AAECC 3, 79–98 (1992)

AAECC
Applicable Algebra in
Engineering, Communication
and Computing
© Springer-Verlag 1992

NC Algorithms for Real Algebraic Numbers

F. Cucker[1,*], **H. Lanneau**[2], **B. Mishra**[3,**], **P. Pedersen**[3,***] **and M.-F. Roy**[2]

[1] Department L.S.I. F. d'Informàtica S-08028 Barcelona, Spain
[2] I.R.M.A.R. U. de Rennes I, F-35042 Rennes France
[3] NYU/Robotics New York University, New York 10003, USA

Received February 28, 1991; revised version August 30, 1991

Abstract. Characterizing real algebraic numbers by a sign-sequence (according to Thom's lemma), using a variant of Ben Or Kozan and Reif algorithm for reducing the solving of systems of polynomial inequalities to the problem of counting real zeroes satisfying one polynomial inequality, and using a multivariate Sturm theory generalizing Hermite quadratic forms method for counting real zeroes, we prove that the computations on real algebraic numbers are in NC.

Keywords: Real algebraic numbers, Parallel algebraic complexity, NC complexity, Thom's lemma, Systems of polynomial inequalities, Sturm theory

Introduction

The abstract data type *real algebraic number* is defined as the subset of the real numbers consisting of real roots of rational polynomials, together with the usual arithmetic operations $+, -, *, /$, as well as the order relations $<, =, >$. The object of our paper is to describe parallel algorithms for this abstract data type based on the *sign-sequence* representation of M. Coste and M.-F. Roy [7]. There are major applications of the order relations to the computation of the topology of real algebraic curves [8] and [22], the analytic structure of real algebraic curves [10], and to the decision problem for semi-algebraic sets [16]. The arithmetic operations have applications to computational number theory and computational geometry. Our paper summarizes the current state of knowledge about parallelization of algorithms for real algebraic numbers.

* Partially supported by DGICyT PB 89/0379, UPC PR9014, and the ESPRIT BRA program of the EC under contract no. 3075, project ALCOM
** Partially supported by NSF CCR-90-02819
*** Supported by AT&T Bell Fellowship, and partially supported by NSA MDA904-89-H-2030

We shall apply the *NC* model of parallel computation. The class *NC* was first described by Nick Pippenger in [21] and it can be roughly defined as the class of sets accepted by uniform families of boolean circuits having polylogarithmic depth and polynomial size (in the size of the input). To be more specific, a set S belongs to NC^k if there is a Turing machine that, given n, generates within space $O(\log(n))$ a circuit C_n with depth $O(\log(n)^k)$ and with n input gates such that, for every word x of size n, C_n returns 1 on x if and only if $x \in S$. The class *NC* is the union of the NC^k for $k \geqq 1$. A reference for the main properties of *NC* is [1]. Since our main concern in this paper derives from algebraic problems, we shall describe our algorithms in terms of arithmetic circuits *i.e.* circuits that have arithmetic nodes instead of Boolean ones (a survey of parallel arithmetic computations can be found in [13]). In a last step, we will replace the arithmetic nodes by Boolean circuits.

Algorithms using the sign-sequence representation reduce to the problem of solving systems of simultaneous polynomial equalities and inequalities. For solving such systems we have algorithms available that are based on the work of Ben-Or, Kozen, and Reif [2] (*BKR* for short). The *BKR* algorithm was applied to the problem of deciding order relations between real algebraic numbers in the original paper of M. Coste and M.-F. Roy [7] where the sign-sequence representation was first introduced. These algorithms have been improved in [23] and implemented within MAPLE and REDUCE in a package called IF (from the French *Inégalités Formelles*); details concerning this package can be found in [9]. *NC* algorithms for the arithmetic operations were given in [18] using the *BKR* algorithm recursively in the bivariate setting.

The present paper uses multivariate Sturm theory [19] to describe a non-recursive *BKR* algorithm in the multivariate setting. This puts all the algorithms into *NC* for any fixed number n of real algebraic numbers. The order relations compare 2 inputs, so they are always in *NC*. For general n, the arithmetic operations have low-depth parallel circuits whose size is singly exponential in n. Such a complexity bound is unavoidable in an algebraic model, since the polynomial defining the sum, product, etc., of n real algebraic numbers may have degree $O(d^n)$, where d bounds the degrees of the polynomials determining the inputs.

In the first section of this paper we describe the sign-sequence data structure. We follow with the design of algorithms for the arithmetic and relational operators in terms of a black box called "consistent sign patterns". The third section describes an algorithm for this black box using a *BKR*-type algorithm in terms of a second level black box, namely the Sturm query, and highlights the combinatorial aspect of the problem. The fourth section describes algorithms for Sturm queries, including a non-recursive multivariate version. We conclude with a complexity analysis.

1 The Sign-Sequence Data Structure

Here we describe an encoding of real algebraic numbers which is well adapted to *NC* computations. The basic idea is to encode such a number in terms of a polynomial which has it as a root, and to attach the sequence of signs which all the derivatives of that polynomial achieve at that root. This sequence turns out to identify the root uniquely among the other roots of the polynomial. Such a representation is non-canonical since other polynomials could also have been used, for this reason we provide efficient equality comparisons.

We begin recalling that the subfield $A \subset \mathbb{R}$ of *real algebraic numbers* consists of all the real roots α of rational polynomials $p \in \mathbb{Q}[x]$. Throughout the following paragraphs we shall speak of "sign-sequences", by which we mean vectors whose components are drawn from the set $\{-1, 0, +1\}$. We also refer to these as "sign-patterns" when we consider them to be drawn from the set of relations $\{<0, =0, >0\}$.

Definition 1.1. *The "sign-representation" of the real algebraic number α which is a root of the polynomial $p \in \mathbb{Q}[x]$, $\deg(p) = d$, is*

$$\langle \alpha \rangle = (p(x), [\operatorname{sgn}(p(\alpha)), \operatorname{sgn}(p'(\alpha)), \ldots, \operatorname{sgn}(p^{(d)}(\alpha))]).$$

The first and last components of the vector $[\operatorname{sgn}(p(\alpha)), \ldots, \operatorname{sgn}(p^{(d)}(\alpha))]$ of course carry no information and could be deleted. They shall be, in this paper, when this leads to simpler formulæ. We retain them in the definition above to simplify the notation. This vector uniquely identifies α among the roots of $p(z)$ by virtue of the following theorem.

Theorem 1.2. (Coste and Roy) *Given α, α' two real-roots of the polynomial $p(x) \in \mathbb{R}[x]$, then $\langle \alpha \rangle = \langle \alpha' \rangle$ implies $\alpha = \alpha'$.*

Proof. The proof is very easy by induction on the degree of p (see [7], or for further details [18]). ■

We denote the vector $[p(x), \ldots, p^{(d)}(x)] \in \mathbb{Q}[x]^{d+1}$ by $\mathscr{T}(p)$ for "Thom sequence", because Theorem 1.2 is a special case of Thom's lemma (see [4]). Rational numbers $r \in \mathbb{Q}$ are represented as $(x - r, [\,])$. The sign representation may be stored using $O(dL)$ bits, where L bounds the bit lengths of the coefficients of $p(x)$. As mentioned in the first paragraph, the sign-sequence representation is not canonical, since a given real algebraic number is the root of many different polynomials. We bypass this problem by providing efficient parallel algorithms for equality comparisons. The sign-sequence representation does not rely on any approximations to the roots and therefore it also applies in the non-Archimedean case, a fact which is used in [8] and [16].

Coste and Roy showed more than theorem 1.2. They showed that if α_1, α_2 are two distinct roots of $p(x)$, then their order relation may be deduced directly from the Thom sequences.

Corollary 1.3. *Suppose*

$$\langle \alpha_1 \rangle = (p(x), [\varepsilon_0, \ldots, \varepsilon_d])$$
$$\langle \alpha_2 \rangle = (p(x), [\delta_0, \ldots, \delta_d])$$

and let k be the smallest number such that $\varepsilon_{d-k} \neq \delta_{d-k}$. Then $k \geqq 1$ and $\varepsilon_{d-k+1} = \delta_{d-k+1}$ is different from 0. There are two cases:

a) *If $\varepsilon_{d-k+1} > 0$, then $\alpha_1 > \alpha_2$ if and only if $p^{(d-k)}(\alpha_1) > p^{(d-k)}(\alpha_2)$.*
b) *If $\varepsilon_{d-k+1} < 0$, then $\alpha_1 > \alpha_2$ if and only if $p^{(d-k)}(\alpha_1) < p^{(d-k)}(\alpha_2)$.* ■

This corollary permits the roots of $p(x)$ to be sorted in *NC*, although it does not immediately solve the problem of ordering two roots of different polynomials.

