A Separation Bound for Real Algebraic Expressions*

Christoph Burnikel[†] Stefan Funke[‡] Kurt Mehlhorn[‡] Stefan Schirra[§]
Susanne Schmitt[‡]

April 16, 2003

Abstract

Real algebraic expressions are expressions whose leaves are integers and whose internal nodes are additions, subtractions, multiplications, divisions, k-th root operations for integral k, and taking roots of polynomials whose coefficients are given by the values of subexpressions. We consider the sign computation of real algebraic expressions, a task vital for the implementation of geometric algorithms. We prove a new separation bound for real algebraic expressions and compare it analytically and experimentally with previous bounds. The bound is used in the sign test of the number type `leda::real`.

1 Introduction

Real algebraic expressions are expressions whose leaves are integers and whose internal nodes are additions, subtractions, multiplications, divisions, k-th root operations for integral k, and taking roots of polynomials whose coefficients are given by the values of subexpressions; the exact definition is given below. Examples are $\sqrt{17}+\sqrt{21}-\sqrt{\sqrt{17}+\sqrt{21}+2\sqrt{357}}$ and $\frac{17+\sqrt{21}}{19}-\frac{18+\sqrt{22}}{20}$. We consider the sign computation of real algebraic expressions.

Our main motivation is the implementation of geometric algorithms. The evaluation of geometric predicates, such as the incircle or the side-of predicate, amounts to the computation of the sign of an expression. Non-linear objects (circles, ellipses, ...) lead to expressions involving roots and hence an efficient method for computing signs of algebraic expressions is an essential basis for the robust implementation of geometric algorithms dealing with non-linear objects.

The separation bound approach is the most successful approach to sign computation; it is, for example, used in the number type `leda::real` [BMS96, BFMS99, MN99] and the number type `Expr` of the `CORE` package [KLPY99]. A *separation bound* is an easily computable function *sep* mapping expressions into positive real numbers such that the value ξ of any non-zero expression E is lower bounded by $sep(E)$, i.e.,

$$\text{either} \quad \xi = 0 \quad \text{or} \quad |\xi| \geq sep(E) \ .$$

Separation bounds allow one to determine the sign of an expression by numerical computation. An error bound Δ is initialized to some positive value, say $\Delta = 1$, and an approximation $\widetilde{\xi}$ of ξ with $|\xi-\widetilde{\xi}| \leq \Delta$ is computed using approximate arithmetic, say floating point arithmetic with arbitrary-length mantissa. If $|\widetilde{\xi}| > \Delta$, the sign of ξ is equal to the sign of $\widetilde{\xi}$. Otherwise, $|\widetilde{\xi}| \leq \Delta$ and hence $|\xi| < 2\Delta$. If $2\Delta \leq sep(E)$, we have $\xi = 0$. If $2\Delta > sep(E)$, we halve Δ and repeat. The worst case complexity of the procedure just outlined is determined by the separation bound; $\log(1/sep(E))$ determines the maximal precision needed for the computation of $\widetilde{\xi}$ and we refer to $\log(1/sep(E))$ as the *bit bound*. If $\xi \neq 0$, the actual

*Partially supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces).

[†] `burnikel@mpi-sb.mpg.de`, ENCOM GmbH, 66740 Saarlouis, Germany

[‡] `{funke,mehlhorn,sschmitt}@mpi-sb.mpg.de`, MPI für Informatik, 66123 Saarbrücken, Germany

[§] `stschirr@isg.cs.uni-magdeburg.de`, Fakultät für Informatik, Otto-von-Guericke Universität Magdeburg, 39106 Magdeburg, Germany

precision required is $\log(1/\xi)$ and hence "easy sign tests" are much faster than the worst case. This feature distinguishes the separation bound approach to sign computation from approaches that explicitely compute a defining polynomial.

Separation bounds have been studied extensively in computer algebra [Can87, Mig82, Sch00, Mig92, Yap99], as well as in computational geometry [BMS94, BFMS00, Yap97, LY01, MS01]. We prove a new separation bound for the following class of *real algebraic expressions*. The value of a real algebraic expression is either a real algebraic number or undefined (at the end of Section 3 we show how to test whether the value of an expression is defined).

(1) Any integer v is a real algebraic expression. The integer is also the value of the expression.

(2) If E_1 and E_2 are real algebraic expressions, so are E_1+E_2, E_1-E_2, $E_1\cdot E_2$, E_1/E_2, and $\sqrt[k]{E_1}$, where $k\geq 2$ is an integer. The value of $\sqrt[k]{E_1}$ is undefined if k is even and the value of E_1 is negative. The value of E_1/E_2 is undefined, if the value of E_2 is zero. The value of E_1+E_2, E_1-E_2, $E_1\cdot E_2$, E_1/E_2, or $\sqrt[k]{E_1}$ is undefined, if the value of E_1 or the value of E_2 is undefined. Otherwise the value of E_1+E_2, E_1-E_2, $E_1\cdot E_2$, and E_1/E_2 is the sum, the difference, the product and the quotient of the values of E_1 and E_2 respectively and the value of $\sqrt[k]{E_1}$ is the k-th root of the value of E_1.

(3) If E_d, E_{d-1}, ..., E_1, E_0 are real algebraic expressions and j is a positive integer with $0\leq j<d$, then $\diamond(j,E_d,E_{d-1},\ldots,E_1,E_0)$ is an expression. If the values of the E_i are defined and ξ_i is the value of E_i, the value of the expression is the j-th smallest real root of the polynomial $\xi_d X^d+\xi_{d-1}X^{d-1}+\ldots+\xi_0$, if the polynomial has at least j real roots. Otherwise, the value is undefined.

Below, expression always means real algebraic expression. An expression is given as a directed acyclic graph (dag) whose source nodes are labeled by the operands and whose internal nodes are labeled by operators. We call an expression *simple* if only items (1) and (2) are used in its definition and we call it *simple and division-free* if, in addition, no division operator occurs in the expression.

