_2$, $\exists x, f$, $\forall x, f$, $f_1 \Rightarrow f_2$, where $f, f_1, f_2 \in F$ A formula is *closed* if no variables occur in it. We write $\mathcal{F}(\Sigma, \mathcal{V})$ for formulas, and $\mathcal{F}(\Sigma)$ for closed formulas. We call *theory* over Σ any set of closed formulas. And we use the classic $\vdash$ predicate to denote provability: $T \vdash \psi$ means that we can deduce ψ from formulas in T.

A theory T *admits quantifier elimination* if, for every $\phi(\boldsymbol{x}) \in \mathcal{F}(\Sigma, \mathcal{V})$, there exists $\psi(\boldsymbol{x}) \in \mathcal{F}(\Sigma, \mathcal{V})$ such that ψ is quantifier free and

$$T \vdash \forall \boldsymbol{x}, ((\phi(\boldsymbol{x}) \Rightarrow \psi(\boldsymbol{x})) \wedge (\psi(\boldsymbol{x}) \Rightarrow \phi(\boldsymbol{x})))$$

However, we'll use a *semantic* characterization of quantifier elimination, defined in the end of then section about semantics.

Semantics : Σ-structures, Models. For any signature $\Sigma = \mathcal{C} \cup \mathcal{F} \cup \mathcal{R}$, a Σ-*structure* is the pair of a set E called the *domain*, and an *interpretation function* I assigning an element of E to each constant symbol in $\mathcal{C}$, a function $E^n \to E$ to each function symbol in $\mathcal{F}$ with arity n, and an n-ary relation on E (i.e. a subset of E^n) to each relation symbol in $\mathcal{R}$ with arity n.

For any Σ-structure A, any term $t(\boldsymbol{x})$, and any list e of values in the domain of A at least as long as $\boldsymbol{x}$, we define inductively $[\boldsymbol{t}(\boldsymbol{x})]_{A,e}$ as

- if $\boldsymbol{t}(\boldsymbol{x})$ is x_i, then $[\boldsymbol{t}(\boldsymbol{x})]_{A,e} = e_i$
- if $\boldsymbol{t}(\boldsymbol{x})$ is c for some $c \in \mathcal{C}$, then $[\boldsymbol{t}(\boldsymbol{x})]_{A,e} = I(c)$
- if $\boldsymbol{t}(\boldsymbol{x})$ is $f(\boldsymbol{s}(\boldsymbol{x}))$ where $f \in \mathcal{R}$, and where $\boldsymbol{s}$ are terms with variables $\boldsymbol{x}$, then $[\boldsymbol{t}(\boldsymbol{x})]_{A,e} = I(f)([\boldsymbol{s}(\boldsymbol{x})]_{A,e})$

For any Σ-structure A, any atomic formula $\phi(\boldsymbol{x}) = R(\boldsymbol{t}(\boldsymbol{x}))$ where $R \in \mathcal{R}$, and where $\boldsymbol{t}$ are terms with variables $\boldsymbol{x}$, and any a list e of values in the domain of A at least as long as $\boldsymbol{x}$, if $([\boldsymbol{t}(\boldsymbol{x})]_{(A,e)})$ is in $I(R)$, we say that A is *a model* of ϕ, denoted by $A, e \models \phi$. This definition is extended to any first order formula ϕ by induction of the structure on ϕ (see [11] for the exact statement). We say that a Σ-structure A is a *model* of a theory T, denoted $A \models T$, if and only if $\forall \phi \in T, A \models \phi$.

We say that two formulas f_1 and f_2 a *equisatisfiable* in a given model M if for any context e, $M, e \models f$ if and only if $M, e \models f$.

We say that a theory T *admits semantic quantifier elimination*, if for every $\phi \in \mathcal{F}(\Sigma)$, there exists $\psi \in \mathcal{F}(\Sigma)$ such that ψ is quantifier free and for any model M of T, and for any list e of values, $M, e \models \phi$ iff $M, e \models \psi$. This is what we'll prove instead, we discuss further the appropriateness of these choices in conclusion.

The theory of algebraically closed fields. The signature we use in this paper[2] to define the theory of fields (and algebraically closed fields) is $\Sigma_{\text{Fields}} = \{0, 1\} \cup \{-, .^{-1}, +, *\} \cup \emptyset$ (so the only atoms are equalities). We will also use $\Sigma_{\text{Rings}} = \{0, 1\} \cup \{-, +, *\} \cup \emptyset$. We call *first order theory of algebraically closed field*, the set $T_{\text{ClosedFields}}$ of axioms of fields plus an axiom scheme explaining that any monic univariate polynomial with coefficient in an algebraically closed field has a root.

