Programming and certifying the CAD algorithm inside the Coq system

Assia Mahboubi

INRIA Sophia Antipolis
2004, routes des Lucioles - B.P. 93
06902 Sophia Antipolis Cedex, France
Assia.Mahboubi@sophia.inria.fr

Abstract. A. Tarski has shown in 1975 that one can perform quantifier elimination in the theory of real closed fields. The introduction of the Cylindrical Algebraic Decomposition (CAD) method has later allowed to design rather feasible algorithms. Our aim is to program a reflectional decision procedure for the Coq system, using the CAD, to decide whether a (possibly multivariate) system of polynomial inequalities with rational coefficients has a solution or not. We have therefore implemented in Coq various computer algebra tools like gcd computations, subresultant polynomial or Bernstein polynomials.

Keywords. computer algebra, Bernstein polynomials, subresultants, CAD, Coq, certification.

1 Introduction

The Coq system [1] has been recently improved to become a system you can compute with : the introduction of a virtual machine to be able to compile terms (see [2]), the implementation of efficient tactics by reflection as well as the technical improvement of machines open the way to perform computer algebra "within" a proof assistant system like Coq, that is to say by programing, certifying and computing with Coq. We present here the development of a tactic called Tarski aiming to decide whether a system of inequalities on multivariate polynomials has a solution in $\mathbb{R}^n$, where $\mathbb{R}$ is the field of real numbers. This problem is known to be decidable since the early work of Alfred Tarski [3]. Anyway the complexity of the decision method he proposed (a tower of exponentials of length the number of variables involved) did not allow to deal with concrete problems.

The method of Cylindrical Algebraic Decomposition (CAD) was introduced by Collins in 1974 in [4] (reprinted in [5]) and has been extensively studied since. It enables a drastic jump in the complexity, since the computations are done in a time polynomial in the maximum degree of the polynomials involved and doubly exponential in the number of variables. It can even be specialised to provide a more efficient tool for systems of a particular shape (see for example [5] and [6], and more generally the works of these two last authors).

Dagstuhl Seminar Proceedings 05021
Mathematics, Algorithms, Proofs
http://drops.dagstuhl.de/opus/volltexte/2006/276

Anyway, this complexity is still big, and the algorithm is not a straightforward one. In fact, computer algebra systems does not always provide such a tool. For example Maple does only provide a procedure allowing the user to isolate the roots of univariate polynomials, called `realroot`. Up to our knowledge, quantifier elimination over real numbers by CAD has only been implemented in the Mathematica [7] computer algebra system. It is also available in the REDLOG system [8],which is an extension of the REDUCE computer algebra system, but this does not use CAD. Such an implementation has also been developed in the Axiom [9] system but the code seems to be deprecated by now. Nevertheless there also exists a system dedicated to this method, called QEPCAD [10]. The CAD algorithm proceeds by eliminating successively the variables of the initial problem, using a projection operator, which may vary. In QEPCAD, this projection operator can be customized by the user to suit best the problem.

Up to our knowledge, no such decision procedure is available at the present time in an other proof assistant, excepted an ongoing work (not yet documented) by S. McLaughlin and J.Harrison, in the HOL Light system [11]. We have already built an other prototype of such a decision tactic for real numbers, using a method due to Hörmander , and this work is described in [12]. This method is really an elementary one, using repeted applications of the intermediate value theorem, but its complexity is as bad as Tarski's one. This tool was using an external oracle programmed in Objective Caml, and proved the generated traces of the computations. Hence the algorithm was not proved correct.

Our aim in this project is to program a generic algorithm of CAD to build the expected decisions procedure, using Coq as a functional typed programming language. Then we will prove the correction of the required specifcations of the functions we have implemented, again in the Coq system. This will allow to build a *tactic*, that is to say an automatic tool which will solve the goals being conjunction of polynomial inequalities over real numbers. This should provide a useful tool for the proofs in real analysis and geometry, where such goals are very frequent and often tedious. But this would also show that the Coq system enables the certification of highly non-trivial computer algebra algorithms.

The main goal of this paper is to describe the algorithm we have implemented for the decision procedure and the related formalisation issues. The two first sections detail the unfolding of the algorithm, first for the univariate problem, then for the multivariate, general case. The third section is dedicated to a simple example. Finally we will mention the formalisation choices made for the basic objects handled during the computations : polynomials and rational numbers, before concluding.

2 Algorithm, univariate case

In this section and the next one, the aim is to describe the geometrical meaning of the cylindrical algebraic decomposition and the tools needed to make use of it. In this development, we follow the algorithms described in [13]. We will mainly recall the results that are stated in different chapters of this book (but it is our

own responsibility), and try to illustrate them with some examples or figures. The proofs we omit in these sections can always be found in [13].

2.1 Real closed fields

The underlying structure of the problem is the one of *real closed field*.

Definition 1 (Real closed field). *Let $(F, \leq)$ be a field totally ordered, such that $\leq$ is compatible with the field operations (F is then called an ordered field). F is a* real closed field *if:*

- *every positive element of F is the sum of some squares of elements of F (F is a* real field*)*
- *every polynomial in $F[X]$ of odd degree has a root in F.*

Example 1. $\mathbb{R}$, the field of real algebraic numbers,...

The following theorem states the main characteristic properties of a real closed field.