The starting point for the present work is the bound given by Burnikel et al. [BFMS00] for simple expressions. We refer to this bound as the BFMS bound in the sequel.

Lemma 1 ([BFMS00]) *Let E be an expression with integral operands and operations $+,-,\cdot,/,\sqrt[k]{\ }$ for integral $k\geq 2$. Let ξ be the value of E, let the weight $D(E)$ of E be the product of the indices (the index of a $\sqrt[k]{\ }$ operation is k) of the radical operations in E, and let $u(E)$ and $l(E)$ be defined inductively on the structure of E by the rules shown in the table below.*

	$u(E)$	$l(E)$
integer N	$\lvert N\rvert$	1
$E_1\pm E_2$	$u(E_1)\cdot l(E_2)+l(E_1)\cdot u(E_2)$	$l(E_1)\cdot l(E_2)$
$E_1\cdot E_2$	$u(E_1)\cdot u(E_2)$	$l(E_1)\cdot l(E_2)$
E_1/E_2	$u(E_1)\cdot l(E_2)$	$l(E_1)\cdot u(E_2)$
$\sqrt[k]{E_1}$	$\sqrt[k]{u(E_1)}$	$\sqrt[k]{l(E_1)}$

Then $\xi=0$ or

$$\left(l(E)u(E)^{D(E)^2-1}\right)^{-1}\leq|\xi|\leq u(E)l(E)^{D(E)^2-1}.$$

If E is division-free, $l(E)=1$, and the above bound holds with $D(E)^2$ replaced by $D(E)$.

Observe the difference between the division-free case and the general case. For simple division-free expressions, the BFMS-bound is the best bound known. Expressions with divisions arise naturally in geometric applications. Inputs to expressions are frequently fractions and, e.g., normalizing a line equation amounts to a division. For expressions with divisions, the BFMS-bound is much weaker than for expressions without divisions. We give an example. Consider the expression

$$\frac{2^{10}\sqrt[8]{2^8-(2^8-1)}-2^6}{1}.$$

Here $u(E) \approx 2^{10}$, $l(E) = 1$ and $D(E) = 8$. So the BFMS bound is $2^{-10\cdot 63} = 2^{-630}$, since E is not division-free and hence the dependence (of the logarithm of the bound) on D is quadratic. Without the final redundant division, the expression is division-free and the bound becomes $2^{-10\cdot 7} = 2^{-70}$. Our new bound handles divisions much better and also applies to a wider class of expressions than the BFMS bound.

This paper is structured as follows. In Section 2, we review the proof of the BFMS bound and motivate our new way of dealing with divisions. In Section 3, we prove our main theorem, a separation bound for expressions defined by (1), (2), and (3). In Sections 4 and 5 we compare our bound analytically and experimentally to previous bounds.

2 A Review of the BFMS bound

An algebraic integer is the root of a polynomial with integer coefficients and leading coefficient one. The following three Lemmas were already used in [BFMS00] and [LY01].

Lemma 2 *Let α be an algebraic integer and let $\deg(\alpha)$ be the algebraic degree of α. If U is an upper bound on the absolute value of all conjugates of α, then*

$$|\alpha| \geq U^{1-\deg(\alpha)}$$

Proof: The proof is simple. Let d be the degree of α and let $\alpha_1 = \alpha$, α_2, ..., α_d be the conjugates of α. The product of the conjugates is equal to the constant coefficient of the defining polynomial and hence in $\mathbb{Z}$. Thus $|\alpha| \cdot U^{d-1} \geq 1$. ∎

Lemma 3 ([Hec70, Loo82] or [BFMS00, Theorem 4]) *Let α and β be algebraic integers. Then $\alpha \pm \beta$, $\alpha\beta$ and $\sqrt[k]{\alpha}$ are algebraic integers.*

We also need to cover item (3) in the definition of algebraic expressions.

Lemma 4 *Let ρ be the root of a monic polynomial*

$$P(X) = X^n + \alpha_{n-1}X^{n-1} + \alpha_{n-2}X^{n-2} + \cdots + \alpha_0$$

of degree n where the coefficients $\alpha_{n-1}, \alpha_{n-2}, \ldots, \alpha_0$ are algebraic integers. Then ρ is an algebraic integer.

Proof: This fact is well-known, a proof can, for example, be found in [Neu90, Theorem 2.4]. We include a proof for completeness. The proof uses an argument similar to the proof of Lemma 3. Let $\alpha_j^{(i_j)}$, $1 \leq i_j \leq \deg(\alpha_j)$, be the conjugates of α_j for $0 \leq j \leq n-1$ and let $\widehat{\alpha}_j$ be the vector formed by the conjugates of α_j. Consider the polynomial

$$Q(X) = \prod_{i_0}\prod_{i_1}\cdots\prod_{i_{n-1}}(X^n + \alpha_{n-1}^{(i_{n-1})}X^{n-1} + \alpha_{n-2}^{(i_{n-2})}X^{n-2} + \cdots + \alpha_0^{(i_0)}).$$

ρ is a root of $Q(X)$ and $Q(X)$ is symmetric in the $\alpha_j^{(i_j)}$ for all j. The theorem on elementary symmetric function implies that $Q(X)$ is a polynomial in X and the elementary symmetric functions $\sigma_1(\widehat{\alpha}_j)$, ..., $\sigma_{\deg(\alpha_j)}(\widehat{\alpha}_j)$. The elementary symmetric function $\sigma_l(\widehat{\alpha}_j)$ is the coefficient of $X^{\deg(\alpha_j)-l}$ in the minimal polynomial of α_j and hence in $\mathbb{Z}$ (since α_j is an algebraic integer). Thus $Q(X)$ is a monic polynomial in $\mathbb{Z}[X]$ and ρ is an algebraic integer. ∎

