Geometric Algorithms
–
Exact Arithmetic, Filtering and Delayed Constructions

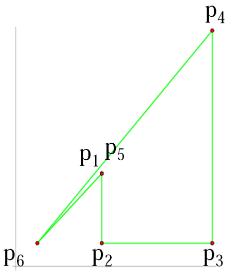
Michael Hemmer

TU Braunschweig

2014, Braunschweig

Outline

- Motivate Exact Computing
- Filtered Predicates
- Lazy Constructions
- CGAL Kernels



Classroom Examples ESA'04

Talk of Kurt Mehlhorn:

Classroom Examples
of Robustness Problems
in Geometric Computations

Recall Motivation

Geometric algorithms are a mix of

- Numerical computation
  (Point coordinates, distances, ...)
- Combinatorial techniques
  (Convex hull, Delaunay Triangulation, ...)

$\Rightarrow$ Small numerical errors can lead to:
Inconsistencies, infinite loops, crashes ...

Exact Geometric-Computation Paradigm

Exact Geometric-Computation Paradigm

Ensure correct control flow of algorithm by:

- Exact evaluation of geometric predicates
  - functions computing discrete results from numerical input
  - Orientation, Compare_xy, ...

Exact Geometric-Computation Paradigm

Ensure correct control flow of algorithm by:

- Exact evaluation of geometric predicates
  - functions computing discrete results from numerical input
  - Orientation, Compare_xy, ...
- Enforces exactness of geometric constructions
  - Intersection, Projection, ...
  - If there are any !

[C. Yap, T. Dubé, 1995]

The Easy Solution

Use exact multi-precision arithmetic

- integers, rational (e.g. GMP, CORE, LEDA)
- even algebraic numbers (e.g. CORE, LEDA)
- exact up to memory limit

Disadvantage: TOO SLOW

The Easy Solution

Use exact multi-precision arithmetic

- integers, rational (e.g. GMP, CORE, LEDA)
- even algebraic numbers (e.g. CORE, LEDA)
- exact up to memory limit

Disadvantage: TOO SLOW

No solution for transcendental numbers!

Find the Balance !

Requirements of the Real RAM model:

- arithmetic operations in constant time
- exact computation over the reals

The naive solutions:

- constant time floating point arithmetic that fails
- exact multi precision arithmetic that is too slow

The Answer are Filters

General filter scheme:

- try to compute a certified result fast (usually constant time)
- if certification fails may try another filter
- if nothing helps, use exact arithmetic

The hope:

- require only constant time for easy instances
- amortize cost for hard cases that use exact arithmetic

General Idea

General idea for filtered predicate:

- For expression E compute approximation $\tilde{E}$ and bound B, such that $|E - \tilde{E}| \leqslant B$ or equivalently:

$$E \in I = [\tilde{E} - B, \tilde{E} + B]$$

- If $0 \in I$ report *failure*, else return $sign(\tilde{E})$.

Recall: Floating Point Arithmetic

- A double float f uses 64 bits
  - 1 bit for the sign s
  - 52 bits for the mantissa $m = m_1 \ldots m_{52}$
  - 11 bits for the exponent $e = e_1 \ldots e_{11}$
- $f = -1^s \cdot (1 + \sum_{1 \leqslant i \leqslant 52} m_i 2^{-i}) \cdot 2^{e-2013}$, if $0 < e < 2^{11} - 1$
  $\ldots$
- for $a \in \mathbb{R}$, let $fl(a)$ be the closest float to a
  for $a \in \mathbb{Z}$: $|a - fl(a)| \leqslant \varepsilon |fl(a)|$, where $\varepsilon = 2^{-53}$
  for $o \in \{+, -, \times\}$: $|f_1 o f_2 - f_1 \tilde{o} f_2| \leqslant \varepsilon |f_1 \tilde{o} f_2|$
- floating point arithmetic is monotone
  e.g.: $b \leqslant c \;\Rightarrow\; a \oplus b \leqslant a \oplus c$

Computing B

For expression E define d_E and mes_E recursively:

E	$\tilde{E}$	mes_E	d_E
a, float	$fl(a)$	$\lvert fl(a)\rvert$	0
$a \in \mathbb{Z}$	$fl(a)$	$\lvert fl(a)\rvert$	1
$X + Y$	$\tilde{X} \oplus \tilde{Y}$	$\lvert\tilde{X}\rvert \oplus \lvert\tilde{Y}\rvert$	$1 + \max(d_X, d_Y)$
$X - Y$	$\tilde{X} \ominus \tilde{Y}$	$\lvert\tilde{X}\rvert \oplus \lvert\tilde{Y}\rvert$	$1 + \max(d_X, d_Y)$
$X \times Y$	$\tilde{X} \otimes \tilde{Y}$	$\lvert\tilde{X}\rvert \otimes \lvert\tilde{Y}\rvert$	$1 + d_X + d_Y$

Then B is defined as follows:

$$|E - \tilde{E}| \leqslant B = ((1 + \varepsilon)^{d_E} - 1) \cdot mes_E$$

[K. Mehlhorn, S.Näher; LEDA BOOK]

Proof

- Monotonicity of floats always guarantees: $\tilde{E} \leqslant mes_E$
- First two rows are trivial
- Lets proof invariant for addition

$$|\tilde{E} - E| \quad = \quad |(\tilde{X} \oplus \tilde{Y}) - (X + Y)|$$

Proof

- Monotonicity of floats always guarantees: $\tilde{E} \leqslant mes_E$
- First two rows are trivial
- Lets proof invariant for addition

$$
\begin{aligned}
|\tilde{E} - E| &= |(\tilde{X} \oplus \tilde{Y}) - (X + Y)| \\
&\leqslant |(\tilde{X} \oplus \tilde{Y}) - (\tilde{X} + \tilde{Y})| + |X - \tilde{X}| + |Y - \tilde{Y}|
\end{aligned}
$$

Proof

- Monotonicity of floats always guarantees: $\tilde{E} \leqslant mes_E$
- First two rows are trivial
- Lets proof invariant for addition

$$
\begin{aligned}
|\tilde{E} - E| &= |(\tilde{X} \oplus \tilde{Y}) - (X + Y)| \\
&\leqslant |(\tilde{X} \oplus \tilde{Y}) - (\tilde{X} + \tilde{Y})| + |X - \tilde{X}| + |Y - \tilde{Y}| \\
&\leqslant \varepsilon \cdot mes_E + |X - \tilde{X}| + |Y - \tilde{Y}|
\end{aligned}
$$

Proof

- Monotonicity of floats always guarantees: $\tilde{E} \leqslant mes_E$
- First two rows are trivial
- Lets proof invariant for addition

$$
\begin{aligned}
|\tilde{E} - E| &= |(\tilde{X} \oplus \tilde{Y}) - (X + Y)| \\
&\leqslant |(\tilde{X} \oplus \tilde{Y}) - (\tilde{X} + \tilde{Y})| + |X - \tilde{X}| + |Y - \tilde{Y}| \\
&\leqslant \varepsilon \cdot mes_E + |X - \tilde{X}| + |Y - \tilde{Y}| \\
&\leqslant \varepsilon \cdot mes_E + ((1+\varepsilon)^{d_X} - 1) mes_X + ((1+\varepsilon)^{d_Y} - 1) mes_Y
\end{aligned}
$$

Proof

- Monotonicity of floats always guarantees: $\tilde{E} \leqslant mes_E$
- First two rows are trivial
- Lets proof invariant for addition

$$
\begin{aligned}
|\tilde{E} - E| &= |(\tilde{X} \oplus \tilde{Y}) - (X + Y)| \\
&\leqslant |(\tilde{X} \oplus \tilde{Y}) - (\tilde{X} + \tilde{Y})| + |X - \tilde{X}| + |Y - \tilde{Y}| \\
&\leqslant \varepsilon \cdot mes_E + |X - \tilde{X}| + |Y - \tilde{Y}| \\
&\leqslant \varepsilon \cdot mes_E + ((1+\varepsilon)^{d_X} - 1) mes_X + ((1+\varepsilon)^{d_Y} - 1) mes_Y \\
&\leqslant \varepsilon \cdot mes_E + ((1+\varepsilon)^{max(d_X, d_Y)} - 1) \cdot mes_E
\end{aligned}
$$