This axiom scheme $(A_n)n \in \mathbb{N}$ defines a countable number of axioms, one per degree of polynomial:

$$(A_n) := \forall a_0, \ldots a_{n-1}, \exists x, x^n + a_{n-1}x^{n-1} \cdots + a_1 x + a_0 = 0$$

The following result is attributed to Tarski:

Theorem 1. *$T_{\text{ClosedFields}}$ admits quantifier elimination.*

The corresponding geometrical formulation of this result, stating that projections of constructible sets are constructible sets is known as Chevalley's Constructibility theorem [7].

2.2 Formalization issues

In a type theoretic proof assistant like the Coq system, it is a common practice to define the interface of an algebraic structure using *record types*. Here is an example of a possible definition for commutative groups:

```
Record zmodule := Zmodule{
  M : Type;
  zero : M;
```

[2] In the Coq formalization, there is an extra unary relation symbol for units, because the field theory is built by extending the one of rings with units.

```
  opp : M -> M;
  add : M -> M -> M;
  _ : associative add;
  _ : commutative add;
  _ : left_id zero add;
  _ : left_inverse zero opp add}.
```

Packaging such structures can be delicate, and record nesting should be used to achieve sharing and inheritance between structures [9, 8]. One can define similar specifications for more complex algebraic structures like `ring`, `integralDomain` and `field`. In our setting [8], the structure of algebraically closed field is formally defined by packing a structure of field with the following extra axiom schema:

```
Definition ClosedField.axiom (R : ring) := forall n P,
  n > 0 -> exists x : R, x ^+ n = \sum_(i < n)(P i) * (x ^+ i).
```

where the notations `x ^+ n` stands for x^n, and the right hand side of the equation is an iterated sum [5] forming the polynomial expression whose coefficients are given by the `(P : nat -> R)` function. So that in the end, we get back the type `closedField` of closed fields.

The syntactic equality predicate on such algebraic structures need not to be defined, as Coq provides such a default equality on any type. In Coq this equality is not decidable in general, and one cannot base a case analysis on two terms being equal or not. However in all what follows, we restrict our study to the case of *discrete* structures, in particular discrete fields. This means that we assume that there is a boolean equality test exactly reflecting Coq equality on the terms. For instance a classical formalization of real numbers could fit this framework through the assumption of a boolean equality test.

The definition of an algebraic structure interface type like `zmodule` can be seen as the definition of a signature Σ, together with some axioms E, shallow embedded in Coq logic. In the case of `zmodule`, the signature is $\{0\} \cup \{-, +\} \cup \emptyset$, and the axioms are the expected ones. Populating such a type, i.e. building an element `(Z : zmodule)` is providing a carrier and an interpretation function, hence a structure Z, together with a proof that it satisfies the set of axioms E_{zmodule}, i.e. that $Z \models E_{\text{zmodule}}$. So the inhabitants of the type `closedField` are a shallow-embedding of the models of $T_{\text{ClosedFields}}$. This is however not sufficient to state the quantifier elimination theorem we are interested in. For this purpose, we need to provide an abstract representation of first order formulas, to be able to quantify over such objects.

We describe the terms over Σ_{Fields} using the following inductive type:

```
Variable T : Type.
Inductive term : Type :=
| Var of nat (* variables *)
| Const of T (* constants *)
| Add of term & term (* addition *)
| Opp of term (* opposite *)
| Mul of term & term (* product *)
```

```
| Inv of term (* inverse *)
```

where we reflect division by the product by an inverse. Similarly, the first order formulas over Σ_{Fields} for the same `T` type parameter are defined by:

```
Inductive formula : Type :=
| Bool of bool
| Equal of term & term
| And of formula & formula
| Or of formula & formula
| Implies of formula & formula
| Not of formula
| Exists of nat & formula
| Forall of nat & formula.
```

where quantifiers explicitly take as argument the name of the variable they bind. We now define a Coq predicate `holds: forall F : field, seq F -> formula F -> Prop`, such that `(holds F e f)` is $F, e \models f$. This predicate is defined following the definition of the $\_, \_ \models \_$ predicate we mention in section 2.1. and requires first defining an `eval: forall F : field, seq F -> term F -> F` function interpreting terms as element in the model with respect to a context, defined following the definition of $[\_]_{(\_,\_)}$.