Theorem 1. *If $(F, \leq)$ is an ordered field, then the following properties are equivalent:*

- *F is real closed*
- *$F[i] = F[X]/(X^2 + 1)$ is an algebraically closed field*
- *F is a real field and F itself is the only real field which is algebraic over F*
- *For any $P \in F[X]$, for every $a < b \in F$, if $P(a)P(b) < 0$, then there exists $x \in F$ such that $a < x < b$ and $P(x) = 0$ (F has the* intermediate value property*).*

Furthermore, the theory of real closed field admits quantifier elimination in the language of real ordered fields :

Theorem 2 (Quantifier elimination). *Let $\Phi(Y)$ be a formula in the language of ordered fields with coefficients in an ordered ring D contained in the real closed field F. Then there is a quantifier free formula $\Psi(Y)$ with coefficients in D such that for every $y \in R^k$, $\Psi(y)$ is true if and only if $\Phi(y)$ is true.*

The problem we are interested in is a particular instance of quantifier elimination, where we are considering only existential formulae Φ.

Remark 1. In the sequel, the real closed field for which the decision procedure is built will be supposed to be *archimedian*. A decision procedure by cylindrical algebraic decomposition is also possible for the non archimedian real closed fields but the algorithms, using Thom encodings, are more intricate, and we are here mainly interested by providing such a tool for the field of real numbers. Hence this archimedian real closed field will from now on supposed to be the set of real numbers and denoted by $\mathbb{R}$.

To build a decision procedure from this theorem, we have to chose a field D over which sign and equality is *decidable*. In the sequel, the set of coefficients will be $\mathbb{Q}$, the set of rational numbers.

Anyway all what follows remain valid for any archimedian real closed field and any polynomials with coefficients in a decidable subring.

2.2 The univariate decision procedure

The univariate problem is the following : given a list of polynomials $P_1, \ldots, P_k \in \mathbb{Q}[X]$, and a list of sign conditions $\sharp_1, \ldots, \sharp_k \in \{<, =, >\}$, then we want to know whether or not there exists an $x \in \mathbb{R}$ satisfying the following system of inequalities:

$$P_1(x)\sharp_1 0 \wedge \cdots \wedge P_k(x)\sharp_k 0.$$

To answer this question, we will compute all the conjunctions of signs that can be realised by the signs of the polynomials at a certain real point. In fact, we will compute an *isolating list* for the family.

Definition 2 (Isolating list). *Let $\mathcal{P} = P_1, \ldots, P_k$ be a finite list of polynomials in $\mathbb{Q}[X]$. Let Roots be the set of all the (real) roots of this polynomials :*

$$Roots := \{x \in \mathbb{R} \quad | \quad \exists i, P_i(x) = 0\}.$$

An isolating list *for $\mathcal{P}$ is a list L of disjoint rational points or closed intervals with rational endpoints, such that each element of Roots belongs to an element of L and each element of L contains exactly one element of Roots.*

Hence the steps of the univariate decision procedure will be the following:

- Compute an ordered isolating list for the family. For each element of the list, remember the gcd of the polynomials it is a root thereof.
- For each element of the list, compute the sign of the polynomials which have a constant sign over this interval, simply by evaluating them at a rational point in the interval.
- Compute two rational points, one smaller than all the elements of the isolating list, and one bigger. Compute a rational point between every two consecutive elements of the isolating list. This is possible since the elements of the list are disjoint and have rational endpoints. We have built a list of rational points between the roots, where no polynomial of the family will be zero. Evaluate the polynomials at these rational points.

The vectors of sign conditions computed, first at each element of the isolating list, then at the rational points between roots, constitute an exhaustive sign table for the family of polynomials (thanks to the intermediate value property and to the correction of the isolating list).

2.3 Computation of isolating lists

The first step of the computation is the restriction of the problem to a finite interval of the real line. This is possible using a straightforward lemma establishing bounds on the real roots of a polynomial from its coefficients.

Lemma 1 (Cauchy bounds on roots). *Consider the polynomial :*

$$P = a_p X^p + \cdots + a_q X^q, \quad p > q, \quad a_p a_q \neq 0$$

Let

$$C(P) = \sum_{q \leq i \leq p} \frac{|a_i|}{|a_p|}$$

and

$$c(P) = \sum_{q \leq i \leq p} \frac{|a_i|}{|a_q|}$$

Then the absolute value of any root of P in $\mathbb{R}$ is smaller than $C(P)$ and bigger than $c(P)$.

Now we have to compute informations about the number of roots of a given polynomial on a given interval. This information can be read on the coefficients of a polynomial by looking at the number of *sign changes* in its coefficients, dropping the zeros.

Theorem 3 (Descartes'law of signs). *Let $P \in \mathbb{R}[X]$. Let $V(P)$ denote the number of sign changes in the list of coefficients of P and $pos(P)$ denote the number of positive real roots of P, counted with multiplicities. Then:*

- *$pos(P) \leq V(P)$*
- *$V(P) - pos(P)$ is even*

Then, the main ingredient for the refining of the relevant intervals will be the coefficients of the polynomial in special basis called *Bernstein basis*.

Definition 3 (Bernstein polynomials). *Let $c, d \in \mathbb{Q}$. The Bernstein polynomials of degree p for c, d are the:*

$$B_{p,i}(c,d) = \binom{p}{i} \frac{(X-c)^i (d-X)^{p-i}}{(d-c)^p} \quad 0 \leq i \leq p$$

Theorem 4 (Bernstein basis). *The Bernstein polynomials of degree p for c, d form a basis of the vector space of polynomials of degree $\leq p$.*

Descartes's law of signs applied to the special case of coefficients in a Bernstein basis gives the following result.