Proof

- Monotonicity of floats always guarantees: $\tilde{E} \leqslant mes_E$
- First two rows are trivial
- Lets proof invariant for addition

$$
\begin{aligned}
|\tilde{E} - E| &= |(\tilde{X} \oplus \tilde{Y}) - (X + Y)| \\
&\leqslant |(\tilde{X} \oplus \tilde{Y}) - (\tilde{X} + \tilde{Y})| + |X - \tilde{X}| + |Y - \tilde{Y}| \\
&\leqslant \varepsilon \cdot mes_E + |X - \tilde{X}| + |Y - \tilde{Y}| \\
&\leqslant \varepsilon \cdot mes_E + ((1+\varepsilon)^{d_X} - 1)mes_X + ((1+\varepsilon)^{d_Y} - 1)mes_Y \\
&\leqslant \varepsilon \cdot mes_E + ((1+\varepsilon)^{max(d_X,d_Y)} - 1) \cdot mes_E \\
&\leqslant ((1+\varepsilon)^{1+max(d_X,d_Y)} - 1) \cdot mes_E = B
\end{aligned}
$$

Proof

- Monotonicity of floats always guarantees: $\tilde{E} \leqslant mes_E$
- First two rows are trivial
- Lets proof invariant for addition

$$
\begin{aligned}
|\tilde{E} - E| &= |(\tilde{X} \oplus \tilde{Y}) - (X + Y)| \\
&\leqslant |(\tilde{X} \oplus \tilde{Y}) - (\tilde{X} + \tilde{Y})| + |X - \tilde{X}| + |Y - \tilde{Y}| \\
&\leqslant \varepsilon \cdot mes_E + |X - \tilde{X}| + |Y - \tilde{Y}| \\
&\leqslant \varepsilon \cdot mes_E + ((1+\varepsilon)^{d_X} - 1) mes_X + ((1+\varepsilon)^{d_Y} - 1) mes_Y \\
&\leqslant \varepsilon \cdot mes_E + ((1+\varepsilon)^{max(d_X, d_Y)} - 1) \cdot mes_E \\
&\leqslant ((1+\varepsilon)^{1+max(d_X, d_Y)} - 1) \cdot mes_E = B
\end{aligned}
$$

Remark

In practice, one replaces

$$B = ((1+\varepsilon)^{d_E} - 1) \cdot mes_E$$

with

$$B = (\varepsilon \cdot d_E) \cdot mes_E,$$

as

$$((1+\varepsilon)^{d_E} - 1) \leqslant \varepsilon \cdot d_E, \text{ for } d_E < \sqrt{1/\varepsilon}.$$

Static and Semi-Static Filter

Static Filter:

- compute B once for all

Static and Semi-Static Filter

Static Filter:

- compute B once for all $\Rightarrow$ very fast

Static and Semi-Static Filter

Static Filter:

- compute B once for all $\Rightarrow$ very fast
- requires an assumption on the range of the input
- for many calls this assumption may be too large

Static and Semi-Static Filter

Static Filter:

- compute B once for all $\Rightarrow$ very fast
- requires an assumption on the range of the input
- for many calls this assumption may be too large

Almost-static filter:

- initialize B based on optimistic assumption
- adjust B if necessary

Static and Semi-Static Filter

Static Filter:

- compute B once for all $\Rightarrow$ very fast
- requires an assumption on the range of the input
- for many calls this assumption may be too large

Almost-static filter:

- initialize B based on optimistic assumption
- adjust B if necessary

Semi-static Filter:

- compute B depending on the input of each call

Static and Semi-Static Filter

Static Filter:

- compute B once for all $\Rightarrow$ very fast
- requires an assumption on the range of the input
- for many calls this assumption may be too large

Almost-static filter:

- initialize B based on optimistic assumption
- adjust B if necessary

Semi-static Filter:

- compute B depending on the input of each call
- still fast, since it essentially only doubles the costs

Static and Semi-Static Filter

Static Filter:

- compute B once for all $\Rightarrow$ very fast
- requires an assumption on the range of the input
- for many calls this assumption may be too large

Almost-static filter:

- initialize B based on optimistic assumption
- adjust B if necessary

Semi-static Filter:

- compute B depending on the input of each call
- still fast, since it essentially only doubles the costs

Combine Static and Semi-Static Filter

- Compute $\tilde{E}$

Combine Static and Semi-Static Filter

- Compute $\tilde{E}$
- try to certify using almost-static B

Combine Static and Semi-Static Filter

- Compute $\tilde{E}$
- try to certify using almost-static B
- otherwise compute semi-static B' and try to certify

Combine Static and Semi-Static Filter

- Compute $\tilde{E}$
- try to certify using almost-static B
- otherwise compute semi-static B' and try to certify