For instance, the interpretation of the abstract formula:

```
'forall 'X_0, 'forall 'X_1, 'forall 'X_2, 'exists 'X_3,
  'X_0 * 'X_3 * 'X_3 + 'X_1 * 'X_3 + 'X_2 == 0 : formula F
```

where some notations pretty-print the constructors of the `formula` inductive type, is the Coq proposition:

```
forall a b c, exists x, a * x * x + b * x + c = 0 : Prop
```

For any `T : Type`, it is straightforward to test if an inhabitant of `formula T` is quantifier free: we just recursively test that this element does not feature any `Exists` or `Forall` constructor. This results in a boolean test:

```
Definition qf_form : forall T :Type, formula T -> bool.
```

Now the Coq theorem we prove is that there exists a transformation:

```
Definition q_elim : forall F : closedField, formula F -> formula F
```

such that:

```
Lemma q_elim_wf : forall (F : closedField) (f : formula F),
  qf_form (q_elim f).
Lemma q_elimP : forall (F : closedField) (f : formula F),
  forall e : seq F, holds e f <-> holds e (q_elim f)
```

This latter theorem is a formalisation of the *semantic quantifier elimination*, assuming that the shallow formalization of models encompasses all models of a given structure.

2.3 Quantifier elimination by projection

For the discrete structures we are interested in, and more generally for first order theories with decidable atoms, the elimination of a single existential quantifier entails full quantifier elimination. We give here a formal account of this reduction, for the special case of the theory of discrete *rings*.

We first show that this problem is sufficiently general by encoding the theory of discrete fields into the one of discrete rings. Atoms of the abstract formulas of the discrete field theory are transformed in the following way:

- Right-hand sides are set at 0 by transforming $(t_1 = t_2)$ into $(t_1 - t_2 = 0)$
- Divisions in the left-hand sides are recursively removed by introducing quantifications over fresh variables: $C[t^{-1}] = 0$ is transformed into:
  $\forall x, ((x * t - 1 = 0 \wedge t * x - 1 = 0) \vee x = t \wedge \neg(\exists x, (x * t - 1 = 0 \wedge t * x - 1 = 0)) \implies (C[x] = 0)$

This transformation `to_rformula` is lifted recursively to any (non atomic formula). We prove that it preserves the semantic of ring formulas in any model of the structure of ring with units[3]. For sake of convenience, we introduce a special data-structure for normalized quantifier-free formulas. They can be represented in disjunctive normal form as:

$$\bigvee_{l \in L} \left(\bigwedge_{i \in I} t_i = 0 \wedge \bigwedge_{j \in J} \neg(t_j = 0) \right)$$

and hence encoded by a list (or sub-formulas in the disjunction), of pairs (one for positive and one for negated atoms) of lists of terms (the left hand sides) : `(seq ((seq term R)*(seq (term R)))`. We consider a field `F`, equipped with an operator:

```
Variable proj : nat -> seq (term F) * seq (term F) -> formula F.
```

whose first integer argument represents the name of a variable, second argument is a quantifier free conjunctive formula, and which computes a new abstract formula. This operator is meant to eliminate a quantifier from any formula of the form:

$$\exists x_n, \bigwedge_{i \in I} t_i = 0 \wedge \bigwedge_{j \in J} \neg(t_j = 0)$$

We hence require that on a formula on the ring signature, this operator always computes a quantifier free formula on the ring signature:

```
Definition wf_proj_axiom := forall i bc
    dnf_rterm bc -> qf_form (proj i bc) && rformula (proj i bc).
```

and that it computes a formula equivalent to its input:

[3] not required to be commutative.

```
Definition holds_proj_axiom :=
  forall n bc e,
  let (ex_n_bc := ('exists 'X_n, dnf_to_form [:: bc])%T in
  (holds e ex_n_bc) <-> (holds e (proj n bc)).
```

where `dnf_to_form` converts back the convenient representation we introduced for DNF quantifier free formulas to an inhabitant of the type `formula F`.

Under the assumptions that the `proj` operator satisfies the properties `wf_proj_axiom` and `holds_proj_axiom`, we can now prove that the field `F` enjoys a full quantifier elimination, meaning that we can implement the `q_elim` function of section 2.3. This quantifier elimination procedure proceeds by recursion on the structure of the formula, eliminating the inner-most quantifier:

- if it is an existential quantifier, the formula has the form $\exists x_n F$, where F is quantifier-free. It converts F is DNF and uses the fact that: $\exists x, \bigvee (\bigwedge p \wedge \bigwedge \neg q)$ is equivalent to $\bigvee (\exists x, (\bigwedge p \wedge \bigwedge \neg q))$ to map the `proj` operator on all the conjunctions of literals.
- if it is an universal quantifier, the formula has the form $\forall x_n F$, where F is quantifier-free. It converts $\forall F$ in $\neg\exists\neg F$, and converts $\neg F$ into DNF before using again the `proj` operator. This transformation is valid since the decidability of atoms implies that the full theory is classical.