Theorem 5. *Let $P \in \mathbb{R}[X]$ of degree p, and $c < d \in \mathbb{Q}$. Let $V_b(P)$ denote the number of sign changes in the list of coefficients of P in the Bernstein basis of degree p for c, d. Then:*

- *If $V_b(P) = 0$ then P has no root on $]c, d[$*
- *If $V_b(P) = 1$ and $P(c)P(d) < 0$, then P has exactly one root on $]c, d[$*
- *Otherwise anything can happen (no root, a single one, several ones).*

Furthermore, if P has either no root or exactly one root in $]c, d[$, and P has no complex root in $\mathcal{C}(c, d-c) \cup \mathcal{C}(d, d-c)$, where $\mathcal{C}(x, r)$ is the complex disk of center $x \in \mathbb{R}$ and radius r, then :

- $V_b(P) = 0$ *iff* P *has no root on* $]c, d[$
- $V_b(P) = 1$ *iff* $P(c)P(d) < 0$, *in this case* P *has exactly one root on* $]c, d[$

This theorem provides a tool to test an interval for the construction of an isolating list of a single polynomial, using the first part of the statement. If we are in the first case, then we know that the interval is not relevant. If we are in the second case, we can put this interval in the isolating list for the polynomial.

If we are in the last case, then we have to split the interval $]c, d[$ and to work on each half, initiating a dichotomy process. The last part of the statement ensures that if we refine the initial interval enough, up to the point were we miss the possible roots in the complex plane, then we will have a precise answer, 0 or 1.

Notice that each time we refine an interval by splitting it into two parts, we need to compute again the coefficients of the polynomial in the new Bernstein basis associated to the new rational endpoints. There is in fact an efficient algorithm (linear in degree and bitsize of the coefficients) to compute both new lists of coefficients (one for each half), given the initial coefficients, the initial endpoints, and the point at which we will split (here the middle).

2.4 Description of the algorithm

Let us describe in details the computation of an isolating list for a family of two polynomials P_1 and P_2. We will denote by $Bern(P, a, c)$ the call to the function computing the number of sign changes in the list of coefficients of P in the Bernstein basis of degree $deg(P)$ for a and b.

- Compute the bounds for the roots of P_1 and P_2, then take the extrema A and B to get bounds on the roots of the family
- Compute an isolating list L_{P_1} for P_1 on $]A, B[$:
  - if $Bern(P_1, A, B) = 0$ then $L_{P_1} = \{\}$
  - if $Bern(P_1, A, B) = 1$ then $L_{P_1} = \{]A, B[\}$
  - else if $P_1(\frac{A+B}{2}) = 0$ then
    - compute L'_{P_1} an isolating list of P_1 on $]A, \frac{A+B}{2}[$,
    - compute L''_{P_1} an isolating list of P_1 on $]\frac{A+B}{2}, B[$,
    - $L_{P_1} = L'_{P_1} \cup L''_{P_1} \cup \{[\frac{A+B}{2}]\}$
  - and if $P_1(\frac{A+B}{2}) \neq 0$ then
    - compute L'_{P_1} an isolating list of P_1 on $]A, \frac{A+B}{2}[$,
    - compute L''_{P_1} an isolating list of P_1 on $]\frac{A+B}{2}, B[$,
    - $L_{P_1} = L'_{P_1} \cup L''_{P_1}$
- if $L_{P_1} = \{\}$ then compute an isolating list for P_2 on $]A, B[$
- else for each element u of L_{P_1}, compute the sign of P_2 on u:
  - if u is a single rational point then evaluate P_2
  - if $u =]c, d[$ then:
    - if $Bern(P_2, c, d) = 0$, then evaluate P_2 at $\frac{c+d}{2}$
    - if $Bern(P_2, c, d) = 1$ then we have to decide P_1 and P_2 have here a common root or not. Let G be the greatest common divisor of P_1 and P_2.

  - if $Bern(G, c, d) = 1$, then the root is common
  - if $Bern(G, c, d) = 0$, then refine u to get an interval (or a single point) u' such that P_2 has not root in u'
  - if $Bern(G, c, d) > 0$, then refine u to get an interval (or a single point) falling in one of the previous cases, by splitting u and testing its middle and the two open halves

  At that point we have built a refined isolating list L'_{P_1} for P_1, such that each element of L'_{P_1} is a root of P_2 or an interval over which P_2 has a constant sign, which has been computed
- For each interval between to consecutive elements of L'_{P_1}, compute an isolating list for P_2 and the sign of P_1 on these intervals. Merge these lists with L'_{P_1} to get L_{P_1,P_2}, an isolating list for the family.
- Compute the sign of P_1 and P_2 at A and B
- Compute the sign of P_1 and P_2 between to consecutive elements of L_{P_1,P_2}, by evaluating them at a rational point between them (the middle of the endpoints for example).

Remark 2. This is an outline of the method, and the implemented algorithm is slightly more complicated by some efficiency concerns : use of square-free parts when possible, remembering signs or gcd computations when it is useful...

3 Algorithm, multivariate case

In this section, we describe the algorithm of Cylindrical Algebraic Decomposition (CAD), which is the method to solve the existential decision problem for the theory of real numbers.

3.1 The multivariate problem

Now, we will deal with multivariate polynomials. Given a list $\mathcal{P}$ of polynomials $P_1, \ldots, P_k \in \mathbb{Q}[X_1, \ldots, X_n]$, and a list of sign conditions $\sharp_1, \ldots, \sharp_k \in \{<, =, >\}$, then we want to know whether or not there exists an $\overline{x} \in \mathbb{R}^n$ satisfying the following system of inequalities:

$$P_1(\overline{x})\sharp_1 0 \wedge \cdots \wedge P_k(\overline{x})\sharp_k 0.$$

To answer this question, we will again compute all the vectors of signs of length k which can be realized by the evaluation of $P_1, \ldots, P_k$ at a point $\overline{x} \in \mathbb{R}^k$. This requires the computation of an *adapted* partition of $\mathbb{R}^k$ into a finite number of cells, meaning that on every cell, every polynomial of the family $\mathcal{P}$ has a constant sign. This will be done recursively, by successive elimination of the variables. Here is the outline of the method :