Lemma 5 ([BFMS00, Lemma 6]) *Let α and β be algebraic integers and let U_α and U_β be upper bounds on the absolute size of the conjugates of α and β, respectively. Then $U_\alpha + U_\beta$ is an upper bound on the absolute size of the conjugates of $\alpha \pm \beta$, $U_\alpha U_\beta$ is an upper bound on the absolute size of the conjugates of $\alpha\beta$, and $\sqrt[k]{U_\alpha}$ is an upper bound on the absolute size of the conjugates of $\sqrt[k]{\alpha}$.*

We also need bounds for the absolute size of roots of monic polynomials. Let $P(X) = X^n + a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + \cdots + a_0$ be a monic polynomial with arbitrary real coefficients, not necessarily integral, and let α be a root of $P(X)$. A *root bound* Φ is any function of the coefficients of P that bounds the absolute value of α, i.e.,

$$|\alpha| \le \Phi(a_{n-1}, a_{n-2}, \ldots, a_0)$$

We require that Φ is monotone, i.e., if $|a_i| \le b_i$ for $0 \le i \le n-1$, then

$$\Phi(a_{n-1}, a_{n-2}, \ldots, a_0) \le \Phi(b_{n-1}, b_{n-2}, \ldots, b_0).$$

Examples of root bounds are:

$$\begin{aligned}
|\alpha| &\le 2\max\left(|a_{n-1}|, \sqrt{|a_{n-2}|}, \sqrt[3]{|a_{n-3}|}, \ldots, \sqrt[n]{|a_0|}\right)\\
|\alpha| &\le 1 + \max\left(|a_{n-1}|, |a_{n-2}|, \ldots, |a_0|\right)\\
|\alpha| &\le \max\left(n|a_{n-1}|, \sqrt{n|a_{n-2}|}, \sqrt[3]{n|a_{n-3}|}, \ldots, \sqrt[n]{n|a_0|}\right)\\
|\alpha| &\le \left(\sqrt[n]{2} - 1\right)^{-1} \max\left(\frac{|a_{n-1}|}{\binom{n}{1}}, \sqrt{\frac{|a_{n-2}|}{\binom{n}{2}}}, \sqrt[3]{\frac{|a_{n-3}|}{\binom{n}{3}}}, \ldots, \sqrt[n]{\frac{|a_0|}{\binom{n}{n}}}\right)
\end{aligned}$$

A proof of all bounds can be found in [Yap99]. The first bound is called the Lagrange-Zassenhaus bound and the middle two bounds are called the Cauchy bounds.

We next briefly review the proof of the BFMS bound. For a division-free simple expression E one observes that the value ξ of E is an algebraic integer (by Lemma 3) and that $u(E)$ is an upper bound on ξ and all its conjugates (by Lemma 5). Furthermore $D(E)$ is an upper bound for the algebraic degree of E. Thus $|\xi| \le u(E)$ and $|\xi| \ge 1/(u(E)^{D(E)-1})$ by Lemma 2.

Expressions with divisions are handled by reduction to the division-free case. Let E be a simple expression and let ξ be its value. We construct a new expression dag, also with value $\xi = val(E)$, containing only a single division. Moreover, the division is the final operation in the dag and hence $val(E) = val(E_1)/val(E_2)$, where E_1 and E_2 are the inputs to the division. The bounds for the division free case apply to E_1 and E_2 and $D(E_1)$ and $D(E_2)$ are at most $D(E)^2$. The construction of the new dag is straightforward. For every node A in the original dag there are two nodes A_1 and A_2 in the new dag such that $val(A) = val(A_1)/val(A_2)$. For the leaves (which stand for integers) the replacement is trivial (we take $A_1 = A$ and $A_2 = 1$) and for interior nodes we use the rules

$$\frac{A_1}{A_2} \pm \frac{B_1}{B_2} \Longrightarrow \frac{A_1B_2 \pm A_2B_1}{A_2B_2} \qquad \frac{A_1}{A_2} \cdot \frac{B_1}{B_2} \Longrightarrow \frac{A_1A_2}{B_1B_2} \qquad \frac{A_1}{A_2} \Big/ \frac{B_1}{B_2} \Longrightarrow \frac{A_1B_2}{A_2B_1} \qquad \sqrt[k]{\frac{A_1}{A_2}} \Longrightarrow \frac{\sqrt[k]{A_1}}{\sqrt[k]{A_2}}\,.$$

In this way, *each root operation in the original dag gives rise to two root operations in the new dag*. This may square the D-value of the expression.

The starting point for the present paper was a simple but powerful observation. Although the transformation rules above are natural, they are not the only way of obtaining division free expressions E_1 and E_2 with $val(E) = val(E_1)/val(E_2)$. Instead of the last rule we may also use

$$\sqrt[k]{\frac{A_1}{A_2}} \quad \Longrightarrow \quad \frac{\sqrt[k]{A_1A_2^{k-1}}}{A_2} \quad \text{or} \quad \frac{A_1}{\sqrt[k]{A_1^{k-1}A_2}}\,.$$

The new rule does not increase the D-value of the expression and hence $D(E_1)$ and $D(E_2)$ are at most $D(E)$. In an earlier version of the paper, we only used the first alternative of the new rule. Chee Yap (personal communication, January 2001) pointed out to us that it is advantageous to have both rules (see the proof of Lemma 6).

3 The New Bound

We derive a separation bound for the expressions defined by items (1) to (3). For items (1) and (2), we use the BFMS rules with the modification proposed in the previous paragraph. The diamond operation allows one to take the root of a polynomial

$$P(X) = \alpha_d X^d + \alpha_{d-1} X^{d-1} + \cdots + \alpha_1 X + \alpha_0$$

where the α_i are arbitrary real algebraic numbers. Every real algebraic number can be written as the quotient of two algebraic integers; this is well-known, but will be reproved below as part of the proof of our main theorem. Let $\alpha_i = \nu_i/\delta_i$ where ν_i and δ_i are algebraic integers. Then

$$P(X) = \frac{\nu_d}{\delta_d} X^d + \frac{\nu_{d-1}}{\delta_{d-1}} X^{d-1} + \cdots + \frac{\nu_1}{\delta_1} X + \frac{\nu_0}{\delta_0}\ .$$