The sequential representation of quantifier free formulas has been introduced to ease the DNF conversions, and their combination with negations in the case of universal quantifiers.

Finally we obtain a full formal proof that if a field is equipped with a `proj` operator, with a proof of the two `wf_proj_axiom` and `holds_proj_axiom` properties, then we can derive a correct quantifier elimination procedure `q_elim : formula F -> formula F`, which transforms any first order formula into a quantifier-free one, and such that the input and the output of the quantifier elimination are equisatisfiable in any model of a ring with units.

This generic process does not intend to provide an efficient decision procedure in general. The possibly numerous conversions in DNF make indeed this method very expensive, at least a tower of exponentials in the number of quantifiers. One can in general improve this complexity using decision procedure which are not based on quantifier elimination, and are more tractable, at least in practice. For instance, the standard way of deciding the theory of an algebraically closed field is to use Gröbner bases [6].

2.4 Decidability

The first order theory of a field is *decidable* if one can construct a *boolean* operator: `s : seq R -> formula R -> bool`, which reflects the satisfiability of any formula, i.e. satisfies the following property:

```
Variable F : field.
Definition DecidableField.axiom (s : seq F -> formula F -> bool):=
  forall e f, (holds e f) <-> (s e f = true).
```

This provides a computational characterization of decidability since s can be seen as a decision procedure for the first order theory of F.

Of course not all fields have a decidable first order theory: for instance the field theory of rational numbers is undecidable [17]. However quantifier elimination entails decidability for any first order theory with decidable atoms. For instance, in the theories we have considered, abstract atoms are equalities, and in all the theories we consider, equality is decidable. It is hence straightforward to construct by structural recursion a boolean test `qf_eval` which correctly reflects the validity of such a quantifier free abstract formula (and remains unspecified on quantified formulas): The correctness of this boolean test is expressed by the lemma:

```
Lemma qf_evalP : forall (e : seq R)(f : formula R),
  qf_form f -> (holds e f) <-> (qf_eval e f = true).
```

where `qf_form` tests that an abstract formula does not contain any quantifier. The function

```
Definition proj_sat e f := qf_eval e (q_elim f).
```

takes a formula, eliminates its quantifiers, and applies the boolean satisfiability test `qf_eval` on the result. It is a correct decision procedure as shown by the formal proof that it satisfies the `DecidableField.axiom` specification.

3 Polynomial arithmetic

In section 2.3, we have shown that quantifier elimination on the first order theory of fields reduces to the one over the signature of rings. This simplification leads to terms that are polynomial expressions in the variables instead of fractions. The price to pay is that we are no longer allowed to use division, even on constant coefficients, even if in fact they are interpreted as elements of a field. The semantic results we can use are hence the one provable in polynomial rings of the form $R[X]$, where R is only an integral domain. In this section, we give a formal account on the results of polynomial arithmetic needed in the sequel.

3.1 Representation

We represent univariate polynomials as lists of coefficients with lowest degree coefficients in head position. We require polynomials to be in normal form, in the sense that the last element of the list is never zero. Hence the type `{poly T}` of polynomials with coefficients in the type `T` is a so-called sigma type, which packages a list, and a proof that it last element is non zero. The zero polynomial is therefore represented by the empty list.

It is convenient and standard to define the degree of a univariate monomial as its exponent, except for the zero constant, whose degree is set at $-\infty$. Then the degree of a polynomial is the maximum of the degree of its monomial. To avoid introducing option types, we simply work here with the size of a polynomial,

which is the size of its list. This lifts the usual definition of degree since in our case:

$$\text{size}(p) = \begin{cases} 0 & \text{, if and only if } p = 0 \\ \deg(p) + 1 & \text{, otherwise} \end{cases}$$

3.2 Pseudo-divisions, pseudo-gcd

As soon as coefficients are equipped with a structure of field, one can program the well-known algorithm of Euclidean division, and define and specify greater common divisor. Here we have a weaker assumption on the coefficients: we only benefit from ring operations, but we still know that this ring is an integral domain. If integral domains are a natural setting for the study of divisibility, Euclidean division is no more possible in general. Indeed this division algorithm involves division among the coefficients of the respective polynomials, which could not be possible inside the integral domain. However it might still remain doable if a sufficient power of the leading coefficient of the divisor divides all the coefficients of the dividend. For instance one cannot perform Euclidean division of $2X^2 + 3$ by $2X + 1$ in $\mathbb{Z}[X]$, but one can divides $2X^2 + 6$ by $2X + 1$ in $\mathbb{Z}[X]$. In the context of integral domains, Euclidean division should be replaced by *pseudo-division*.