2 Algorithms for Real Algebraic Numbers

Our approach is "top-down". In order to calculate with the sign-sequence representation, we shall need a black box called "consistent sign patterns". With this black box and the preceding encoding of real algebraic numbers, it is possible to implement the arithmetic of real algebraic numbers.

2.1. The Black Box CSP (Consistent Sign Patterns)

A *system* of polynomials $\bar{p} = (p_1, \ldots, p_k)$ in n variables is a finite list of polynomials whose zero set is a zero-dimensional variety (that is, a finite number of complex points). We denote by $Z_R(\bar{p})$ all the real zeros of $\bar{p}$, and by r the number $\# Z_R(\bar{p})$. We refer to vectors whose components are selected from $\{-1, 0, +1\}$ as *sign-sequences.* Using these notations we can give a specification for *CSP.*

CSP Black box

Input: a system $\bar{p}$, with $p_i(\bar{x}) \in \mathbb{Q}[\bar{x}]$, $i = 1, \ldots, k$ and a list $\bar{q} = [q_1(\bar{x}), \ldots, q_s(\bar{x})] \subset \mathbb{Q}[\bar{x}]^s$, where $\bar{x} = (x_1, \ldots, x_n)$.
Output: the sign-sequences achieved by $\bar{q}$ at the solutions $\alpha \in Z_R(\bar{p})$.

The output then consists of "*C*onsistent *S*ign *P*atterns" of the q_j's at the real zeros of the p_i's, *i.e.* all the vectors $[\operatorname{sgn}(q_1(\alpha)), \ldots, \operatorname{sgn}(q_s(\alpha))]$, for $\alpha \in Z_R(\bar{p})$.

Since our real algebraic numbers are defined by univariate polynomials, we shall use this black box for the particular case of a system of the form $p_i(x_i)$, $i = 1, \ldots, n$.

Given the *CSP* black box, it is possible to find the "sign-representation" of the real roots of this particular system by taking

$$CSP([p_1(x_1), \ldots, p_n(x_n)]; [\mathcal{T}(p_1(x_1)), \ldots, \mathcal{T}(p_n(x_n))]).$$

Given the black box *CSP* we can readily design algorithms for the arithmetic of real algebraic numbers according to the following pattern. We find the sign-representation for the sum, product, etc. of n real algebraic numbers by first finding a polynomial which has the sum, product, etc. as a root. The second step is to identify the particular root corresponding to the inputs among all the other roots of the constructed polynomial. The algorithms exhibited in this section appear in [19]; they are non-recursive, multivariate improvements of the bivariate algorithms used in [18].

Lemma 2.2. *Suppose $\alpha_i, i = 1, \ldots, n$ are n real algebraic numbers, with $p_i(\alpha_i) = 0$. Then the sum and product of the α_i are roots of the following rational polynomials, respectively*:

$$S(x) = \prod_{\{p_1(\beta_1) = \cdots = p_{n-1}(\beta_{n-1}) = 0\}} p_n(x - (\beta_1 + \cdots + \beta_{n-1}))$$

$$P(x) = \prod_{\{p_1(\beta_1) = \cdots = p_{n-1}(\beta_{n-1}) = 0\}} p_n(x/(\beta_1 \cdot \cdots \cdot \beta_{n-1}))$$

Proof. $S(x) = 0$ if and only if some conjugates $\beta_1, \ldots, \beta_{n-1}$ make $x - (\beta_1 + \cdots + \beta_{n-1})$ a root β_n of p_n, (i.e.), if and only if $x = \beta_1 + \cdots + \beta_{n-1} + \beta_n$. This shows that $S(x)$ has as roots all the sums of conjugates of roots of the p_i; certainly the sum of any particular set of conjugates is among them. The argument for $P(x)$ is similar. The two polynomials are rational because they are symmetrized over the rational system $p_1(x_1) = \cdots = p_{n-1}(x_{n-1}) = 0$. We recognize $S(x)$ and $P(x)$ as Poisson products which may be computed via multivariate resultants (see [19]). ■

2.3. *Sign-Representation for* $\sum_i \alpha_i$

We suppose we are given the sign-representations $\langle \alpha_i \rangle$ for real algebraic numbers α_i with $p_i(\alpha_i) = 0, i = 1, \ldots, n$ and we want to compute the sign representation for $\sum_i \alpha_i$, as a root of S.

To do so, first compute

$$CSP([p_1(x_1), \ldots, p_n(x_n)]; [\mathscr{T}(p_1(x_1)), \ldots, \mathscr{T}(p_n(x_n)), \mathscr{T}(S(x_1 + \cdots + x_n))]).$$

This computation outputs the sign-vectors achieved by the sequence

$$[\mathscr{T}(p_1(x_1)), \ldots, \mathscr{T}(p_n(x_n)), \mathscr{T}(S(x_1 + \cdots + x_n))]$$

at all real zeros $(\alpha_1, \ldots, \alpha_n) \in Z_R(\bar{p})$. The sign-pattern produced by $\mathscr{T}(S)$ at the particular root $\sum_i \alpha_i$ will appear among the trailing strings of signs. It may be identified by doing a prefix search among the sign-patterns of $\mathscr{T}(p_1), \ldots, \mathscr{T}(p_n)$ for the sign-pattern corresponding to the inputs $\alpha_i, i = 1, \ldots, n$.

2.4. *Sign-Representation for* $\prod_i \alpha_i$

We suppose we are given the sign-representations $\langle \alpha_i \rangle$ for real algebraic numbers α_i with $p_i(\alpha_i) = 0, i = 1, \ldots, n$ and we want to compute the sign representation for $\prod_i \alpha_i$, as a root of P.

To do so, first compute

$$CSP([p_1(x_1), \ldots, p_n(x_n)]; [\mathscr{T}(p_1(x_1), \ldots, \mathscr{T}(p_n(x_n)), \mathscr{T}(P(x_1, \ldots, x_n))]).$$

The argument here is the same as for $\sum_i \alpha_i$. We obtain all the sign-vectors achieved by

$$[\mathscr{T}(p_1(x_1)), \ldots, \mathscr{T}(p_n(x_n)), \mathscr{T}(P(x_1 \cdots x_n))]$$

at the zeros $(\alpha_1, \ldots, \alpha_n) \in Z_R(\bar{p})$. The sign-pattern achieved by $\mathscr{T}(P)$ at the particular root $\prod_i \alpha_i$ of $P(x)$ will appear among the trailing strings of signs as before, and it may be identified by doing a prefix search among the sign-patterns of $\mathscr{T}(p_1), \ldots, \mathscr{T}(p_n)$.

2.5. *Sign-Representation for* $-\alpha$

We suppose we are given the sign-representations $\langle \alpha \rangle$ for a real algebraic number α, and we want to compute the sign representation for $-\alpha$. If $\langle \alpha \rangle = (p(x), [s_1, s_2, \ldots, s_{n-1}])$ then $\langle -\alpha \rangle = (p(-x), [s_1, -s_2, \ldots, (-1)^n s_{n-1}])$.

2.6. *Determining* $\alpha <, =, > \beta$

We suppose we are given the sign-representations $\langle\alpha\rangle$ and $\langle\beta\rangle$ for real algebraic numbers α root of p and β root of q, and we want to decide the order relation between α and β.

Again, we first compute

$$CSP([p(x), q(y)]; [\mathscr{T}(p(x)), \mathscr{T}(q(y)), (x-y)]).$$

For each solution (α, β) of $p(x) = q(y) = 0$, this prefixes the sign of $(\alpha - \beta)$ with $\langle\alpha\rangle\langle\beta\rangle$. The sign of $\alpha - \beta$ is sufficient to decide $\alpha > \beta, \alpha = \beta$, or $\alpha < \beta$.