Let $D = \prod \delta_i$. By multiplication with D we obtain

$$D \cdot P(X) = (\nu_d D/\delta_d) X^d + (\nu_{d-1} D/\delta_{d-1}) X^{d-1} + \cdots + (\nu_1 D/\delta_1) X + (\nu_0 D/\delta_0),$$

a polynomial whose coefficients are algebraic integers. We next derive a monic polynomial. To get rid of the leading coefficient $(\nu_d D/\delta_d)$, we multiply by $(\nu_d D/\delta_d)^{d-1}$ and substitute $X/(\nu_d D/\delta_d)$ for X. We obtain

$$D \cdot (\nu_d D/\delta_d)^{d-1} \cdot P\left(\frac{X}{\nu_d D/\delta_d}\right) = Q(X) =$$

$$X^d + (\nu_{d-1} D/\delta_{d-1}) X^{d-1} + \cdots + (\nu_d D/\delta_d)^{d-2} (\nu_1 D/\delta_1) X + (\nu_d D/\delta_d)^{d-1} (\nu_0 D/\delta_0)$$

which is monic and has algebraic integer coefficients. The root bounds of Section 2 provide us with an upper bound on the size of the roots of $Q(X)$: the size of any root of $Q(X)$ is bounded by

$$u = \Phi((\nu_{d-1} D/\delta_{d-1}), \ldots, (\nu_d D/\delta_d)^{d-2} (\nu_1 D/\delta_1), (\nu_d D/\delta_d)^{d-1} (\nu_0 D/\delta_0)).$$

Since the roots of P are simply the roots of Q divided by $\nu_d D/\delta_d$, this suggests to extend the definitions of u and l as follows: For an expression E denoting a root of a polynomial of degree d with coefficients given by $E_d, E_{d-1}, E_{d-2}, \ldots, E_0$ we define

$$u(E) = \Phi(\ldots, \left(u(E_d) \prod_{k \neq d} l(E_k)\right)^{d-i-1} u(E_i) \prod_{k \neq i} l(E_k), \ldots) \quad \text{and} \quad l(E) = u(E_d) \prod_{k \neq d} l(E_k)\ .$$

We still need to define the weight $D(E)$ of an expression. We do so in the obvious way. The weight $D(E)$ of an expression dag E is the product of the weights of the nodes and leaves of the dag. Leaves and $+, -, \cdot$ and $/$-operations have weight 1, a $\sqrt[k]{\ }$-node has weight k, and a $\diamond(j, E_d, \ldots)$-operation has weight d.

We can now state our main theorem.

Theorem 1 *Let E be an expression with integer operands and operations $+, -, \cdot, /, \sqrt[k]{\ }$ for integral k and $\diamond(j, \ldots)$ operations. Let ξ be the value of E. Let $u(E)$ and $l(E)$ be defined inductively on the structure of E according to the following rules:*

	$u(E)$	$l(E)$
integer N	$\lvert N \rvert$	1
$E_1 \pm E_2$	$u(E_1) \cdot l(E_2) + l(E_1) \cdot u(E_2)$	$l(E_1) \cdot l(E_2)$
$E_1 \cdot E_2$	$u(E_1) \cdot u(E_2)$	$l(E_1) \cdot l(E_2)$
E_1 / E_2	$u(E_1) \cdot l(E_2)$	$l(E_1) \cdot u(E_2)$
$\sqrt[k]{E_1}$ *and* $u(E_1) \geq l(E_1)$	$\sqrt[k]{u(E_1) l(E_1)^{k-1}}$	$l(E_1)$
$\sqrt[k]{E_1}$ *and* $u(E_1) < l(E_1)$	$u(E_1)$	$\sqrt[k]{(u(E_1)^{k-1} l(E_1)}$
$\diamond(j, E_d, \ldots, E_0)$	$\Phi(\ldots, \left(l(E)^{d-i-1} u(E_i) \prod_{k \neq i} l(E_k)\right), \ldots)$	$u(E_d) \prod_{k \neq d} l(E_k)$

Let $D(E)$ be the weight of E. Then either $\xi = 0$ or

$$\left(l(E) u(E)^{D(E)-1}\right)^{-1} \leq \lvert \xi \rvert \leq u(E) l(E)^{D(E)-1}$$

Proof: We show that the rules for u and l keep the invariant that there are algebraic integers β and γ such that $\xi = \beta/\gamma$ and $u(E)$ is an upper bound on the absolute size of the conjugates of β and $l(E)$ is an upper bound on the absolute size of the conjugates of γ.

We prove this by induction on the structure of E. The base case is trivial. If E is an integer N, we take $\beta = N$ and $\alpha = 1$; β is the root of the polynomial $X - N$ and α is a root of $X - 1$.

Now let $E = E_1 \pm E_2$. By induction hypothesis we have $\xi_j = \beta_j/\gamma_j$ for $j = 1,2$. We set $\beta = \beta_1\gamma_2 \pm \beta_2\gamma_1$ and $\gamma = \gamma_1\gamma_2$. Since algebraic integers are closed under additions, subtractions and multiplications, β and γ are algebraic integers. By Lemma 5, $u(E) = u(E_1) \cdot l(E_2) + l(E_1) \cdot u(E_2)$ is an upper bound on the absolute size of the conjugates of β. Similarly, $l(E)$ is an upper bound on the absolute size of the conjugates of γ.

If $E = E_1 \cdot E_2$, we set $\beta = \beta_1\beta_2$ and $\gamma = \gamma_1\gamma_2$. The claim follows analogously to the previous case by Lemma 5.

If $E = E_1/E_2$, we set $\beta = \beta_1\gamma_2$ and $\gamma = \beta_2\gamma_2$. Again, the claim follows using Lemma 5.

If $E = \sqrt[k]{E_1}$ and $u(E_1) \geq l(E_1)$, we set $\beta = \sqrt[k]{\beta_1\gamma_1^{k-1}}$ and $\gamma = \gamma_1$. Since algebraic integers are closed under $\sqrt[k]{\ }$-operations, β is an algebraic integer. By Lemma 5, $u(E)$ is an upper bound on the absolute size of the conjugates of β. There is nothing to show for $\gamma = \gamma_1$.