$\mathbb{R}[X_1, \ldots, X_n]$ $\mathbb{R}[X_1, \ldots, X_{n-1}]$

$$\mathcal{P} = P_1, \ldots, P_s \xrightarrow[operator]{projection} \mathcal{Q} = Q_1, \ldots, Q_t$$

$$\text{CAD and signs for } \mathcal{P} \xleftarrow[phase]{lifting} \text{CAD and signs for } \mathcal{Q}$$

$\mathbb{R}^n$ $\mathbb{R}^{n-1}$

Given a family $\mathcal{P}$ of polynomials in n variables, we compute a (possibly larger) new family $\mathcal{Q}$, where X_n has been eliminated. The correction of the projection operator we use to eliminate X_n means that if we are given a certain partition $\mathcal{S}_{n-1}$ of $\mathbb{R}^{n-1}$ adapted to the new family $\mathbb{Q}$, we are able to compute a partition $\mathcal{S}_n$ adapted to the initial family $\mathcal{P}$. A cell $C_{\mathcal{P}} \subset \mathbb{R}^n$ of the new partition will be obtained by:

- taking a cell $D_{\mathcal{Q}} \subset \mathbb{R}^{n-1}$ from the decomposition adapted to $\mathcal{Q}$
- building the cylinder $D_{\mathcal{P}} \times \mathbb{R} \subset \mathbb{R}^n$
- cutting it in "a polynomial way", that is to say:
  - either taking the intersection of this cylinder with the graph of a polynomial function
  - or taking the semi-cylinder above (resp. under) the graph of a polynomial function
  - or taking a part of the cylinder delimited by the graph of two polynomial functions.

The elements of $\mathcal{S}_n$ are called the cells of level n. The set of successive partitions $\mathcal{S}_n, \ldots, \mathcal{S}_1$ is called a cylindrical decomposition of $\mathbb{R}^n$.

In the univariate case, the signs of a polynomial were determined by their evaluation at a rational point in the intervals of the isolating list not containing a root for this polynomial. The generalisation of this notion is the one of *cylindrical sample point*

Definition 4 (Cylindrical set of sample points). *Let $\mathcal{S} := \mathcal{S}_1, \ldots, \mathcal{S}_n$ be a cylindrical decomposition of $\mathbb{R}^n$. A cylindrical set of sample points of $\mathcal{S}$, $\mathcal{A} = \mathcal{A}_1, \ldots, \mathcal{A}_n$ is a list of n sets such that :*

- *for every i, $1 \leq i \leq n$, $\mathcal{A}_i$ is a finite subset of $\mathbb{R}^i$ which intersects every $S \in \mathcal{S}_i$.*
- *for every i, $1 \leq i \leq n-1$, $\pi_i(\mathcal{A}_{i+1}) = \mathcal{A}_i$, where π_i is the projection from $\mathbb{R}^{i+1}$ to $\mathbb{R}^i$, forgetting the last coordinate.*

Now we can describe the algorithm of cylindrical algebraic decomposition. It computes a cylindrical set of sample points for a cylindrical decomposition adapted to a family of polynomials and the set of each polynomial on each cell. This is an outline of the different steps, which will be described hereafter. We

will call $Elim_{X_n}$ the projection operator mentioned above, which eliminate the variable X_n from the family. We want to compute a cylindrical decomposition for $\mathcal{P} \subset \mathbb{Q}[X_1, \ldots, X_n]$. Note that the order of the variables is important since it determines the order of their elimination.

- Initialisation: set $C_n(\mathcal{P}) := \mathcal{P}$
- Elimination phase : compute $C_i(\mathcal{P}) := Elim_{X_{i+1}}(C_{i+1}(\mathcal{P}))$ for $i = n - 1, \ldots, 1$.
- Lifting phase:
  - Compute the sample points of the cells in $\mathcal{S}_1$ by characterising the roots of $C_1(\mathcal{P})$ and choosing a point in each interval (for example take the middle, see 2.4)
  - for every $i = 2 \ldots n$, compute the sample points of the cells of $\mathcal{S}_i$ from the sample point of $\mathcal{S}_{i-1}$: consider, for every sample point x of a cell in $\mathcal{S}_{i-1}$, the list L of non-zero polynomials $P_i(x, X_i)$ with P_i in $C_i(\mathcal{P})$. Characterise the roots of L and choose a point in each interval they determine.
- Output the sample points of the cells of level n and the signs of $P \in \mathcal{P}$ on the cells of level n.

Remark 3. The elimination phase corresponds of course to the application of the projection operator mentioned on the diagram.

The lifting phase builds the cells of level i given the set of cells of level $i-1$. We have claimed this is possible by taking a cell of level i, building the cylinder above and cutting it in a polynomial way. Anyway we know that the behaviour of all the polynomials in $C_i(\mathcal{P})$ is uniform on every cell of level i, with respect to the $i-1$ first coordinates. This means that, to study the polynomials of $C_i(\mathcal{P})$ over a cell $D \in \mathcal{S}_{i-1}$, we can choose an arbitrary point $x \in \mathbb{R}^{i-1}$ in the cell D, and study the polynomials $P(x, X_i)$, for $P \in C_i(\mathcal{P})$ instead. The result we will obtain on this line will remain valid for the whole cell. This is exactly what is meant by the fact that the cells of level i will be obtained by cutting the cylinders above the cells of level $i-1$ with graphs of *functions* (which are here moreover polynomials), which are in fact the roots of the polynomials in $C_i(\mathcal{P})$ above this cylinder. An example is described on Figure 1.