Definition 1 (Pseudo-division). *Let I be an integral domain. Let $P = a_p X^p + \ldots a_0 \in I[X]$ and $Q = b_q X^q + \ldots b_0 \in I[X]$. We define the* pseudo-division *of P by Q as the Euclidean division of $b_q^d P$ by Q, where d is the smallest integer such that the division is possible inside $I[X]$.*

Note that d always exists and is at most $p - q + 1$. We denote it `scalp p q`. The Euclidean pseudo-division also returns the pseudo-quotient `divp` denoted `%/`, and the peusdo-remainder `modp` denoted `%%`, satisfying the following specification:

```
Lemma divp_spec : forall p q,
  (scalp p q)%:P * p = p %/ q * q + p %% q.
```

Note that in general, the correcting coefficient `scalp p q` could be smaller than a power of the leading coefficient. The smallest value is in fact a power of the content of the divisor. All the possible choices in the correcting factor lead to different (but associated) values in the pseudo-quotient and remainder.

We say that `p` *pseudo-divides* `q`, denoted `(p %| q)` if the pseudo-remained of `p` by `q` is zero. We recover some standard lemmas about divisibility like:

```
Lemma dvdp_mul : forall d1 d2 m1 m2 : {poly R},
  d1 %| m1 -> d2 %| m2 -> d1 * d2 %| m1 * m2.
```

The pseudo greatest common divisor `gcdp` is obtained by replacing division by pseudo-division in the Euclidean algorithm. This is not the optimal algorithm to compute such a greatest common divisor, which is a non trivial problem. We choose here a naive implementation, since at this point, we are not concerned with efficiency. However we recover standard properties of the standard greatest common divisor, like:

```
Lemma root_gcd : forall p q x,
  root (gcdp p q) x = root p x && root q x.
```

where `root p x` means that `p` evaluates to `0` at the value `x`.

3.3 Common roots, exclusive roots

Recall form section 2.3 that we aim at eliminating the existential quantifier from a formula of the form:

$$(1) \quad \exists x \in F, \bigwedge_{i=1}^{n} P_i(x) = 0 \wedge \bigwedge_{j=1}^{m} Q_j(x) \neq 0$$

Indeed, after the reduction from the first order theory of fields to the first order theory of ring, and the normalization of atoms, atoms are zero conditions on polynomial expressions. In other words, given two finite families of polynomials (P_i) and (Q_j), we need to decide if there exists a point in the underlying field which is a common root of the (P_i)s but root of no Q_j:

$$(1) \Leftrightarrow \exists x \in F, (\gcd_{i=1}^{n} P_i)(x) = 0 \wedge (\prod_{i=1}^{m} Q_i)(x) \neq 0$$

Given two polynomials P and Q with coefficients in an integral domain R, we introduce the greatest divisor of P coprime to Q, denoted $\mathrm{gdco}_Q(P)$ and recursively defined as:

$$\mathrm{gdco}_Q(P) = \begin{cases} 1 & \text{, if } P = 0 \wedge Q = 0 \\ 0 & \text{, if } P = 0 \wedge Q \neq 0 \\ P & \text{, if gcdp } P\ Q = 1 \\ \mathrm{gdco}_Q(P \ /\% \ \text{gcdp } P\ Q) & \text{, otherwise} \end{cases}$$

In particular, $\mathrm{gdco}_Q(P)$ satifies the following property:

$$\exists x \in R, P(x) = 0 \wedge Q(x) \neq 0 \Leftrightarrow \exists x \in R, \mathrm{gdco}_Q(P)(x) = 0$$

Introducing this $\mathrm{gdco}_Q(P)$ operator provides a new equivalent to (1):

$$(1) \Leftrightarrow \exists x \in F, \mathrm{gdco}_{(\prod_{i=1}^{m} Q_i)} \left(\gcd_{i=1}^{n} P_i\right)(x) = 0$$

which in particular simplifies the one used in [2]. But if the underlying field is algebraically closed, then any polynomial has a root as soon as it is non constant or null, hence:

$$(1) \Leftrightarrow \mathrm{size}\left(\mathrm{gdco}_{(\prod_{i=1}^{m} Q_i)} \left(\gcd_{i=1}^{n} P_i\right)\right) \neq 1 \quad (2)$$

But this is not a first order formula. Indeed, size, gcd and gdco are not expressible as terms, because they may depend on other variables than x.