If $E = \sqrt[k]{E_1}$ and $u(E_1) < l(E_1)$, we set $\beta = \beta_1$ and $\gamma = \sqrt[k]{\beta_1^{k-1}\gamma_1}$. Since algebraic integers are closed under $\sqrt[k]{\ }$-operations, γ is an algebraic integer. By Lemma 5, $l(E)$ is an upper bound on the absolute size of the conjugates of γ. There is nothing to show for $\beta = \beta_1$.

Finally, let E be defined by a $\diamond(j, E_d, \ldots, E_0)$-operation. We set

$$\gamma = \beta_d\gamma_{d-1}\gamma_{d-2}\cdots\gamma_0$$

and

$$\beta = \diamond(j, 1, \beta_{d-1}\gamma_d\gamma_{d-2}\cdots\gamma_0, \gamma\beta_{d-2}\gamma_d\gamma_{d-1}\gamma_{d-3}\cdots\gamma_0, \ldots, \gamma^{d-1}\gamma_d\gamma_{d-1}\gamma_{d-2}\cdots\gamma_1\beta_0).$$

By the discussion preceding the statement of our main theorem, $\xi = \beta/\gamma$, β and γ are algebraic integers, $l(E)$ is an upper bound on the absolute size of the conjugates of γ, and $u(E)$ is an upper bound on the absolute value of the conjugates of β. This completes the induction step.

Rewriting ξ as β/γ corresponds to a restructuring of the expression dag defining E into an expression dag E' with a single division-operation. We have $D(E') = D(E)$.

We still need to argue that $D(E)$ is an upper bound on the algebraic degree of β. This follows from the fact that every operation leads to a field extensions whose degree is bounded by the weight of the operation.

We now have collected all ingredients to bound the absolute value of ξ from below. If $\xi \neq 0$, we have $\beta \neq 0$. The absolute value of β and all its conjugates is bounded by $u(E)$. Thus $|\beta| \geq (u(E)^{\deg(\beta)-1})^{-1}$ by Lemma 2. Also $|\gamma| \leq l(E)$. Thus

$$|\xi| = \frac{\beta}{\gamma} \geq \frac{1}{u(E)^{\deg(\beta)-1}} \cdot \frac{1}{l(E)} \geq \frac{1}{u(E)^{D(E)-1} \cdot l(E)}$$

∎

The value of an algebraic expression may be undefined. Divisions by zero and taking a root of even degree of a negative number are easily caught by the sign test. We next argue that the sign test also allows us to test whether the diamond-operation is well defined. For this matter, we need to determine the number of zeros of a polynomial. Sturm sequences, see [Mig92, chapter 5] or [Yap99, Chapter 7] are the appropriate tool. The computation of Sturm sequences amounts to a gcd computation between a polynomial and its derivative. Our sign test is sufficient to implement a gcd computation.

4 Comparison to Other Constructive Root Bounds

We compare our new bound to previous root bounds provided by Mignotte [Mig92], Canny [Can87], Dubé/Yap [YD95], BFMS [BFMS00, MS01], Scheinermann [Sch00], Li/Yap [LY01]. We refer to the bound presented in this paper as BFMSS. Root bounds can be compared along two axees: according to the class of expressions to which they apply and according to their value.

The bound by Scheinerman applies to division-free simple expressions, the bounds by Mignotte, Dubé/Yap and the BFMS bound apply to simple expressions. The bounds in [LY01] and [MS01] apply to expressions defined by items (1) to (3) with the restriction that the E_d to E_0 in (3) must be integers. Canny's bound is most general. It applies to algebraic numbers defined by systems of multi-variate polynomial equations with integer coefficients.

We next discuss the quality of the bounds. In [BFMS00, LY01] it was already shown that the BFMS-bound is never worse than the bounds by Mignotte, Canny, Dubé/Yap, and Scheinermann for division-free expressions. In [LY01] it was also shown that the BFMS bound and the Li/Yap bound are incomparable.

Lemma 6 (C. Yap, personal communication) *Let E be an arbitrary simple expression, let u and l be defined as in the original BFMS-bound, let u' and l' be defined as in Theorem 1, and let $D = D(E)$ be the degree bound of E. Then*

$$l(E)u(E)^{D^2-1} \geq l'(E)u'(E)^{D-1} ,$$

i.e., the improved bound is always at least as strong as the orginal BFMS-bound.

Proof: We show

$$\frac{u(E)}{l(E)} = \frac{u'(E)}{l'(E)} \qquad \text{and} \qquad u'(E) \leq u(E)^{D(E)} \qquad \text{and} \qquad l'(E) \leq l(E)^{D(E)}$$

by induction on the structure of E. Assume that these relations hold. Then

$$l(E)u(E)^{D^2-1} = \frac{l(E)}{u(E)}(u(E)^D)^D \geq \frac{l'(E)}{u'(E)}u'(E)^D = l'(E)u'(E)^{D-1}$$

and we are done.