2.7. *Sign-representation for* $1/\alpha$

We suppose we are given the sign-representations $\langle\alpha\rangle$ for a real algebraic number α with $p(\alpha) = 0$, and we want to compute the sign representation for $1/\alpha$.

First determine the sign of α by calling $CSP([p(x)]; [\mathscr{T}(p(x)), x])$. If $\alpha = 0$, then stop. Otherwise, if $(d = \deg(p))$ the reversed polynomial $q(x) = x^d p(1/x)$ has the inverses $1/\alpha$ as roots. Now, we multiply each component of $\mathscr{T}(f)(1/x)$ by a sufficiently high *even* power of x so that it becomes a polynomial without changing its sign. Call the resulting sequence $\rho(x)$. Next compute $CSP([q(x)]; [\rho(x), \mathscr{T}(q)])$. This prefixes the sign-codes at the roots $1/\alpha$ of q with the signs acquired by $\mathscr{T}(p)$ at α, so again we may locate the sign-code for any particular $1/\alpha$ among all the conjugate sign-codes.

2.8. *Sign-Representations for* $a + \alpha$ *and* $a \cdot \alpha$

We suppose we are given the sign-representations $\langle\alpha\rangle$ for a real algebraic number α with $p(\alpha) = 0$, and we want to compute the sign representation for $a + \alpha$ and $a \cdot \alpha$ where $a \in \mathbb{Q}$. The polynomial $p(x - a)$ has $a + \alpha$ as a root, and the signs of its derivatives at $a + \alpha$ are identical to those of $p(x)$ at α. The polynomial $q(x) = a^d p(x/a)$ has $a \cdot \alpha$ as a root. In this case

$$\mathscr{T}(q)(x) = [a^d p(x/a), a^{d-1} p'(x/a), \ldots, a p^{(d-1)}(x/a), p^{(d)}(x/a)],$$

and the signs depend on $\langle\alpha\rangle$, the sign of a, and the parity of the exponents (in the case that a is negative).

3 Designing *CSP* with Generalized Sturm Theory

In this section we turn our attention to the question of implementing the black box *CSP* in terms of a second level black box, the Sturm query. Our algorithm at this level is a *BKR*-type poly-log depth combining tree which takes information about how many real zeros of the system V satisfy various constraints $\bigwedge_{j \in J} q_j s_j$, where s_j is one of $\{<0, =0, >0\}$, and composes this information to count how many zeros satisfy all possible conjuctions of constraints

$$\bigwedge_{j \in J} q_j s_j \wedge \bigwedge_{k \in K} q_k s_k.$$

At this level we consider the "combinatorial complexity" of our algorithms, in terms of the number of real roots r of the system and the total number s of polynomials q_j, assuming unit cost for Sturm queries. In next section we will enter inside the Sturm queries and see exactly what they cost.

3.1. The Sturm Query Black Box

Let $\bar{p}$ be a system, and q be a polynomial. Let

$$c(\bar{p}, q > 0) = \#\{x \in Z_R(\bar{p}) | q(x) > 0\}$$
$$c(\bar{p}, q < 0) = \#\{x \in Z_R(\bar{p}) | q(x) < 0\}$$
$$c(\bar{p}, q = 0) = \#\{x \in Z_R(\bar{p}) | q(x) = 0\}.$$

We shall let $SQ(\bar{p}, q) = c(\bar{p}, q > 0) - c(\bar{p}, q < 0)$. We refer to it as a *Sturm query*.
The specification for the SQ blackbox is then the following.

Sturm query black box
Input: a system $\bar{p} = (p_1, \dots, p_n)$ and a polynomial q
Output: $SQ(\bar{p}, q) = c(\bar{p}, q > 0) - c(\bar{p}, q < 0)$

Now we shall explain how, given this subroutine, one can design an algorithm for *CSP*.

Definition 3.2. *We shall denote by B_k the set of bit-vectors of length k, $\{0, 1\}^k$, and by S_k the set of sign-patterns of length k, $\{< 0, = 0, > 0\}^k$.*

The algorithm that follows depend upon two simple but critical observations.
Points $x, y \in S_k$ may be used to represent sign-patterns achieved by polynomials q_1 and q_2 at some subset finite $[\alpha_1, \dots, \alpha_k]$, and componentwise products xy in this case correspond to sign-patterns of the product polynomial $q_1 q_2$.

On the other hand, points in B_k may be used to represent subsets of the above mentioned $[\alpha_1, \dots, \alpha_k]$ since they can be viewed as characteristic functions for those subsets. If $\varphi, \psi \in B_k$, then the componentwise product $\varphi \cdot \psi$ corresponds to the intersection of the subsets they represent, and if those subsets are *disjoint*, then $\varphi + \psi$ represents their union.

In particular, a point in B_k may represent the subset of the real zeros of $\bar{p}$ where a given vector of polynomials (like the Thom sequence) achieves a particular sign-pattern. In order to better describe this situation we introduce the following notation.

Definition 3.3. *Given $\bar{\alpha} = [\alpha_1, \dots, \alpha_m]$, $\bar{q} = [q_1, \dots, q_m] \in \mathbb{Q}[\bar{x}]^m$, and $\sigma \in S_m$, define the "sign-condition"*

$$C(\bar{\alpha}, \bar{q}\sigma) = [v(\forall i\, q_i(\alpha_1)\sigma_i), \dots, v(\forall i\, q_i(\alpha_m)\sigma_i)]$$

where $v(p) = 1$ if p is true, $v(p) = 0$ if p is false.

That is to say, $C(\bar{\alpha}, \bar{q}\sigma)$ is the bit-vector representing the subset of points of $\bar{\alpha}$ where $\bar{q}$ adopts the sign-pattern σ.

We also introduce a notation to describe which sign-pattern is achieved by a polynomial at some sequence of real points.

Definition 3.4. *Given $q \in \mathbb{Q}[\bar{x}]$, and a vector $\bar{\alpha} = [\alpha_1, \ldots, \alpha_m]$ of points $\subset \mathbb{R}^n$, define* $\operatorname{sgn}|q, \bar{\alpha}) \in S_m$ *as*

$$\operatorname{sgn}|q, \bar{\alpha}) = [\operatorname{sgn}(q(\alpha_1)), \ldots, \operatorname{sgn}(q(\alpha_m))].$$

With these definitions and identifying $+1$ and >0, -1 and <0, 0 and $=0$, we get the following fundamental lemma.

Lemma 3.5. *Given a finite set $\bar{\alpha}$ and $q \in \mathbb{Q}[x]$, then*

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} C(\bar{\alpha}, q=0) \\ C(\bar{\alpha}, q>0) \\ C(\bar{\alpha}, q<0) \end{bmatrix} = \begin{bmatrix} \operatorname{sgn}|1, \bar{\alpha}) \\ \operatorname{sgn}|q, \bar{\alpha}) \\ \operatorname{sgn}|q^2, \bar{\alpha}) \end{bmatrix}$$

Proof. We first remark that the meaning of the matrix product in the statement is

$$(Ax)_i = \sum_{j=1}^{3} a_{i,j} x_j,$$

where the vectors x_j are added componentwise. Note that the statement of the lemma contains the hidden assertion that the result, *a priori* defined in $(\mathbb{Z}^k)^3$ actually lies in S_k^3.