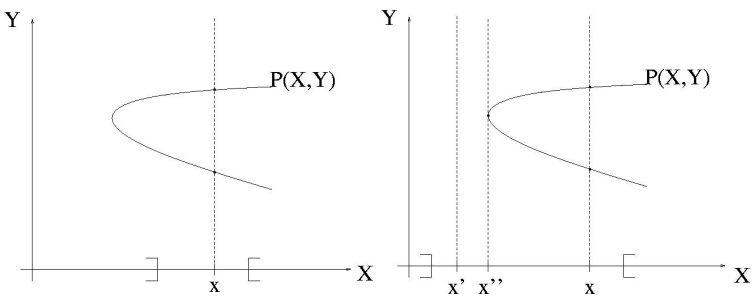

Fig. 1. On the right, the interval cannot be a cell of level one: $P(x, Y)$ has two roots, $P(x', Y)$ has no root, $P(x'', Y)$ has a single root. On the left it can be a cell.

3.2 The elimination phase

The elimination phase makes a heavy use of the *subresultants coefficients*. We will here define these quantities and explain where they occur.

Definition 5 (Sylvester-Habicht matrix, subresultant coefficient).
Let D be an integral domain and $P = p_n X^n + \ldots p_0$ belongs to $D[X]$ with $p_n \neq 0$ and $Q = q_m X^m + \ldots q_0$ belongs to $D[X]$ with $q_m \neq 0$.

The matrix $SH_j(P, Q)$ of $X^{q-j-1}P, \ldots, P, Q, \ldots, X^{p-j-1}Q$ in the monomial basis of $D[X]$ is called the $j^{th}-$Sylvester-Habicht matrix of P and Q.

$$\begin{bmatrix} p_n & \cdots & & & p_0 \\ & \ddots & & \iddots & \\ 0 & & p_n & \cdots & p_0 \\ 0 & & & & \\ & q_m & \cdots & & q_0 \\ & \iddots & & \ddots & \\ q_m & \cdots & q_0 & & 0 \end{bmatrix}$$

The j^{th} signed subresultant coefficient denoted by $sr_j(P, Q)$ is the determinant of the square matrix obtained by taking the first $n + m - 2j$ columns of $SH_j(P, Q)$.

Remark 4. Subresultants coefficients can also be defined as the leading coefficients of the so-called *subresultant polynomials* and this is used for an efficient, recursive computation of these coefficients. Anyway the definition we give here is equivalent and more concise, if not algorithmic.

Subresultants polynomials have been extensively studied (see for example [14]) and there is a wide range of possible definitions, inducing various algo-

rithms of computations. For a detailed description of the algorithms we have implemented for the computation of subresultants, again refer to [13].

We also define the set of truncations $Tru(Q)$ of a polynomial $Q \in \mathbb{Q}[X_1, \dots X_n]$ as a finite subset of $\mathbb{Q}[X_1, \dots X_{n-1}][X_n]$ as follows :

- if the leading coefficient of Q is an element of $\mathbb{Q}$, then $Tru(Q) = \{Q\}$
- else $Tru(Q) := \{Q\} \cup Tru(Q - lcoef_{X_k}(Q) X_k^{deg_{X_k}(Q)})$

Now we are ready to define the projection operator.

Definition 6 (Projection operator). *Consider a finite family $\mathcal{P} \subset \mathbb{Q}[X_1, \dots, X_{n-1}][X_n]$. $Elim_{X_n}(\mathcal{P})$ is the list of the following polynomials, when they are not in $\mathbb{Q}$:*

- $sr_j(R, \frac{\partial R}{\partial X_k})$ *for* $P \in \mathcal{P}, \quad deg_{X_k}(P) = p \geq 2, \quad R \in Tru(P),$ $j = 0 \dots deg_{X_k}(R) - 2$
- $sr_j(R, S)$ *for* $P, Q \in \mathcal{P}, \quad R \in Tru(P), \quad S \in Tru(Q),$ $j = 0 \dots min(deg_{X_k}(R), deg_{X_k}(S)) - 1$
- $lcoef(R)$ *for* $P \in \mathcal{P}, \quad R \in Tru(P)$.

Subresultants coefficients of couples of polynomials and of polynomials with their derivatives keep the necessary information while projecting. Adding the truncations is necessary to deal with the cells where certain leading coefficients of some polynomials are vanishing.

3.3 The lifting phase

For the lifting phase, given a point $x \in \mathbb{Q}[X_1, \dots, X_{i-1}]$ in a cell of level $i-1$, we study the polynomials of the family $C_i(\mathcal{P})$ over the real line above x, that is to say, for each $P \in C_i(\mathcal{P})$ we study the sign of $P(x, X_i)$.

For the univariate case, it was convenient to isolate a root of the family by an interval with rational endpoints. What generalises this is an *isolating parallelepiped*.

Definition 7 (Isolating parallelepiped). *A parallelepiped isolating $z = (z_1, \dots, z_i) \in \mathbb{R}^i$ is a list:*

$$(\mathcal{T}_1, I_1), \dots (\mathcal{T}_i, I_i)$$

where :

- $\mathcal{T}_1 \in \mathbb{Q}[X_1], \dots \mathcal{T}_k \in \mathbb{Q}[X_1, \dots, X_k]$.
- I_1 *is a closed interval with rational endpoints or a rational containing the root z_1 of $\mathcal{T}_1$ and no other root of $\mathcal{T}_1$.*
- I_2 *is a closed interval with rational endpoints or a rational containing the root z_2 of $\mathcal{T}_2(z_1, X_2)$ and no other root of $\mathcal{T}_2(z_1, X_2)$*
- ...