The proof of the equality is a simple induction on the structure of E. The base case is clear. In the inductive step we write u_1 instead of $u(E_1)$ and similarly for E_2, l, u' and l'. If $E = E_1 + E_2$, we have

$$\frac{u(E)}{l(E)} = \frac{u_1 l_2 + u_2 l_1}{l_1 l_2} = \frac{u_1}{l_1} + \frac{u_2}{l_2} = \frac{u'_1}{l'_1} + \frac{u'_2}{l'_2} = \frac{u'(E)}{l'(E)} .$$

Multiplication and division are handled similarly. If $E = \sqrt[k]{E_1}$, we have (assuming $u'_1 \geq l'_1$, the case $u'_1 < l'_1$ is handled similarly)

$$\frac{u(E)}{l(E)} = \frac{\sqrt[k]{u_1}}{\sqrt[k]{l_1}} = \sqrt[k]{\frac{u_1}{l_1}} = \sqrt[k]{\frac{u'_1}{l'_1}} = \frac{\sqrt[k]{u'_1(l'_1)^{k-1}}}{l'_1} = \frac{u'(E)}{l'(E)} .$$

For the inequalities we have to work slightly harder. The base case is again clear; observe that $D = 1$ in the base case. It is also clear that $u(E) \geq 1$ (or $u(E) = 0$) and $l(E) \geq 1$ for all E. If $E = E_1 \pm E_2$, we have (using $D \geq D_1$ and $D \geq D_2$)

$$u(E)^D = (u_1 l_2 + u_2 l_1)^D \geq (u_1 l_2)^D + (u_2 l_1)^D \geq u_1^{D_1} l_2^{D_2} + u_2^{D_2} l_1^{D_1} \geq u'_1 l'_2 + u'_2 l'_1 = u'(E)$$

and

$$l(E)^D = (l_1 l_2)^D \geq l_1^{D_1} l_2^{D_2} \geq l'_1 l'_2 = l'(E) .$$

Multiplication and division are handled similarly. If $E = \sqrt[k]{E_1}$, we have $D(E_1) = D(E)/k$ and hence (assuming $u'_1 \geq l'_1$, the case $u'_1 < l'_1$ is handled similarly)

$$u(E)^D = u_1^{D/k} = u_1^{D_1} \geq u'_1 \geq \sqrt[k]{u'_1(l'_1)^{k-1}} = u'(E) \qquad \text{and} \qquad l(E)^D = l_1^{D/k} = l_1^{D_1} \geq l'_1 = l'(E) .$$

∎

We next show that the new bound can be significantly better than the old bound. Consider the expression $F = \sqrt[k]{x/a}$ and $E = F - F$ where x is a ck-bit integer for some constant c and a is a d-bit integer for some constant d. Then $D(E) = k$. We evaluate both bounds as functions of k.

For the BFMS-bound we have $\log u(F) = (1/k)ck = c$, $\log l(F) = d/k$, $\log u(E) = 1 + c + d/k$, $\log l(E) = 2d/k$ and hence the BFMS bit bound is $(k^2 - 1)\log u(E) + \log l(E) = \Theta(k^2)$.

For the BFMSS-bound we have $\log u(F) = (1/k)(ck + d(k-1)) = c + d - d/k$, $\log l(F) = d$, $\log u(E) = 1 + c + 2d - d/k$, $\log l(E) = 2d$ and hence the BFMSS bit bound is $(k-1)\log u(E) + \log l(E) = \Theta(k)$.

It remains to compare the BFMSS and the Li/Yap bound. For division-free simple expressions, the bounds are identical. For expressions with divisions, the bounds are incomparable.

We start with an example, where the BFMSS-bound is significantly better. Let[1] $E_0 = 17/3$, let $F_i = \sqrt{E_{i-1}}$ and $E_i = F_i + F_i$ for $1 \le i \le k$, and let $E = E_k - E_k$. Then $\deg(E_i) = \deg(F_i) = 2^i$. We evaluate both bounds as functions of k.

For the BFMSS bound, we have

$$\begin{aligned}
\log u(E_0) &= \log 17, \\
\log l(E_0) &= \log 3, \\
\log l(F_i) &= \log l(E_{i-1}), \\
\log u(F_i) &= \frac{1}{2}(\log u(E_{i-1}) + \log l(E_{i-1})), \\
\log l(E_i) &= 2\log l(F_i) = 2\log l(E_{i-1}) = 2^i \log 3, \\
\log u(E_i) &= 1 + \log u(F_i) + \log l(F_i) \\
&= 1 + \frac{1}{2}\log u(E_{i-1}) + \frac{3}{2}\log l(E_{i-1}) \\
&= 1 + \frac{1}{2}\log u(E_{i-1}) + \frac{3}{2}2^{i-1}\log 3 \\
&= \sum_{0\le j<i} 2^{-j} + 2^{-i}\log 17 + 2^{i-1}\frac{3}{2}\log 3 \sum_{0\le j<i} 4^{-j} \\
&\le 2 + 2^{-i}\log 17 + 2^i \log 3
\end{aligned}$$

and hence $\log u(E) = 1 + \log u(E_k) + \log l(E_k) \le 3 + 4\cdot 2^k$ and $\log l(E) = 2\log l(E_k) \le 4\cdot 2^k$. We conclude that the BFMSS bit bound is equal to $(2^k - 1)(3 + 4\cdot 2^k) + 4\cdot 2^k = \Theta(4^k)$. Increasing k by one, quadruples the numbers of bits.

The Li/Yap bound involves the lead coefficient of the minimal polynomial and is at least the logarithm of the lead coefficient. Li and Yap compute the following estimates lc for the lead coefficients. Let $d_i = D(E_i) = D(F_i) = 2^i$. Then $\log lc(E_0) = \log 3 \ge 1$, $\log lc(F_i) = \log lc(E_{i-1})$, $\log lc(E_i) = 2\cdot d_i \cdot \log lc(F_i) = 2\cdot 2^i \log lc(E_{i-1}) = 2^i \prod_{1\le j\le i} 2^j \cdot \log 3 \ge 2^{i(i+3)/2}$, $\log lc(E) = 2\cdot 2^k \log lc(E_k)$ and hence the Li-Yap bit bound is $\Omega(2^{k^2/2})$. Increasing k by one multiplies the required number of bits by more than 2^k.

We next give an example where the Li/Yap bound is better. We start with the fraction 17/3, square k times and then take roots k times. The weight of the expression is 2^k and $\log u(E) \ge 2^k$. The BFMSS bit bound is therefore $\Omega(4^k)$. On the other hand, the Li/Yap bound is $O(2^k)$.

An implementation should compute the Li/Yap and BFMSS bounds and use the better of the bounds.