The bit vectors on the left hand side are defined by mutually exclusive sign-conditions, hence are disjoint, hence their sum corresponds to the union of their associate sets of points. The first row says that the union $C(\bar{\alpha}, q=0) + C(\bar{\alpha}, q>0) + C(\bar{\alpha}, q<0)$ of the subsets of points where q is zero, positive, or negative, respectively (which accounts for all the real zeros) equals the sign vector 1 of all 1's. The second row has an entirely different significance: it restates the definition of $\operatorname{sgn}(q)$. The difference of bit-vectors produces a sign-vector which is positive at the positive points and negative at the negative points. The third row says that the union $C(\bar{\alpha}, q>0) + C(\bar{\alpha}, q<0)$ equals the sign-vector of q^2, which is correct because q^2 is positive when q is either positive or negative. ■

Definition 3.6. *For $x \in B_k$ or $x \in S_k$, let $\mu(x) = \sum_{i=1}^{k} x_i$. Also, for $x \in B_k^n$ or $x \in S_k^n$, let $\mu(x) = [\mu(x_1), \ldots, \mu(x_n)]$.*

Lemma 3.7. *If $x \in B_k^n$, and $A \in \mathbb{Z}^{n \times n}$, then $\mu(Ax) = A\mu(x)$.*

Proof. If x_j is the j-th component of x, let $x_{j,l}$ denote the l-th bit of x_j. Then

$$\mu((Ax)_i) = \sum_{l=1}^{k} \sum_{j=1}^{n} a_{i,j} x_{j,l} = \sum_{j=1}^{n} a_{i,j} \sum_{l=1}^{k} x_{j,l} = (A\mu(x))_i. \quad ■$$

Corollary 3.8. *Given $q \in \mathbb{Q}[x]$, we have the equality*

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} c(\bar{p}, q=0) \\ c(\bar{p}, q>0) \\ c(\bar{p}, q<0) \end{bmatrix} = \begin{bmatrix} SQ(\bar{p}, 1) \\ SQ(\bar{p}, q) \\ SQ(\bar{p}, q^2) \end{bmatrix}$$

Proof. Apply μ to both side of the equation in Lemma 3.5. ■

The results of such queries may be combined using the "BKR tensor identity" [2] as follows

Lemma 3.9. *Suppose $C \in B_k^n$ and $C' \in B_k^m$ specify n and m subsets of $Z_R(\bar{p})$, respectively. Suppose $\sigma \in S_k^n$ and $\sigma' \in S_k^m$ specify sign-patterns achieved on $Z_R(\bar{p})$ by n and m polynomials q_i and q'_j, $i = 1, \ldots, n$, $j = 1, \ldots, m$. Then for any $A \in \mathbb{R}^{n \times n}$, and $A' \in \mathbb{R}^{m \times m}$*

$$AC = \sigma; A'C' = \sigma' \Rightarrow (A \otimes A')(C \otimes C') = \sigma'',$$

where $\sigma'' \in S_k^{nm}$ is the vector of sign-patterns of the mn polynomials $q_i q'_j$.

Proof. This lemma follows from the definition of the tensor product: $(A \otimes B)(v \otimes w) = (Av) \otimes (Bw)$. The point is that the tensor product $C \otimes C'$ on the left hand side has components which represent intersections of sign-conditions, whereas the tensors $\sigma'' = \sigma \otimes \sigma'$ on the right hand side represent the sign-vectors of product polynomials. ■

3.10. *How to Obtain CSP*

We are now ready to describe how to obtain *CSP*. Using a log-depth combining tree, we can compute a single large linear relation between the vector of every possible conjunction of sign-conditions on $q_1, \ldots, q_s$ and the vector of sign-vectors achieved by the polynomials

$$q_v = \prod_{i=1}^{M} q_i^{v_i}$$

(where v_i varies in $\{0, 1, 2\}^m$) at $Z_R(\bar{p})$ (the zeros of the system $p_1(x_1) = \cdots = p_n(x_n) = 0$). At every level of the tree, the accumulating linear relations may be mapped via μ to corresponding linear relations between the cardinalities $c(\bar{p}, \bar{q}_v \sigma)$ and the sign-counts given by Sturm queries for the product polynomials q_v.

The key observation of Ben-Or, Kozen, and Reif was that only sign-conditions with non-zero cardinalities are of interest; those with cardinality zero should be discarded on the way up the tree. This process of removing empty sign-conditions requires deleting those columns i of A (in the relation $Ac = s$) where $c_i = 0$, and then extracting a maximum-rank square sub-matrix of the resulting system. There can never be more than $r = \#Z_R(\bar{p})$ different non-empty sign-conditions, and this limits the size of the arrays which occur.

The pattern of binary combinations among the relations $Ac = s$ evidently produces a tree whose depth is logarithmic in s, the number of polynomials whose consistency is being checked. The circuit size is polynomial in s by virtue of the fact that the total number of nodes in any binary tree with s leaves is less than of equal to $s + s/2 + s/4 + \cdots + 1 = 2s - 1$.

Without this reduction the subcircuits solving *CSP* would become exponential in the input parameter s, trebling in size at each level going up the tree. As it is, the matrices can never exceed $r = \#Z_R(\bar{p})$ in dimension, since that is the maximum number of distinct non-empty sign-patterns which can be achieved: one per distinct real zero. In the section four where we discuss implementations of Sturm queries we shall see that this makes it possible to keep the Sturm queries subcircuits in *NC*. Each combining step involves the tensor product of two linear systems each of size bounded by r, therefore of total size bounded by r^2. We refer to this as parallel *BKR*.

When the number of processors is small, it may be preferable to combine the sign-patterns using an unbalanced binary tree which looks like a ladder: adding only one new constraint at each level. This produces a size bound of $3 \cdot r$ for the matrices, although it sacrifices the logarithmic depth of the combining circuit. Circuit size remains polynomial. We shall denote this approach by vectorial *BKR*.

Let us now enter into further details about parallel *BKR* and vectorial *BKR*, as well as about improvements in the case of coding real algebraic numbers.

3.11. Optimizations and Algorithms for CSP

We shall consider practical optimizations of the algorithms presented above and in [7], which includes the calculation of order relations between arbitrary real algebraic numbers as well as the determination of the sign of $\bar{q}(\bar{x}) \in \mathbb{Q}[\bar{x}]^k$ at the real roots of $\bar{p}(\bar{x}) \in \mathbb{Q}[\bar{x})^n$. In particular, a detailed analysis of the algorithm allows to prove that at each node of the *BKR* tree, the polynomials to consider are products of at most $\log(r)$ polynomials in $\bar{q}$. This fact will not modify the depth of the circuits computing *CSP* but will reduce their size.

The following data structure contains the information at one node of a *BKR* combining tree; it is essentially a "frame" which allows the computation to be resumed at that node.

Definition 3.11.1. *We shall say that T is a Tarski Type (TDT in the sequel), if T is a list of the form $[\bar{p}, \bar{q}, \sigma, c, A, Q, s]$, where:*

- *$\bar{p}$ is a system of polynomials*
- *$\bar{q}$ is a list of k polynomials*
- *σ is the list of the non-empty sign conditions realized by $\bar{q}$ at the real zeros of $\bar{p}$*
- *c is the corresponding vector of cardinalities of the sets of zeros verifying the non-empty sign conditions*
- *$A \in \mathbb{Z}^{l \times l}$ is an $l \times l$ matrix, where l is the number of non-empty sign conditions*
- *Q is a list of l polynomials*
- *$s \in S_k^l$ is an l-vector of integer numbers satisfying*

  i) $A \cdot c = s$
  ii) $s_i = SQ(\bar{p}, Q_i)$.

That is to say, the various parts of T stand in the relation generated by the *BKR* tensor identity as one combines sign-consistency conditions going up the tree. By retaining this partial computation we avoid having to recompute. We shall call $\bar{p}$ the *system* of T, $q_1, \ldots, q_k$ the *list of polynomials* of T, l the *dimension* of T, and k its *size*.

In the following we describe in detail the modules with the *BKR* algorithm, and we include a number of practical optimizations which use the Tarski Data Types. The second one, which we call *VCSP* (vectorial consistent sign patterns) is not in *NC* but is useful when few processors are available since it has an optimal speed-up, i.e. parallel time is close to sequential time divided by number of processors.

3.11.2. Adding the Inequalities "by Packages". The first procedure explains the combination of two nodes in the *BKR* tree.