– *I_k is a closed interval with rational endpoints or a rational containing the root z_k of $\mathcal{T}_k(z_1, \ldots, z_{k_1}, X_k)$ and no other root of $\mathcal{T}_k(z_1, \ldots, z_{k_1}, X_k)$.*

In order to build a set of relevant isolating parallelepipeds, we are going to use polynomials from the successive cylindrifying families we have computed. To compute the set of rational intervals of the parallelepiped, we will need, just like in the univariate case, to compute signs and check existence of roots, but for polynomials with *algebraic coefficients*, the $P(x, X_i)$ with $P \in C_i(\mathcal{P})$.

There are two arguments allowing us to do so. First, we know how to compute the sign of a univariate polynomial at a root of an other one (see 2.4). Then, we know how to compute this sign, using only the *coefficients* of the first polynomial (see 2.3). It is hence possible to isolate the roots of a $P(x, X_i)$ with $P \in C_i(\mathcal{P})$ using recursive calls to the univariate procedure. In fact these calls will be performed by the two recursively defined functions:

– *Real triangular isolation*: Given a parallelepiped isolating $z \in \mathbb{R}^k$ and a non zero polynomial $P \in \mathbb{R}[X_1, \ldots, X_{k+1}]$, computes a parallelepiped isolating (z, y) for each y root of $P(z, X_{k+1})$.
– *Sign at an isolated point*: Given a parallelepiped isolating $z \in \mathbb{R}^k$ and a non zero polynomial $P \in \mathbb{R}[X_1, \ldots, X_k]$, computes the sign of P at z, and possibly an other (refined) parallelepiped isolating z.

Once these tools are available, it is possible to perform all the computations analogous to the ones in the univariate case, namely:

– *Comparison at an isolated point*: Given a parallelepiped isolating $z \in \mathbb{R}^{k-1}$ and two polynomials $P, Q \in \mathbb{Q}[X_1, \ldots, X_k]$, computes a parallelepiped isolating each (z, y) for y a root of $P(z, X_k)$ (resp. $Q(z, X_k)$), and the sign of $Q(z, y)$ (resp. $P(z, y)$)
– *Triangular sample*: Given a parallelepiped isolating $z \in \mathbb{R}^{k-1}$ and a family of polynomials $\mathcal{P} \in \mathbb{Q}[X_1, \ldots, X_k]$, computes parallelepipeds isolating the roots of the non zero polynomials in $\mathcal{P}(z, X_k)$, an element of $\mathbb{Q}$ smaller and an element greater than all these roots, and all the signs of the elements of $\mathcal{P}(z, X_k)$ at the roots computed.

Then the lifting phase is only a combination of the previous functions : computing a set of sample points for $\mathcal{C}_1$ is only applying the univariate algorithm 2.4. To compute a triangular sample at level n, compute a triangular sample at level $n-1$. For every parallelepiped isolating a root x, compute the list L of non zero polynomials $P(x, X_n) \in \mathcal{C}_n$ using *Sign at an isolated point*. Then run *Triangular sample* for L and x.

4 Example

We try to give an idea of the geometrical behaviour of the method, by unfolding the computations over a small (simple) example. Here we consider the family

with only one polynomial $\mathcal{P} := \{X^2 + Y^2 - 1\} \subset \mathbb{Q}[X, Y]$, whose set of roots is the unit circle in the plane.

First, we compute the successive cylindrifying families. Here there are only two such families, as two variables are involved.

- $\mathcal{C}_2(\mathcal{P}) = \{X^2 + Y^2 - 1\}$
- $\mathcal{C}_1(\mathcal{P}) = \{X^2 - 1\}$ (no relevant truncation, only one subresultant coefficient)

Now let us begin the lifting phase. The sample of points for cells of level 1 will consist in 5 points : the two roots of $X^2 - 1$ and one point in each interval they determine. Let us first suppose that the dichotomy process has allowed to find the exact value of the roots -1 as a single point and isolated 1 by the interval $[\frac{1}{2}, \frac{3}{2}]$. This is not realistic, considering the symmetry of the problem, but this will enable us to study both configurations which can be encountered.

Then the triangular sample at level 1, labelled with the sign of the polynomial in $C_1(\mathcal{P})$ can be:

$$(\{-2\}, +),\ (\{-1\}, 0),\ (\{0\}, -),\ ([\ \frac{1}{2}, \frac{3}{2}\ ], 0),\ (\{2\}, +)$$

which can be represented as the following sign table:

	$-2 \in]-\infty, -1[$	-1	$0 \in]-1, 1[$	$[\frac{1}{2}, \frac{3}{4}]$ isolating 1	$2 \in]1, \infty[$
$X^2 - 1$	$+$	0	$-$	0	$+$

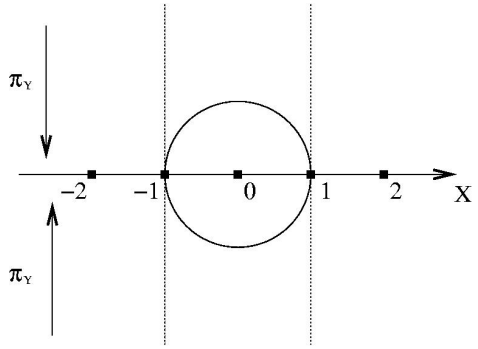

Fig. 2. Cells of level 1 for the unit circle

Nota that these cells of level one are what we have called before parallelepipeds (of level one). Now we have to lift again, to build cells of level 2. We will consider the cell of level 1 one after the other and for each one study the polynomial in $C_2(\mathcal{P})$ on the real line above the sample point.

- Above -2 : $P(-2, Y) = Y^2 + 3$ has a constant positive sign.
  Check that a real line above any point of this cell will never meet the circle.
  The half plane $\{(x, y) \mid x < -1, y \in \mathbb{R}\}$ is a single cell.