For example, let's take the formula $\phi(y) := \exists x, x = 0 \wedge xy \neq 0$. We know that

$$\phi(y) \Leftrightarrow \text{size}\left(\text{gdco}_{xy}(x)\right) \neq 1$$

But $\text{gcdo}_{xy}(x) = \begin{cases} 0 & \text{if } y = 0 \\ 1 & \text{if } y \neq 0 \end{cases}$, so that $\begin{cases} \text{if } y = 0 & \phi(y) \Leftrightarrow \text{size}(0) \neq 1 \\ \text{if } y \neq 0 & \phi(y) \Leftrightarrow \text{size}(1) \neq 1 \end{cases}$.

Hence

$$\phi(y) \Leftrightarrow ((\text{size}(0) \neq 1) \wedge (y = 0)) \vee ((\text{size}(1) \neq 1) \wedge (y \neq 0))$$

And the same kind of transformations happens to size so that

$$\phi(y) \Leftrightarrow ((\neg\text{false}) \wedge (y = 0)) \vee ((1 \neq 1) \wedge (y \neq 0))$$

Coming back to the general case, we can see that the traduction of (2) into a first order formula uses case analysis of the form $c = 0$ on coefficients c (not just on variables) of the polynomials P_i and Q_i. In each case ($c = 0$ and $c \neq 0$), the formula we obtain is independent from the value of the variables. And we can iterate this transformations to get a great disjunction of cases on which we get a first order formula. This disjunction is still a first order formula.

In the next part we'll present the algorithm that systematizes this case analysis and reconstitution of formulas.

4 Quantifier elimination for algebraically closed fields

Let $\mathcal{P}, \mathcal{Q} \subset \mathcal{T}(\Sigma_{\text{Rings}}, \{x_1, \dots x_n\})$ two finite sets of terms. In this section, we describe how to effectively transform a formula :

$$\phi := \exists x_k, \bigwedge_{p \in \mathcal{P}} p = 0 \wedge \bigwedge_{q \in \mathcal{Q}} \neg(q = 0)$$

over Σ_{Rings} into a quantifier free formula ψ in $\mathcal{F}(\Sigma_{\text{Rings}}, \{x_1, \dots x_n\})$ such that:

$$\forall M \models T_{\text{ClosedField}}, \forall e \in M^n, (M, e) \models \phi \Leftrightarrow (M, e) \models \psi$$

Yet we have seen in section 3.3 than different values for the context lead to different candidates for the quantifier free formula ψ. It is still possible to construct such a ψ because we can construct an algebraic, quantifier free, and independent of models description of a finite partition of the space of parameters into cells, each cell corresponding to a uniform shape for ψ. The description of this partition is obtained by analyzing the tree of successive zero tests performed when computing the degrees and greatest prime divisors involved in the expression (2) of section 3.3.

The terms of $\mathcal{P} \cup \mathcal{Q}$ are algebraic expression over the $\{x_1, \dots, x_n\}$, with 0 and 1 constants. They can be seen as polynomials in $R[x_1, \dots, x_n]$. Up to ring theory, we can even reorder the subterms of t, to factorize the powers of x_k, At this syntactic level, we introduce the type of *formal polynomials*, defined as lists of ring terms:

```
Definition polyF (T : Type) := seq (term T).
```

We also define the function:

```
Definition abstrX (R : Ring) (i : nat) (t : term R) : (polyF R) :=
    ...
```

by induction on a `(t : term F)`, which computes the formal polynomial associated to `t` seen as a polynomial in the variable x_i. A formal polynomial t can be interpreted as a univariate polynomial $[p_t]_e$ given a large enough context:

```
Fixpoint eval_poly (R : Ring)(e : seq R) (pf : polyF R) :=
if pf is c :: qf then (eval_poly e qf)*'X + (eval e c)%:P else 0.
```

We need to define again all the operations we have used in the informal presentation of section 3 like the size and greatest common division, to make them operate of formal polynomials. We moreover expect this transformation to be semantically sound. For instance, for a function `(`f`: poly R -> A)`, with an arbitrary return type `A`, we could ask its formal counterpart `(`F_f`: term F -> A)` to satisfy:

$$\forall t \in \mathcal{T}(\Sigma, \{x_1, \dots x_n\}), \forall M \models T_{\text{ClosedFields}}, \forall e \in M^n, f([p_t]_e) = [F_f(t)]_e$$

This is unfortunately not possible: consider the size function on polynomials, applied to the polynomial x. According to the value assigned by a given context to x, the size of x will be either 0 or 1. But there is no way to encode a case analysis at the syntactic level of terms handled by a formal counterpart F_{size}. In fact, these formal functions on terms should return *lists of values*, reflecting all the possible tests the body of the polynomial function performs. Going back to section 3.3, the output quantifier free formula is a disjunction over all the different values of the degree, specified by the conditions leading to these respective values. This construction requires inspecting the invariants of the code of the functions, to prove the correctness and soundness of the generated conditions. This approach is the one usually described in the literature (see e.g. [2]).