5 Experimental Evaluation

The separation bound approach to sign determination of algebraic numbers is used in the number types `real` of LEDA [LED] and `Expr` of CORE [KLPY99]. We report about the improvements in running time due to the new separation bounds and due to a recent reimplementation of `leda::real`. We also compare CORE and `leda::real`.

All tests are based on LEDA 4.4 with the most recent arithmetic module incorporated. For the tests with the CORE library we used CORE v1.5 available from [LY01]; it uses the Li/Yap-bound. All benchmarks

[1] Any other fraction will also work as the initial value.

are performed on a Sun Ultra 5 with a 400 MHz UltraSPARC-IIi processor, 128 MB RAM, running SunOS 5.9. We used g++ 3.2.1 as a compiler unless stated otherwise, times are always measured in seconds.

We briefly review the implementation of `leda::real`, a detailed description of an older implementation is available in [BMS96]. The number type supports the sign determination of simple algebraic expressions. Expressions are represented by their expression dag $\mathcal{G}(E)$. The input values of E are contained in the leaves of the dag, every inner node corresponds to an arithmetical operation, and the root corresponds to E.

When the sign of an algebraic number E needs to be dermined, the datatype first computes a separation bound q_E. Using `leda::bigfloat` arithmetic (= floating-point numbers with exponent and mantissa of arbitrary length), the datatype computes successively intervals of decreasing length that include E, until the interval does not contain zero or the length of the interval is less than q_E.

Several shortcuts are used to speed up the computation of the sign. First, a double approximation $\widetilde{E}$ and an error bound *err* such that $|E-\widetilde{E}| \leq err$ is stored with every node of the expression dag. As long as the double approximation $\widetilde{E}$ is known to be exact, i.e. $err = 0$, no expression graph is constructed and $\widetilde{E}$ represents E.

Secondly, if the double approximation $\widetilde{E}$ suffices to determine the sign of E, i.e. $0 \notin [\widetilde{E}-err, \widetilde{E}+err]$, no `leda::bigfloat` computation is triggered. This technique is called a *floating-point filter*.

In a reimplementation, we made the following improvements:

(1) the separation bound is the better of the Li/Yap and our improved bound.

(2) the implementation of the underlying `leda::bigfloat` arithmetic has been improved; at the beginning it was based on number type `leda::integer` for integer numbers of arbitrary size, now it directly operates on vectors of `long` integers.

(3) memory management within the `leda::real` datatype has been improved; in particular, space for the `leda::bigfloat` approximations is now only allocated if `leda::bigfloat` computation is necessary for a sign determination.

(4) the built-in floating-point filters have been improved, both with respect to running time as well as precision.

Improvements (2) to (4) have already been incorporated into public releases of LEDA starting with release 4.3. Overall, the efficiency has improved for 'easy instances' (i.e. instances that do not need the `leda::bigfloat` computation) due to improved floating-point filter techniques as well as for 'difficult instances' due to the improved separation bounds and `leda::bigfloat` implementation.

We turn to our experiments. The source code of all experiments is available at `http://www.mpi-sb.mpg.de/~funke/SepBound.html`. Many of the experiments make use of L-bit random integers. We generated them outside the `leda::real` number type and used them as inputs for our expressions.

(1) The first test is a simple check of a binomial expression. Let $x = \frac{a}{b}, y = \frac{c}{d}$ where a, b, c, and d are L-bit integers and let $E = (\sqrt{x}+\sqrt{y}) - \sqrt{x+y+2\sqrt{xy}}$. For the old BFMS-bound we get a $\text{sep}_{BFMS} = 160L+381$, for our improved bound $\text{sep}_{improv} = 96L+60$, whereas the LiYap-bound gives $\text{sep}_{LiYap} = 28L+60$. This is of course reflected in the running times.

L	25	50	100	200	400	800	1600
BFMS	0.04	0.07	0.19	0.51	1.46	4.25	13.11
Improv	0.01	0.04	0.08	0.19	0.53	1.51	4.60
LiYAP	0.00	0.01	0.02	0.04	0.09	0.22	0.62

(2) Let x and y be L-bit integers, $C = (\sqrt{x}-\sqrt{y})/(x-y))$ and $E = C - C$. For both our old and improved bound we get $\text{sep}_{BFMS} = \text{sep}_{Improv} = 6L+64$, whereas the LiYap-bound gives $\text{sep}_{LiYap} = 65L+91$. Again this shows in the running times.

L	500	1000	2000	4000	8000	16000
BFMS	0.02	0.04	0.11	0.25	0.70	2.03
Improv	0.02	0.04	0.10	0.25	0.70	2.06
LiYAP	0.48	1.39	4.07	12.09	36.15	112.91

We now turn to examples for which we have already proved differing asymptotic behaviour of the bounds in Section 4.

(3) First consider $F = \sqrt[k]{x/y}$ and $E = F - F$ where x is a $100k$-bit integer and y a 32-bit integer. The BFMS bound is $\Theta(k^2)$, whereas the new bound is $\Theta(k)$. The Li/Yap bound is also $\Theta(k)$ and even better than our new bound.

k	2	4	8	16	32	64
BFMS time	0.01	0.02	0.19	1.92	20.26	70.57
BFMS bound	391	1683	6751	26781	106396	421787
Improv. time	0.01	0.01	0.01	0.05	0.11	0.32
Improv. bound	214	538	1198	2524	5179	10459
LiYap time	0.01	0.01	0.01	0.03	0.04	0.12
LiYap bound	150	346	750	1564	3195	6427

(4) For $E_0 = 17/3, F_i = \sqrt{E_{i-1}}, E_i = F_i + F_i, E = E_k - E_k$, the BFMS and the BFMSS bound for E is $\Theta(4^k)$ (but with different constant factors), whereas the Li-Yap bound is $\Omega(2^{k^2})$.

k	2	3	4	5	6
BFMS time	0.01	0.01	0.06	1.25	12.55
BFMS bound	237	1213	5885	27645	126973
Improv. time	0.01	0.01	0.01	0.07	0.50
Improv. bound	76	284	1084	4220	16636
LiYap time	0.01	0.02	2.65	2354	(too long)
LiYap bound	140	2076	65596	4194428	536871164

In the following test we compare different implementations: `real(1)` (as of LEDA 4.2) denotes our old implementation[2] and `real(2)` the new implementation.