- The same above 2. The half plane $\{(x, y) \mid x > 1, y \in \mathbb{R}\}$ is a single cell.
- Above -1 : $P(-1, Y) = Y^2$ has a single root 0. There are three cells of level 2 above this cell of level 1 : two half-lines $\{(x, y) \mid x = -1, y < 0\}$ and $\{(x, y) \mid x = -1, y > 0\}$ and one reduced to a single point $\{\ (-1, 0)\ \}$. Check this line meets the circle at a single point.
- The same above 1, replacing -1 by 1, but computations are differents, since we do not know the exact value of the root (which is in fact 1). We have to study $P(z, Y) = Y^2 + (z^2 + 1)$ where z is the only root of $X^2 + 1$. For this we have to compute the coefficients of $P(z, Y)$ in various Bernstein basis, and the count the sign changes. But coefficients of $P(z, Y)$ will be polynomials in X evaluated at z. Computing the sign of such a coefficient will only be computing the sign of a polynomial at the isolated root z of $X^2 - 1$. And this is possible using the univariate algorithm. Here computations are even simpler because we already know that $z^2 - 1$ is 0, by its definition.
- Above 0 : $P(0, Y) = Y^2 - 1$, just like in the case of the cell $\{-1\}$ we will find five cells of level 2 : $\{(x, y) \mid y < \sqrt{1 - x^2}\}$, $\{(x, y) \mid y > \sqrt{1 - x^2}\}$, $\{(x, y) \mid y = \sqrt{1 - x^2}\}$, $\{(x, y) \mid y = -\sqrt{1 - x^2}\}$,

These 13 cells are described on figure 3.

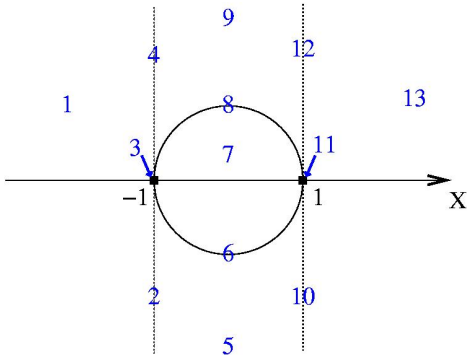

Fig. 3. Cells of level 2 for the unit cercle

5 Formalization and efficiency

Programming such an algorithm demands a special care in the implementation of the basic objects and operations : this is common sense. Anyway this is all the more true when we are programming inside a system like the Coq system, in which computations are made using the reduction machine of the system, without possibility of using machine integers (we will discuss this point in 5.2). In this section, we present our choices for representations of the basic objects manipulated by the algorithm : polynomials and rational numbers.

5.1 Implementation of multivariate polynomials

The set $D[X_1, \ldots, X_n]$ of multivariate polynomials with coefficients in a set D will be built recursively from the construction of univariate polynomials, thanks to the isomorphism $D[X_1, \ldots, X_n] \equiv D[X_1, \ldots, X_{n-1}][X_n]$.

Univariate polynomials over a set of coefficient D a represented in *sparse Horner form*.

Definition 8 (sparse Horner form). *A polynomial in sparse Horner form is:*

- *either a constant polynomial*
- *or of the form $PX^i + c$, where $c \in D$ is a constant, $i \in \mathbb{N}^*$ is called the power index, and P is a polynomial in sparse Horner form.*

This inductive definition can be declared in the Coq system by the following instruction:

```
 Inductive Pol1 (D:Set) : Set :=
| Pc : D -> Pol1 D                                  (* Constant *)
| PX : Pol1 D -> positive -> D -> Pol1 D.       (* Non constant *)
```

Then the set of polynomials with n (ordered) variables is represented by a dependent type (depending on n the number of variables), which is the application of `Pol1` to the set of polynomials with n variables.

Remark 5. The power index intends to avoid the holes in the representation of polynomial introduced by the jumps in degree in the ordered lists of its monomials.

Yet another kind of holes arise with the recursive representation of multivariate polynomials:holes in variables. For example the representation of X_1 as an element of $D[X_1] \ldots [X_{100}]$ will be term of depth 100, first nesting 99 constructors of constant polynomials `Pc`, then building `PX 1 1 0`. This can be solved by dropping the idea of a type depending on the number of variables and adding an extra constructor which allows to inject of polynomial in $D[X_1 \ldots X_n]$ in any $D[X_1 \ldots X_m]$ with $n < m$.

This would lead to the following inductive definition:

```
 Inductive Pol (D:Set) : Set :=
| Pc : D -> Pol D                                    (* Constant *)
| Pinj : Pol D -> positive -> Pol D (* Injection of non constant *)
| PX : Pol D -> positive -> Pol D -> Pol D.      (* Non constant *)
```

Anyway, this representation, which was really a keystone in [15] because of its compactness, turns out not to be so convenient in our case. We have here lost the indication of the number of variables of a polynomial on its type : it has to be computed. Since the algorithm described above is recursive in the number of variables of the polynomials considered and since the descending phase consists in eliminating successively the variables, using this representation would lead to extra computations each time we will have to check that a polynomial is

constant. Moreover this will demand extra lemmas and checks for the correction proofs of the operations.

On the other hand, the complexity of the algorithm, which is doubly exponential in the number of variables, will not allow to deal with a number of variables legitimating this representation.

An element of type `Pol1` is not always in normal form : there can be head zeros and there can also be several choices for the value of the power index. Anyway, the most compact representation of a polynomial, deleting all the irrelevant leading zeros and always choosing the biggest possible value for the power index is such a normal form.

Instead of using a normalisation function, which would introduce an extra cost to all the operations, we will always consider that arguments of the operations are always in normal form and program operations which maintain this invariant.