Procedure *PCSPC* (parallel consistent sign pattern combination)

Input: two TDT's T_1 and T_2 with the same system $\bar{p}$, and list of polynomials $q_1, \dots, q_{k_1}$ and $q'_1, \dots, q'_{k_2}$ respectively.
Output: a TDT T with system $\bar{p}$ and list of polynomials $q_1, \dots, q_{k_1}, q'_1, \dots, q'_{k_2}$.
Let T_1 and T_2 be given respectively by

$$[p, \bar{q}_1, \sigma_1, c_1, A_1, Q_1, s_1] \quad \text{and} \quad [p, \bar{q}_2, \sigma_2, c_2, A_2, Q_2, s_2]$$

where $\sigma_1 = [\sigma_{1,1}, \dots, \sigma_{1,k_1}]$ and $\sigma_2 = [\sigma_{2,1}, \dots, \sigma_{2,k_2}]$, and let n_1, n_2, k_1 and k_2 be their dimensions and sizes. The procedure performs the following steps:

1. Compute a new list $\bar{q}$ catenating $\bar{q}_1$ and $\bar{q}_2$, and a new list Q' of $n_1 \cdot n_2$ elements whose $((j-1)n_2 + l)^{\text{th}}$ element (for $1 \leqq j \leqq n_1$ and $1 \leqq l \leqq n_2$) is the product of the j^{th} element of Q_1 by the l^{th} element of Q_2. Then compute the vector s' of dimension $n_1 \cdot n_2$ such that $s_i = SQ(\bar{p}, r_i)$ where r_i is the i^{th} element of Q'. This last computation is performed with $n_1 \cdot n_2$ independent processors, one for each Sturm query.
2. Define a list of $n_1 \cdot n_2$ $(k_1 + k_2)$-tuples of sign conditions whose $((j-1)n_2 + l)^{\text{th}}$ element (for $1 \leqq j \leqq n_1$ and $1 \leqq l \leqq n_2$) is $\sigma_{1,j}$ followed by $\sigma_{2,l}$.
3. Compute the product A' of A_1 and A_2.
4. Compute an $(n_1 \cdot n_2)$-dimensional vector c' satisfying the equation $A' \cdot c' = s'$.
5. Determine a new vector c'' by deleting the zero components of c', a new rectangular matrix A'' by deleting in A' the columns corresponding to these components, and a new list σ of $(k_1 + k_2)$-tuples of sign conditions by deleting the ones non satisfied (i.e. those whose corresponding coordinate in c'' is zero). Let n denote the number of elements in σ.
6. Determine an invertible square submatrix A of A'' by keeping the first n independent rows, a list Q obtained from Q' by keeping the elements corresponding to the rows of A, and a vector s obtained from s' in the same way.
7. Return the TDT $T = [\bar{p}, \bar{q}, \sigma, c, A, Q, s]$.

Procedure *PCSP* (parallel consistent sign patterns)

Input: a system of polynomials $\bar{p}$ and a list of polynomials $q_1, \dots, q_s$.
Output: a TDT with system $\bar{p}$ and list of polynomials $q_1, \dots, q_s$.

The procedure is done in several steps. In step 0 we compute in parallel the values $c(\bar{p}, q_j = 0)$, $c(\bar{p}, q_j > 0)$ and $c(\bar{p}, q_j < 0)$ for $1 \leqq j \leqq s$. We get as output of this step s TDT's. In Step 1 we apply *PCSPC* $\left\lceil \frac{s}{2} \right\rceil$ times to the pairs of TDT's for q_1 and q_2, q_3 and $q_4, \dots q_{k-1}$ and q_k obtaining $\left\lceil \frac{s}{2} \right\rceil$ new TDT's.

Following the process in this way, we perform in step j, $\left\lceil \frac{s}{2^j} \right\rceil$ applications of *PCSPC*, combining $\left\lceil \frac{s}{2^{j-1}} \right\rceil$ TDT's into $\left\lceil \frac{s}{2^j} \right\rceil$ new TDT's. We clearly finish this process after $\lceil \log s \rceil$ steps obtaining the desired TDT.

The following proposition will be helpful to sharpen size bounds in complexity estimations.

Proposition 3.11.3. *Let r be the number of elements of $Z_R(\bar{p})$. The polynomials Q_i in the output of PCSP $(\bar{p}; q_1, \dots, q_s)$ are products of at most* $\log r$ polynomials *q_j or q_j^2 $(j = 1, \dots, s)$.*

Proof. The proof is a variation on the one given for the sequential case, see [8]. The main point of the proof consists in proving that if a polynomial $Q = \prod_{i \in I_1} q_i \prod_{i \in I_2} q_i^2$ (with $I_1 \cap I_2 = \emptyset$) appears in the output of *PCSP*, for each subsets J_1 and J_2 of I_1 and I_2, the polynomial $Q' = \prod_{i \in J_1} q_i \prod_{i \in J_2} q_i^2$ appears in the output of *PCSP*. ■

3.11.4. Adding Inequalities "one by one". Procedure *VCSPC* (vectorial consistent sign pattern combination)

Input: a TDT's $T = [p, [q_1, \ldots, q_k], \sigma, c, A, Q, s]$ and a polynomial q_{k+1}.
Output: a TDT $\tilde{T}$ with system $\bar{p}$ and list of polynomials $q_1, \ldots, q_{k+1}$.

Let n be the dimension of T. The procedure performs the following steps:

1. Compute a list Q' of $3n$ elements such that, for $1 \leqq i \leqq n$

   the i^{th} element of Q' is the i^{th} element of Q,
   the $(n+i)^{\text{th}}$ element of Q' is the i^{th} element of Q multiplied by q_{k+1}, and
   the $(2n+i)^{\text{th}}$ element of Q' is the i^{th} element of Q multiplied by q_{k+1}^2.

   Then compute the vector s' of dimension $3n$ such that $s_i = SQ(\bar{p}, r_i)$ where r_i is the i^{th} element of Q'. This last computation is performed with $2n$ independent processors, since we already know the first n coordinates of s'.
2. Now define a list of $3n(k+1)$-tuples of sign conditions whose first n elements are the input k-tuples followed by 0, and whose second and third sets of n elements are the same k-tuples but followed by $+1$ and -1 respectively.
3. Compute the product A' of A with
$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{bmatrix}.$$
   i.e. the matrix
$$A' = \begin{bmatrix} A & A & A \\ 0 & A & -A \\ 0 & A & A \end{bmatrix}$$
   where A is the matrix of T.
4. Compute a $3n$-dimensional vector c' satisfying the equation $A' \cdot c' = s'$.
5. Determine a new vector c'' by deleting the zero components of c', a new rectangular matrix A'' by deleting in A' the columns corresponding to these components, and a new list $\tilde{\sigma}$ of $(k+1)$-tuples of sign conditions by deleting the ones non satisfied. Let $\tilde{n}$ denote the number of elements in $\tilde{\sigma}$.
6. Determine an invertible square submatrix $\tilde{A}$ of A'' by keeping the first $\tilde{n}$ independent rows, a list $\tilde{Q}$ obtained from Q' by keeping the elements corresponding to the rows of $\tilde{A}$, and a vector $\tilde{s}$ obtained from s' in the same way.
7. Return the TDT $[\bar{p}, [q_1, \ldots, q_{k+1}], \tilde{\sigma}, \tilde{c}, \tilde{A}, \tilde{Q}, \tilde{s}]$.

Procedure *VCSP* (vectorial consistent sign patterns)

Input: a sequence of polynomials with integer coefficients $p, q_1, \ldots, q_s$.
Output: a TDT T with system p and remaining polynomials $q_1, \ldots, q_s$.

In particular we get from T the satisfied sign conditions on the elements of $Z_R(\bar{p})$ as well as the number of roots satisfying each one of them. The procedure just performs *VCSPC* k times.

Again, we have the following result.

Proposition 3.11.5. *Let r be the number of real roots of the system $\bar{p}$. The polynomials Q_i in the output of $VCSP\ (\bar{p}; q_1, \ldots, q_s)$ are products of at most* $\log r$ *polynomials q_j or q_j^2.*

Proof. The proof is given in [8]. As in 3.11.3, the main point of the proof consists in proving that if a polynomial $Q = \prod_{i \in I_1} q_i \prod_{i \in I_2} q_i^2$ (with $I_1 \cap I_2 = \emptyset$) appears in the output of *VCSPC*, for each subsets J_1 and J_2 of I_1 and I_2, the polynomial $Q' = \prod_{i \in J_1} q_i \prod_{i \in J_2} q_i^2$ appears in the output of *VCSPC*. ∎

3.11.6. Real Algebraic Numbers. In the particular case we study, where the system $\bar{p}$ consists of n univariate polynomials, we want to describe a procedure *PRAN* for encoding real algebraic numbers.

Procedure *PRAN* (parallel real algebraic numbers)

Input: a system of univariate polynomials $\bar{p} = (p_1, \ldots, p_n)$ of degrees d_i, $i = 1, \ldots, n$.
Output: a TDT with system $\bar{p}$ and remaining polynomials $p_1', p_1'', \ldots, p_1^{(d_1)}, \ldots, p_n', p_n'', \ldots, p_n^{(d_n)}$.