(5) As in our first example we take $x = \frac{a}{b}, y = \frac{c}{d}$, where a, b, c, d are L-bit integers, and $E = (\sqrt{x} + \sqrt{y}) - \sqrt{x + y + 2\sqrt{xy}}$. As we can see, the improved implementation of the `leda::real` datatype already leads to a speedup of a factor of 4, even when using the same separation bound. The new separation bounds give another considerable speedup.

L		25	50	100	200	400	800	1600	3200
`real(1)`	sep_{BFMS}	0.08	0.22	0.71	1.86	5.97	16.86	47.2	154.8
`real(2)`	sep_{BFMS}	0.03	0.08	0.17	0.49	1.49	4.25	12.9	39.33
`real(2)`	sep_{improv}	0.02	0.03	0.08	0.18	0.55	1.56	4.56	13.39
`real(2)`	sep_{LiYap}	0.01	0.01	0.02	0.04	0.08	0.22	0.59	1.70
CORE/Expr	sep_{LiYap}	0.04	0.10	0.10	0.11	0.82	0.84	12.21	12.34

(6) The final comparison concerns easy sign tests. The following expression arises during Fortune's sweep-line algorithm for Voronoi diagrams: $E = \frac{a+\sqrt{b}}{c} - \frac{a'+\sqrt{b'}}{c'}$ where a, a', b, b', c, and c' are random $3L$-, $6L$-, and $2L$-bit integers. The root bounds do not play a role here, only the efficiency of the implementation, in particular the floating-point filters comes into play.

L	50	100	200
`double`	0.01	0.01	0.01
`real(2)`	0.13	0.14	13
CORE/Expr	1.06	1.06	5354

Clearly, pure `double` arithmetic is the fastest, creating the expression dag does not come without cost. The huge increase in running time for $L = 200$ can be explained by the fact that in this case, the numbers get too large to be representable by a double (remember that we create integers of length $6L$). Therefore the floating-point filters will always fail and `leda::bigfloat` arithmetic has to be used.

[2] Unfortunately for this old version we only have timings available for compilations with g++ 2.95.4.

6 Conclusions

We presented a new separation bound for algebraic expressions. The bound applies to a wide class of expressions and is easily computable. For many expressions it gives much better bounds than previous bounds resulting in significant gains in running time. We see two main challenges: (1) For algebraic numbers defined by systems of polynomials, Canny's bound is the best bound known. Provide a better bound. (2) Our bound as well as the Li/Yap bound is very easy to compute. In the context of expensive sign computations it is worthwile to investigate more expensive methods for computing separation bounds.

References

[BFMS99] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. Exact efficient computational geometry made easy. In *Proceedings of the 15th Annual Symposium on Computational Geometry (SCG'99)*, pages 341–350, 1999.

[BFMS00] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. A strong and easily computable separation bound for arithmetic expressions involving radicals. *Algorithmica*, 27:87–99, 2000.

[BMS94] C. Burnikel, K. Mehlhorn, and S. Schirra. How to compute the Voronoi diagram of line segments: Theoretical and experimental results. In Springer, editor, *Proceedings of the 2nd Annual European Symposium on Algorithms - ESA'94*, volume 855 of *Lecture Notes in Computer Science*, pages 227–239, 1994.

[BMS96] C. Burnikel, K. Mehlhorn, and S. Schirra. The LEDA class *real* number. Technical Report MPI-I-96-1-001, Max-Planck-Institut für Informatik, Saarbrücken, 1996.

[Can87] J.F. Canny. *The Complexity of Robot Motion Planning*. The MIT Press, 1987.

[Hec70] E. Hecke. *Vorlesungen über die Theorie der algebraischen Zahlen*. Chelsea, New York, 1970.

[KLPY99] V. Karamcheti, C. Li, I. Pechtchanski, and Chee Yap. A core library for robust numeric and geometric computation. In *Proceedings of the 15th Annual ACM Symposium on Computational Geometry*, pages 351–359, Miami, Florida, 1999.

[LED] LEDA (Library of Efficient Data Types and Algorithms). `www.mpi-sb.mpg.de/LEDA/leda.html`.

[Loo82] R. Loos. Computing in algebraic extensions. In B. Buchberger, G. E. Collins, and R. Loos, editors, *Computer Algebra. Symbolic and Algebraic Computation*, volume 4 of *Computing Supplementum*, pages 173–188. Springer-Verlag, 1982.

[LY01] C. Li and C. Yap. A new constructive root bound for algebraic expressions. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 496–505, 2001.

[Mig82] M. Mignotte. Identification of Algebraic Numbers. *Journal of Algorithms*, 3(3):197–204, September 1982.

[Mig92] M. Mignotte. *Mathematics for Computer Algebra*. Springer, 1992.

[MN99] K. Mehlhorn and S. Näher. *The LEDA Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 1999. 1018 pages.

[MS01] K. Mehlhorn and S. Schirra. Exact computation with leda_real - theory and geometric applications. In G. Alefeld, J. Rohn, S. Rumpf, and T. Yamamoto, editors, *Symbolic Algebraic Methods and Verification Methods*. Springer Verlag, Vienna, 2001.

[Neu90] J. Neukirch. *Algebraische Zahlentheorie*. Springer-Verlag, 1990.

[Sch00] E. R. Scheinerman. When close enough is close enough. *American Mathematical Monthly*, 107:489–499, 2000.

[Yap97] Yap. Towards exact geometric computation. *CGTA: Computational Geometry: Theory and Applications*, 7, 1997.

[Yap99] C.K. Yap. *Fundamental Problems in Algorithmic Algebra*. Oxford University Press, 1999.

[YD95] C.K. Yap and T. Dube. The exact computation paradigm. In *Computing in Euclidean Geometry II*. World Scientific Press, 1995.