To avoid this painful formula reconstruction, we diverge from this standard presentation. First, we transform polynomial operations to return *formulas* instead of terms. This allows to encode case analysis, using the simple construction:

```
Definition ifF (then else cond: formula F) : formula F :=
  ((cond /\ then) \/ ((~ cond) /\ else))%T.
```

Simultaneously, we concisely internalize the administration of conditions in the body of the formal counterpart by programming them in *continuation passing style* (CPS). The CPS version of a function `f : `A_1`->...-> `A_n`-> B` has the form `f_cps (k : B -> T) (`a_1`: `A_1`)...(`a_n`: `A_n`) : T`, where `k` is called the *continuation*. For example, the rewritten `size` function is

```
Fixpoint sizeT (k : nat -> formula F) (p : polyF) :=
  if p is c::q then
```

```
    sizeT (fun n => if n is m.+1 then k m.+2
                    else ifF (k 0%N) (k 1%N) (c == 0)%T) q
  else k 0%N.
```

which is a straightforward translation of a CPS version of the size function over the polynomials of section 3.1. Note that working with continuations means that the code handles the formulas to be output, instead of the arguments, and can hence feature the `ifF` construction to directly build the case disjunction in the language of formulas. The zero test on formal polynomial is then:

```
Definition isnull (k : bool -> formula F) (p: polyF) :=
  sizeT (fun n => k (n == 0%N)) p.
```

The semantic specification that a CPS function `fT : (B -> formula F)-> (A1->...-> An-> formula F)` builds the formal counterpart of a function `f : A1->...-> An-> B` is formally expressed by a lemma of the form:

```
Lemma fTP : forall (k : B -> formula F) (a1 : A1) ... (an : An),
forall (e : seq (term F)),
 qf_eval e (fT k a1 ... an) = qf_eval e (k (f a1 ... an)).
```

For example, `sizeT` is correct w.r.t. `size` since we prove that:

```
Lemma sizeTP : forall k p e,
  qf_eval e (sizeT k p) = qf_eval e (k (size (eval_poly e p))).
```

We transform and specify the `gdcop` operation of 3.3 in the same CPS way to obtain a gcdpT function, naturally extended to lists of formal polynomials by gdcopTs. And we can express a first quantifier elimination lemma, that takes a list of formal polynomials `ps` and a formal polynomial `q`:

```
Definition ex_elim_seq (ps : seq polyF) (q : polyF) : formula F :=
  gcdpTs (gdcopT q (sizeT (fun n => Bool (n != 1%N)))) ps.
```

where `Bool` is a contsturctor of `formula F`.

The code can be read using monadic operations (this is no more Coq code) as :

```
  p <- gcdpTs ps;
  d <- gdcopT q p;
  n <- sizeT d;
  return (n != 1)
```

The projection function required at section 2.3 in finally implemented as:

```
Definition proj (x : nat) (pqs : seq (term F) * seq (term F)) :=
  ex_elim_seq (map (abstrX x) (fst pqs))
    (abstrX x (\big[Mul/1%T]_(q <- snd pqs) q)).
```

where `\big` is a notation for iterated operators (cf [5]), which constructs the formal term representing the product of the polynomials coming from the list

`(snd pqs)`. Proving that the `proj` operator output quantifier free formulas is straightforward: we have to check that each continuation can only output quantifier free formula. This is done by a trivial case analysis on each of the continuations. Proving its semantic correctness, i.e. the `holds_proj_axiom` axiom of section 2.3, is a combination of the semantic correctness of the CPS functions with the results formally proved in section 3.3.

5 Conclusion

From a model-theoretic point of view, our approach however deserves further discussion. In this work, we weaken the syntactic characterization of quantifier elimination given in section 2.1 along two directions. The first one is that our shallow approach only capture a semantic characterization of the equivalence between formulas: working at the abstract level of equivalence within the first order language of fields would require a deep embedding of the first order provability predicate. In the current formalization, nothing prevent us to use higher-order features in our proofs of semantic equivalence. An other option would be to provide a formal proof of Gödel's completeness theorem [12]. This result has already been formalized within the Isabelle system [16]. It would be interesting to investigate the work needed to come up with a similar result in Coq. We could for instance rely on the structures already introduced for a Coq formal proof of Gödel's incompleteness theorem [14], and the constructive approaches to the Gödel's completeness result [3]. Yet this would also require us to deeper formalize the $\models$ relation, and to prove formally that our shallow approach captures all the models of a given structure. Again there are here at least two options. The first one is to consider only models that can be defined in Coq, which should lead to a straightforward proof. A more general option would be to consider all the models definable in set theory, possibly relying on previous works about the formalization of set theory in Coq [1, 18].