The satisfied sign conditions code the roots of $\bar{p}$ and make it possible to sort them as we have seen. The procedure just applies independently *PCSP* in parallel for each i to p_i and $p_i', p_i'', \ldots, p_i^{(d)}$.

Remark 3.11.7. A better strategy can be followed in the case when few processors are available. It is suggested by the following facts.

1. We are not interested in the signs of all the derivatives on the roots of p_i but just on the signs necessary to "separate" them.
2. Once the roots are discriminated from each other, we just need the signs of the derivatives of lowest degree in order to sort them.
3. If for some roots the k-th derivative vanishes Thom's lemma applies to this derivative, so we need no more sign computations for characterizing this set of roots.

Following this remark, a procedure called *RAN add* is described in [23] which stops the sign computations for the roots once a zero sign is reached or the roots have been separated. It involves a modification of the linear system used in *VCSPC*. Using it, a procedure *VRAN* can be designed which is similar to the algorithm *RAN* in [23] but computes in parallel all Sturm queries each time a derivative is added.

Since the number of elements in every non empty sign condition on the derivatives obtained in the output of *PRAN* is equal to 1, if we are interested on consistent sign patterns for a list of polynomials at real algebraic numbers, we can consider each polynomial independently.

Procedure *PRANCSP* (parallel real algebraic numbers and consistent sign patterns)

Input: the output of *PRAN* for a system $\bar{p}$ of univariate polynomials, as well as a sequence of polynomials $q_1, \ldots, q_s$.

Output: the signs taken by $q_1, \ldots, q_s$ on the real roots of $\bar{p}$.

If the system $\bar{p}$ has r roots the dimension of the input TDT is r and its vector c is $(1, 1, \ldots, 1)$. So, we can independently perform *VCSPC* for this TDT and every q_j, $1 \leqq j \leqq s$, in parallel, to get the desired signs.

4 Algorithms for Sturm Queries

We turn now to the lowest level of our algorithms, in which we construct subcricuits for the Sturm queries. In the first paragraph we recall the univariate methods for answering Sturm queries and in the second we give information on new multivariate methods, and make them explicit in the particular case of a system of univariate polynomials. It turns out that in the two cases, Sturm queries computations involve only linear algebra subroutines. In order to do this with the required parallel complexity, we need circuits for calculating determinants and also for calculating rank (for the size reduction). One of the central results in this subject is Berkowitz's algorithm for computing the characteristic polynomial [3] that we quote in subsection 4.3.

4.1. The Univariate Case

There are two main ways for computing the Sturm query $SQ(p,q) = c(p, q > 0) - c(p, q < 0)$ in the univariate case.

4.1.1. Sturm–Sylvester Sequence. It can be done, for example, by using the classical Sturm–Sylvester sequence in the following way. Let $\mathrm{Stu}(p, p'q)$, for univariate polynomials p, q, be defined as the sequence

$$\begin{aligned} \mathrm{Stu}_0(p, p'q) &= p \\ \mathrm{Stu}_1(p, p'q) &= p'q \\ &\cdots \\ \mathrm{Stu}_{m+1}(p, p'q) &= -\mathrm{Rem}(\mathrm{Stu}_{m-1}(p, p'q), \mathrm{Stu}_m(p, p'q)) \\ &\cdots \end{aligned}$$

where Rem is the remainder in Euclidean division. Then the Sturm query $SQ(p,q)$ is the difference between the sign variations of $\mathrm{Stu}(p, p'q)$ at $-\infty$ and the sign variations of $\mathrm{Stu}(p, p'q)$ at $+\infty$. This is very easy to prove, by a straightforward generalization of the classical proof of Sturm theorem. This result was first observed by Sylvester [24].

Using subresultant theory, where precise relations are described between the remainders in the Euclidean division and subdeterminants extracted from Sylvester matrix, it is possible to compute the Sturm query by linear algebra subroutines, that is, computing determinants and then evaluating their signs. Details about it appear in [14] or [15] where the Sturm–Habicht sequence is introduced and studied.

4.1.2. Hermite's Method

Let p be a monic polynomial of degree d and q a polynomial. Let $(\alpha_i)_{i=1,\ldots,p}$ be the zeros of p in $\mathbb{C}$.

Let us define a quadratic form $B(p,q)$ with d variables $y_0, y_1, \ldots, y_{d-1}$, by

$$B(p,q) = \sum_{i=1}^{d} q(\alpha_i)(y_0 + y_1\alpha_i + \cdots + y_{d-1}\alpha_i^{d-1})^2$$

each root of p being counted with multiplicity, so that the coefficients of the quadratic form are symmetric functions of the α_i.

One has

$$B(p,q) = \sum_{e,f=0}^{d-1} \sum_{i=1}^{d} q(\alpha_i)\alpha_i^{e+f} y_e y_f.$$

Theorem 4.1.3. (Hermite's method) *With the above notations we have*:
i) *the rank of $B(p,q)$ is equal to the number of zeros of p which are not zeros of q in $\mathbb{C}$.*
ii) *the signature of $B(p,q)$* is equal to $SP(p,q) = c(p, q > 0) - c(p, q < 0)$.

Proof. Let $\beta_1, \ldots, \beta_n$ be the distinct real zeros of p, and $m_1, \ldots, m_n$ their multiplicities. Also, let $\gamma_1, \overline{\gamma_1}, \ldots, \gamma_m, \overline{\gamma_m}$ be the complex (non real) distinct zeros of p, and $w_1, \ldots, w_m$ their multiplicities.

For $a \in \mathbb{C}$, let y be the linear form on $\mathbb{C}^n$ defined by

$$y(a,x) = y_0 + y_1 a + \cdots + y_{d-1}a^{d-1}$$

and let $b(a,x) = y(a,x)^2$. The quadratic form $B(p,q)$ is equal to

$$B(p,q) = \sum_{j=1}^{n} m_j q(\beta_j) b(\beta_j, x) + \sum_{h=1}^{m} w_h(q(\gamma_h)b(\gamma_h, x) + q(\overline{\gamma}_h)b(\overline{\gamma}_h, x)).$$

Linear forms $y(\beta_j, x), y(\gamma_h, x), y(\overline{\gamma}_h, x)$ are linearly independent (the zeros are distinct and it is sufficient to consider a van der Monde determinant). This gives (i).

Writing $q(\gamma_h) = d_h^2$ are decomposing $d_h(b(\gamma_h, x)$ under the form $p_h + iq_h$ with p_h et q_h real linear forms, it is clear that $q(\gamma_h)b(\gamma_h, x) + q(\overline{\gamma}_h)b(\overline{\gamma}_h, x)$ is the difference of two squares of real linear forms.

The signature of $B(p,q)$ depends then only on the n first terms of the form and is hence equal to $c_{>0}(p,q) - c_{<0}(p,q)$. ■

Let us now explain how to compute $B(p,q)$. Since polynomials qx^{e+f} can be reduced modulo, p, we need to compute linear combinations of the Newton sums

$$s_k = \sum_{i=1}^{p} q(\alpha_i)\alpha_i^k, k = 0, \ldots, d-1.$$

For computing these Newton sums, let us consider the development in $\frac{1}{X}$ of the rational function $\frac{p'}{p}$. Since $p = \prod_{i=1,\ldots,p} (X - \alpha_i)$ one has

$$\frac{p'}{p} = \sum_{i=1}^{p} \frac{1}{(X - \alpha_i)}.$$

The coefficient of $\frac{1}{X^k}$ in the development of $\frac{p'}{p}$ in $\frac{1}{X}$ is hence with the preceding notations s_{k-1}.

Once the Hermite's quadratic form $B(p, q)$ is known, it is only needed to evaluate its signature. The evaluation of the signature of the Hermite quadratic form can be made using linear algebra subroutines. Once the coefficients of the quadratic form are computed, one computes the characteristic polynomial of the associated symmetric matrix. All the roots of this characteristic polynomial are real (perhaps with multiplicities). Using Descartes's rule of signs, which counts exactly the number of positive (and hence negative) real roots with multiplicities when all the roots are real, a very simple sign evaluation gives the signature, which is the difference between the number of positive eigenvalues (counted with multiplicity) and the number of negative eigenvalues (counted with multiplicities).

So only linear algebra subroutines are required.