For this purpose we will use a function `mkPX`, which can be seen as an intelligent constructor, and which builds a polynomial $PX^i + c$ in normal form when given in argument P a polynomial in normal form, i a power index and c a constant. This intelligent constructor will be used to perform each time it is needed cheap local normalisations.

In our formalisation of polynomials, we require an *integral domain* as a set of coefficients. An integral domain is a ring equipped with a partial binary division operation. Then it is possible to program the operations needed to equip the set of univariate polynomials over an integral domain with the structure of an integral domain. We also implement the basic tools required by the algorithm : evaluation, derivative, subresultant polynomials, greatest common divisor,...

5.2 Computing with rational numbers

We have chosen to abstract the representation of rational numbers for this development. We can program the extended signature needed for such an implementation of rationals and let the user choose the one he wants to compute with. This allows to test the efficiency of the computations according to the choices made for the operations over rational numbers.

For example if we chose to represent rational numbers as couples of $\mathbb{Z} \times \mathbb{N}^*$, shall we always normalise the fractions (this would lead to numerous gcd computations) or wait until the bitsizes of the numerator or denominators reach a certain size?

Rational numbers as fractions are implemented in Coq using binary integers. This means that computations with the reduction machine of Coq are made using basis 2. This is far from the efficiency provided by the use of machine integers, like it is the case in computer algebra systems. The Coq system does not provide machine integers, anyway it should be possible to implement them within the system and to extract this implementation to the `BingInt`s of Objective Caml [16], using Coq's extraction feature. This is an ongoing work by Benjamin Grégoire and Laurent Théry.

Another implementation which could turn out to be efficient in the case of this decision procedure would be rational numbers with lazy computations. In fact, each time we evaluate polynomials at some rational numbers, we are never interested in the exact value of the result be only in its sign. This should be the typical field of application of a lazy arithmetic of rational numbers, like the Stern-Brocot implemented in Coq by Milad Niqui (see [17]).

At the present time, we have tested the two first possibilities, and the partial normalisation of fraction is the most efficient. We have also extracted our development with the implementation of machine integers. Still this development it not yet stabilised.

We have not yet been able to test the lazy arithmetic described in [17] because one primitive function remains to be implemented for the rational signature to be completed but we hope we will be able to test it soon.

6 Conclusion

At the present time, we have completely implemented the univariate algorithm, as well as the algorithm of computation of subresultant polynomials in the Coq system. Since subresultants are the most intricate part of the multivariate case, finishing the implementation should not raise further significant difficulty. We have always tried to program in an efficient way, in order to be able to test significant examples, but without doing too much tricky optimisations before having proved the correction of procedures. Due to the many tools needed for this procedure, this already provides a Coq toy- computer algebra systems for symbolic computations over polynomials

This implementation is modular, parametrised by the implementation of rational numbers. This has been done on purpose to be able to test various possibilities and to rely on powerful available rational arithmetics by extraction of the program.

This parametrisation, as well as the recursive structure of the algorithm is implemented with Coq records. The module system of Coq does not allow to build recursive modules, but this can be mimicked with records and fixpoints.

Now we are ready to start the most important part of the work : building the proofs of correction of the algorithms used.

References

1. Barras, B., Boutin, S., Cornes, C., Courant, J., Filliâtre, J.C., Giménez, E., Herbelin, H., Huet, G., Muñoz, C., Murthy, C., Parent, C., Paulin, C., Saïbi, A., Werner, B.: The Coq Proof Assistant Reference Manual Version 8.0. Institut National de Recherche en Informatique et en Automatique. (2004) http://coq.inria.fr/.
2. Grégoire, B., Leroy, X.: A compiled implementation of strong reduction. In: International Conference on Functional Programming 2002, ACM Press (2002) 235–246
3. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. The RAND Corporation, Santa Monica. U.S. Air Force Project RAND, R-109. (1948)

4. G.E.Collins: Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. LNCS **33** (1975) 134–183
5. Caviness, B., Johnson, J., eds.: Quantifier Elimination and Cylindrical Algebraic Decomposition. Texts and monographs in symbolic computation. Springer (1998)
6. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. Journal of Symbolic Computation **12** (1991) 299–328
7. Weisstein, E.W., et al: Cylindrical algebraic decomposition (2004) From MathWorld–A Wolfram Web Resource.
8. Dolzmann, A., Sturn, T.: Redlog: computer algebra meets computer logic. SIGSAM Bull. **31** (1997)
9. Bronstein, M., et al.: AXIOM - The Scientific Computation System. The Axiom Team. (2004) Homepage:http://axiom.axiom-developer.org/axiom-website/community.html.
10. C.W.Brown: QEPCAD. (2004) http://www.cs.usna.edu/ qepcad/B/QEPCAD.html.
11. Harrison, J.: The hol light system (2000) Homepage:http://www.cl.cam.ac.uk/users/jrh/hol-light/.
12. Mahboubi, A., Pottier, L.: Elimination des quantificateurs sur les réels en coq. In: Journées franc̃aises des Languages Applicatifs. (2002)
13. Basu, S., Pollack, R., Roy, M.: Algorithms in real algebraic geometry. Volume 10 of Algorithms and Computation in Mathematics. Springer Verlag (2003)
14. von zur Gathen, J., T.Lcking: Subresultants revisited. Theoretical Computer Science (2003) 199–239
15. Grégoire, B., Mahboubi, A.: Proving equalities in a commutative ring done right in coq. (submitted) (2005)
16. Leroy, X., Doligez, D., Garrigue, J., Rémy, D., Vouillon, J.: The Objective Caml system,release 3.08. Institut National de Recherche en Informatique et en Automatique. (2004) http://caml.inria.fr/.
17. Niqui, M., Bertot, Y.: Qarith: Coq formalisation of lazy rational arithmetic. In: TYPES. (2003) 309–323