However, one of our main motivation was to provide a convenient framework for the user to get a proof-producing decision procedure for the first-order theory of algebraically closed fields. We have not proved correct an *efficient procedure*, but this was not our purpose. Efficient decision procedures for the theory of real closed fields usually best deal with the universal fragment of the theory using Gröbner bases computations. Harrison [10] has formalized both quantifier elimination for the theory of complex numbers, and a proof producing version of Buchberger algorithm [6]. An efficient Gröbner based tactic is also available in Coq [15] for complex numbers.

The reduction of quantifier elimination to the elimination of a single existential is a standard result. Nipkow proposes in [13] a modular framework to build decision procedures along this motive. However, the continuation passing style we use to feed the prerequisite existential elimination seems an original idea. This approach makes the programing and the specification elegant and concise. Moreover, the produced formulas are in general more compact than the one produced by an invariant-based approach.

To the best of our knowledge, the present work is the first to address formally quantifier elimination for a generic structure of algebraically closed fields. We plan to investigate how the method we have presented scales to the theory of real closed fields. This seems very natural since the standard elimination algorithms are very similar in their structure [2].

Acknowledgments The authors wish to thank Georges Gonthier for numerous comments and improvements. The proofs relating quantifier elimination and decidability were done in close collaboration with him, and he suggested the successful idea of using CPS-style to transform algorithms on polynomials into formula producing ones. The authors are also grateful to Thierry Coquand for his comments and suggestions on preliminary versions of this work.

References

1. Bruno Barras. Sets in coq, coq in sets. In *Proceedings of the 1st Coq Workshop*. Technical University of Munich Research Report, 2009.
2. Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
3. Stefano Berardi and Silvio Valentini. Krivine's intuitionistic proof of classical completeness (for countable languages). *Ann. Pure Appl. Logic*, 129(1-3):93–106, 2004.
4. Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development, Coq'Art: the Calculus of Inductive Constructions*. Springer-Verlag, 2004.
5. Yves Bertot, Georges Gonthier, Sidi Ould Biha, and Ioana Pasca. Canonical big operators. In *Theorem Proving in Higher-Order Logics*, volume 5170 of *LNCS*, pages 86–101, 2008.
6. B. Buchberger. Groebner Bases: Applications. In A. V. Mikhalev and G. F. Pilz, editors, *The Concise Handbook of Algebra*, pages 265–268. Kluwer Academic Publishers, Dordrecht, Netherlands., 2002.
7. Claude Chevalley and Henri Cartan. Schémas normaux; morphismes; ensembles constructibles. In *Séminaire Henri Cartan*, volume 8, pages 1–10. Numdam, 1955-1956. `http://www.numdam.org/item?id=SHC_1955-1956__8__A7_0`.
8. Georges Gonthier and François Garillot and Assia Mahboubi and Laurence Rideau. Packaging mathematical structures. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *TPHOLs*, volume 5674 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2009.
9. Herman Geuvers, Randy Pollack, Freek Wiedijk, and Jan Zwanenburg. A constructive algebraic hierarchy in Coq. *Journal of Symbolic Computation*, 34(4):271–286, 2002.
10. 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`.
11. Wilfried Hodges. *A shorter model theory*. Cambridge University Press, 1997.
12. Kurt Gödel. *Über die Vollständigkeit des Logikkalküls*. PhD thesis, University of Vienna, Austria, 1929.

13. Tobias Nipkow. Linear quantifier elimination. In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated Reasoning (IJCAR 2008)*, volume 5195 of *LNCS*, pages 18–33. Springer, 2008.
14. Russel O'Connor. *Incompleteness & Completeness, Formalizing Logic and Analysis in Type Theory*. PhD thesis, Radboud University Nijmegen, Netherlands, 2009.
15. Loic Pottier. Connecting gröbner bases programs with coq to do proofs in algebra, geometry and arithmetics. In *LPAR Workshops*, volume 418 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
16. Tom Ridge and James Margetson. A mechanically verified, sound and complete theorem prover for first order logic. In Joe Hurd and Thomas F. Melham, editors, *TPHOLs*, volume 3603 of *Lecture Notes in Computer Science*, pages 294–309, 2005.
17. Julia Robinson. Definability and decision problems in arithmetic. *Journal of Symbolic Logic*, 14:98–114, 1949.
18. Carlos T. Simpson. Formalized proof, computation, and the construction problem in algebraic geometry, 2004.