Remark 4.1.4. In fact it turns out that it is possible to compute the signature of the Hermite quadratic form through principal minors rather than through characteristic polynomial, since the associated symmetric matrix is a Haenkel matrix. Since it is clear by the definitions that these principal minors are just the principal coefficients of the Sturm–Habicht sequence, it is possible to prove that Sturm method and Hermite method, who look very different at first sight maybe very precisely related through Sturm–Habicht sequences. This can be found in [14] or [15]).

4.2. *The Multivariate Case*

The Sturm–Sylvester sequence does not generalize to the multivariate case. On the other hand, generalizations of Hermite method are possible.

Hermite's method has been already used in some problems in computational algebra, in the univariated setting (see [12] and [17]). In [19] Hermite's method is generalized and an implementation for generalized Sturm queries (for systems defined by a number of equations equal to the number of variables) is given. A new generalization of Hermite's method giving the Sturm query computation in case of a system (given by an arbitrary number of equations) will be given in [20]. In the two cases, the Sturm query is equal to the signature of a quadratic form, generalizing Hermite quadratic form for the univariate case. The coefficients of the quadratic form can be obtained through symmetric functions computations (in [19]) or through trace of multiplications (in [20]). Once the quadratic form is computed, Descartes' rule applied to the characteristic polynomial of the associated symmetric matrix gives the required signature.

Let us explain the computations in [19] in our particular simple case of a system

$$\bar{p} = (p_1(x_1) = \cdots = p_n(x_n))$$

of polynomials with degrees $d_1, \ldots, d_k$.

Let F be the set of sequences $e = (e_1, \ldots, e_n)0 \leq e_k < d_k$ for all k. The cardinality of F is $D = \prod_{i=1,\ldots,n} d_i$. If $\alpha = (\alpha_1, \ldots, \alpha_n)$ is a root of $\bar{p}$, and $e \in F$ we write

$$x^e = x_1^{e_1} x_2^{e_2} \cdots x_n^{e_n}, \quad \alpha^e = \alpha_1^{e_1} \alpha_2^{e_2} \cdots \alpha_n^{e_n}.$$

The Hermite's quadratic form is

$$B(\bar{p}, q) = \sum_{\alpha \in Z(\bar{p})} q(\alpha) \left(\sum_{e \in F} y_e \alpha^e \right)^2 = \sum_{e \in F, f \in F} \sum_{\alpha \in Z(p)} q(\alpha) \alpha^{e+f} y_e y_f$$

(where $Z(\bar{p})$ is the set of complex zeros of $\bar{p}$, and the sum takes into account the multiplicity of the root; if $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$, its multiplicity is the product of the multiplicities of the α_i's as zeros of the p_i's).

The number of variables of this quadratic form is D.

We have again the following theorem:

Theorem 4.2.1. (Hermite's method) *With the above notations*
i) *the rank of $B(p, q)$ is equal to the number of zeros of $\bar{p}$ which are not zeros of q in $\mathbb{C}$.*
ii) *the signature of $B(\bar{p}, q)$ is equal to $c(\bar{p}, q > 0) - c(\bar{p}, q < 0)$.*

Proof. The proof is a straightforward generalization of the proof in the univariate case. ■

It is always possible, by reducing modulo $\bar{p}$ to replace $qx^e x^f$ by a polynomial whose degree in each variable x_k is smaller than d_k. Hence in order to compute $B(\bar{p}, q)t$, one may compute the symmetric functions

$$\sum_{\alpha \in Z(\bar{p})} \alpha_1^{e_1} \alpha_2^{e_2} \cdots \alpha_n^{e_n}.$$

where $e \in F$, $f \in F$ and make linear combinations of them.

These symmetric functions are just products of the Newton sums associated respectively to the roots $p_1, \ldots, p_n$.

The computation of the Newton sums starting from the coefficients of the p_i's has been explained in the previous subsection.

4.3. Parallel Linear Algebra

In order to end our "top-down" approach it is needed to give now parallel implementation to linear algebra subroutines needed in the previous Sturm queries computations. We use the work of [3].

Definition 4.3.1. *Let $SD(LD)$ denote the class of functions which may be computed by uniform families of arithmetic circuits of size bounded by L and Depth bounded by D.*

Then, as Berkowitz [3] proves:

Theorem 4.3.2. *Let α be a real number such that the product of two $n \times n$ matrices can be computed in $SD(n^\alpha, \log(n))$ (currently $\alpha < 2.376$, see [6]), and M an $n \times n$ matrix. Then for every $\varepsilon > 0$ the characteristic polynomial of M can be computed in $SD(n^{\alpha+\varepsilon+1}, \log^2(n))$. Moreover the circuit required can be computed uniformly (i.e. in logspace).* ■

Remark 4.3.3. Suppose the entries of the matrix considered in the preceding theorem are integers and t bounds their sizes. In that case the sizes of the intermediate results obtained during the circuit computation are bounded by $O(n(t + \log n))$. This is because the circuit is constructed as a tree of depth $O(\log n)$ whose nodes perform products of matrices, and the circuits realizing these products have multiplications only at the first level, with the rest involving additions. Using this algorithm may other problems can be solved in NC. For instance

1) $RANK(M)$ = the rank of an $n \times m$ matrix M with real entries ($m \leqq n$).
2) $CLEAN(M)$ = an $m \times m$ submatrix of the $n \times m$ matrix M of rank m.

Proposition 4.3.4. We have the following complexity bounds:
1) $RANK(M)$ can be computed in $SD(n^{\alpha+\varepsilon+1}, \log^2(n))$
2) $CLEAN(M)$ can be computed in $SD(n^{\alpha+\varepsilon+2}, \log^2(n))$. Moreover, if t bounds the sizes of the entries of M, the intermediate computed values have sizes bounded by $O(n(t+\log n))$.

Proof. 1) First compute $M^t \cdot M$, which is an $m \times m$ real symmetric matrix, and then compute $\phi(\lambda)$, its characteristic polynomial. $RANK(M)$ equals $m - \operatorname{ord}(\phi(\lambda))$, where ord is the degree of the lowest non-zero coefficient. The bounds follow from Berkowitz's result.

2) Compute, for $2 \leq i \leq n$ the rank r_i of the matrix given by the first i rows of M, and we put $r_1 = 1$. Now return the matrix containing the i-th row of M if and only if $r_i > r_{i-1}$. This given $n-1$ applications of $RANK(M)$ from which the bounds are deduced.

The claim about the sizes easily follows from the last remark because we may apply Berkowitz's algorithm to matrices of the form BB^t (with B a submatrix of M) whose entries have size bounded by $t + \log n$. The reader can see these as well as other parallel algorithms for computer algebra in [5]. ∎

5 Complexity of the Univariate Algorithm

For our complexity bounds we shall consider the "classical" circuits which multiply two n-bit numbers within size $O(n^2)$ and depth $O(\log(n))$. Also, we shall suppose that the input polynomials have integer coefficients.

Thus, let $p, q_1, \ldots, q_s \in \mathbb{Z}[X]$ and let us consider the following parameters: d, a bound on the degrees and sizes of $p, q_1, \ldots, q_s$ (we recall that if $p = a_n X^n + \cdots + a_0$ its size is defined as $|p| = \log(\sqrt{a_n^2 + \cdots + a_0^2})$), and r, the number of real roots of p. Then we have

Lemma 5.1. *Let $p, q \in \mathbb{Z}[X]$ with degrees d and e respectively and sizes bounded by t. Then the Sturm–Habicht query $Sth(p,q)$ can be computed by a binary circuit with size $O(d(d+e)^{5.5}(t+\log(d+e))^2)$ and depth $O(\log^2(d+e)(\log(d+e)+\log t))$.*

Proof. We know that the query $SQ(p,q)$ can be calculated as the principal coefficients of the Sturm–Habicht sequence of p and q. Now, since these coefficients are defined as determinants of matrices with size bounded by $d+e$ we get, using Berkowitz's result, a bound of $O(d(d+e)^{3.5})$ on the number of arithmetical operations, and a bound of $O(\log^2(d+e))$ for the parallel time complexity. Finally, since the entries have sizes bounded by t, the intermediate values are bounded in size by $O((d+e)(t+\log(d+e)))$. The statement easily follows. ∎

Proposition 5.2. *The algorithm PCSP runs in parallel time $O(\log s \log^3 d)$ using $O(sr^2(d(d\log d)^{7.5} + r^{13}\log^2 r))$ processors.*

Proof. At each node in the combining tree we compute $O(r^2)$ Sturm–Habicht queries. Since the degrees and the sizes of the second inputs of these queries are bounded by $O(d \log r)$, by applying Proposition 4.3, we can compute each one within parallel time $O(\log^3 d)$ using $O(d(d\log r)^{7.5})$ processors. On the other hand we call the CLEAN procedure for a matrix with $O(r^2)$ rows whose entries have unit size. According to the complexity of the CLEAN procedure this uses $O(r^{13}\log^2 r)$ processors within parallel time $O(\log^2 r)$. Adding up these bounds and considering that the combining tree has $O(s)$ nodes and depth $\log s$, we deduce the statement. ∎

Corollary 5.3. *Let d be a bound of the degree and size of p and r the number of its real roots,*

i) *The algorithm PRAN runs in parallel time $O(\log^4 d)$ using $O(dr^2(d(d\log d)^{7.5} + r^{13}\log^2 r))$ processors.*
ii) *If e is a bound on the degree and size of the polynomials in a list $\bar{q} = (q_1, \ldots, q_s)$, set $L = e + d\log r$. The algorithm PRANCSP runs in parallel time $O(\log^3 L)$ using $O(srdL^{7.5})$ processors.*

Proof. It is enough to observe that we can run *PRAN* with $p^{(i)}/i!$ instead of $p^{(i)}$ and those polynomials have size bounded by $i + |d| = O(d)$. So we just substitute d for s in the preceding proposition.

Since the roots are separated, we can now independently apply the procedure *VCSPC* for each q_j $1 \leqq j \leqq s$. Each one of these computations calculates $2r$ Sturm–Habicht sequences whose second inputs have degree and size bounded by L. As before, this can be done within parallel time $O(\log^3 L)$ using $O(drL^{7.5})$ processors. Note that in this case we do not need to clean the resulting $3r \times 3r$ matrix since the independent part is formed by the first r rows. ■

Remark 5.4. One observes that the parameter r is not strictly meaningful since its worst case value coincides with d. We have included it here because it turns out to be important from an average case point of view. It can be shown that for a wide class of random distributions on the coefficients of a polynomial of degree d, the expected number of real roots of such a polynomial is asymptotically $(2/\pi)\log d$. So, on average we shall need only $O(sd\log^2 d(d\log d)^{7.5})$ processors when running *PCSP*. For these results see [11].

6 Complexity of the Multivariate Case

The total complexity of the arithmetic on real algebraic numbers is clearly in the class *NC* of the parameter D, product of the degrees of the polynomials in $\bar{p}$.

In terms of the number n of inputs, we get a circuit of size $d^{O(n)}$, where d is a bound on the degrees of the polynomials.

The determination of each symmetric function needed in Hermite quadratic form amounts to n table look-ups followed by n multiplications. Clearly this can be done in constant time in parallel. All the different symmetric functions may be evaluated independently (hence simultaneously) as well, so that the parallel complexity is a constant.

The overall circuit complexity for any fixed n remains in *NC*.

7 Conclusion

We summarize here the total complexity of our algorithms.

In all cases the parallel *BKR* combining tree is in NC^1, *i.e.* it has depth $\log(s)$ where s bounds the size of the input encoding. The size of the parallel *BKR* combining tree is linear in s and polynomial in the number of real roots r of the system.

In the multivariate case have shown algorithms for the Sturm queries at the nodes of the combining tree whose circuit size is bounded by $D^{O(1)}$, where D bounded the products of the degrees of n inputs. This is in *NC* for any fixed n, and in particular it is always in *NC* for the order relations, which involve only some linear polynomials.

In the univariate case we exhibited a number of optimizations involving Sturm–Habicht sequences which gives NC algorithms for all the cases.

References

1. Balcázar, J. L., Díaz, J., Gabarró, J.: Structural Complexity. vol 2, EATCS Monographs of Theoretical Computer Science. Berlin, Heidelberg, New York: Springer 1990
2. Ben-Or, M., Kozen, D., Reif, J.: The complexity of elementary algebra and geometry. J. Comp. Sys. Sci. **32**, 251–264 (1986)
3. Berkowitz, S.: On computing the determinant in small parallel time with a small number of processors. Inform. Process. Lett. **18**, 147–150 (1984)
4. Bochnak, J., Coste, M., Roy, M.-F.: Géométrie algébrique réelle. Berlin, Heidelberg, New York: Springer 1988
5. Borodin, A., von zur Gathen, J., Hopcroft, J.: Fast parallel matrix and GCD computations. Inform. Control. **52**, 241–256 (1982)
6. Coppersmith, D., Vinograd, S.: Matrix multiplication via Arithmetic progressions. J. Symb. Comput. **9**, 251–280 (1990)
7. Coste, M., Roy, M.-F.: Thom's lemma, the coding of real algebraic numbers and the computation of the topology of semi-algebraic sets. J. Symb. Comp. **5**, 121–129 (1988)
8. Cucker, F., González, L., Rosselló, F.: On algorithms for real algebraic plane curves. Proceedings of the MEGA'90. Progress in Mathematics. vol. **14**, pp. 63–89, Basel: Birkhäuser 1991
9. Cucker, F., González, L., Roy, M.-F., Szpirglas, A.: IF, a package for deciding systems of inequalities. Proceedings of the IX Conference of the Chilean Computer Science Society, pp. 77–86 (1989)
10. Cucker, F., Pardo, L. M., Raimondo, M., Recio, T., Roy, M.-F.: On the computation of the local and global analytic branches of a real algebraic curve. Proceedings of the A.A.E.C.C.-5, 1987. Springer LNCS vol. **356**, pp. 161–182. Berlin, Heidelberg, New York: Springer 1989
11. Cucker, F., Roy, M.-F.: A theorem on random polynomials and some consequences in average complexity. J. Symb. Comp. **10**, 405–409 (1989)
12. Fitchas, N., Galligo, A., Morgenstern, J.: Algorithmes rapides en sequentiel et en parallèlle pour l'elimination des quantificateurs en Géometrie Elementaire. Séminaire sur les Structures Algébriques Ordonnées. Publication Mathématiques de l'Université Paris VII, **32**, 103–145 (1989)
13. von zur Gathen, J.: Parallel arithmetic computations: a survey. Proc. 13th Conf. MFCS. LNCS vol. **233**, pp. 93–112. Berlin, Heidelberg, New York: Springer
14. González, L., Lombardi, H., Recio, T., Roy, M.-F.: Sturm-Habicht sequence. Proceedings of the ISSAC-89, Portland, 136–145 (1989)
15. González, L., Lombardi, H., Recio, T., Roy, M.-F.: Sous-résultants et spécialisation de la suite de Sturm I et II. I: Informatique Théorique Applications **24**, 561–588 (1990); II: to appear in Informatique Théorique et Applications
16. Heintz, J., Roy, M.-F., Solerno, P.: On the complexity of semialgebraic sets. Proceedings of the IFIP'89, San Francisco. North-Holland, 293–298 (1989)
17. Korkina, E. I., Kushnirenko, A. G.: Another proof of the Tarski–Seidemberg theorem. Translated from Sibirskij Matematicheskij Zhurnal **26**, 94–98 (1985)
18. Mishra, B., Pedersen, P.: Arithmetic with real algebraic numbers is in NC. Proceedings of ISSAC-90, 120–126 (1990)
19. Pedersen, P.: Counting Real Zeros, thesis, New York University, NYU Technical Report 545-R243 (1991)
20. Pedersen, P., Roy, M.-F., Szpirglas, A.: Counting Real Zeros in the multivariate case, to appear in MEGA 92
21. Pippenger, N.: On simultaneous resource bounds. Proceedings of 20th Found. of Comp. Sci., pp. 307–311, 1979
22. Roy, M.-F.: Computation of the topology of a real algebraic curve. Proceedings of the Conference on Computational geometry and topology. Astérisque **192**, 17–33 (1990)
23. Roy, M.-F., Szpirglas, A.: Complexity of computations on real algebraic numbers. J. Symb. Comp. **10**, 39–51 (1990)
24. Sylvester, J. T.: On a theory of syzygetic relations of two rational integral functions, comprising an application to the theory of Sturm's function. Trans. R. Soc. London, 429–586 (1853)