Disadvantage: Still considerable overestimation of error

Combine Static and Semi-Static Filter

- Compute $\tilde{E}$
- try to certify using almost-static B
- otherwise compute semi-static B' and try to certify

Disadvantage: Still considerable overestimation of error
Idea: Observe concrete error while computing $\tilde{E}$

Interval Arithmetic

For operands $x = [\underline{x}, \overline{x}]$ and $y = [\underline{y}, \overline{y}]$ set:

$$
\begin{aligned}
[x] + [y] &:= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\
[x] - [y] &:= [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \\
[x] \cdot [y] &:= [\min\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\}] \\
[x]/[y] &:= x \cdot [1/\overline{y}, 1/\underline{y}] \text{ if } 0 \notin [y] \\
[x]^{1/2} &:= [\underline{x}^{1/2}, \overline{x}^{1/2}] \text{ if } 0 \leqslant [x]
\end{aligned}
$$

Interval Arithmetic

For operands $x = [\underline{x}, \overline{x}]$ and $y = [\underline{y}, \overline{y}]$ set:

$$
\begin{aligned}
[x] + [y] &:= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\
[x] - [y] &:= [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \\
[x] \cdot [y] &:= [\min\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\}] \\
[x]/[y] &:= x \cdot [1/\overline{y}, 1/\underline{y}] \text{ if } 0 \notin [y] \\
[x]^{1/2} &:= [\underline{x}^{1/2}, \overline{x}^{1/2}] \text{ if } 0 \leqslant [x]
\end{aligned}
$$

Round in proper directions for floating point interval arithmetic

Interval Arithmetic

For operands $x = [\underline{x}, \overline{x}]$ and $y = [\underline{y}, \overline{y}]$ set:

$$
\begin{aligned}
[x] + [y] &:= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\
[x] - [y] &:= [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \\
[x] \cdot [y] &:= [\min\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\}, \max\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\}] \\
[x]/[y] &:= x \cdot [1/\overline{y}, 1/\underline{y}] \text{ if } 0 \notin [y] \\
[x]^{1/2} &:= [\underline{x}^{1/2}, \overline{x}^{1/2}] \text{ if } 0 \leqslant [x]
\end{aligned}
$$

Round in proper directions for floating point interval arithmetic

$\Rightarrow$ Inclusion Property

Dynamic Filter

- compute $\tilde{E} = [E]$ using floating point interval arithmetic
- result is certified if $0 \notin [E]$
- disadvantage: a bit slower than semi static filter
- advantage: better control of the error $\Rightarrow$ less filter failures

Remark: It is possible to avoid changes in rounding mode $\triangle, \bigtriangledown$, e.g.: $[x] + [y] := [- \triangle (-\underline{x} - \underline{y}), \triangle(\overline{x} + \overline{y})]$

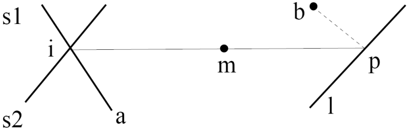
Filter Summary

Three main types:

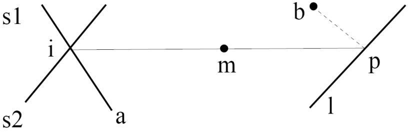
(almost) static filter	B is pre-computed as fast as floating point arithmetic very low accuracy
semi-static filter	B depends on input of each call 2 times slower than floating point still low accuracy
dynamic filter	compute $\tilde{E} = [E]$ with interval arithmetic 3-8 times slower than floating point high accuracy

What about cascaded geometric constructions ?

What about cascaded geometric constructions ?



What about cascaded geometric constructions ?



$$orientation\_3(a, m, b)?$$

Delayed / Lazy Constructions

Lazy Number Type

- always compute an interval
- also store history in a DAG*
- $\Rightarrow$ can compute exact if needed

*DAG = Directed Acyclic Graph

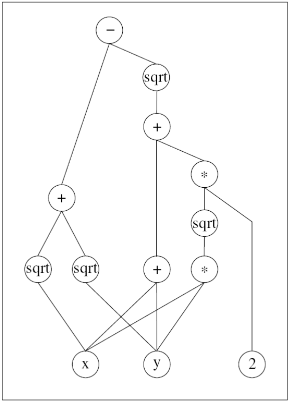


Fig. 3. Example DAG: $\sqrt{x} + \sqrt{y} - \sqrt{x + y + 2\sqrt{xy}}$.

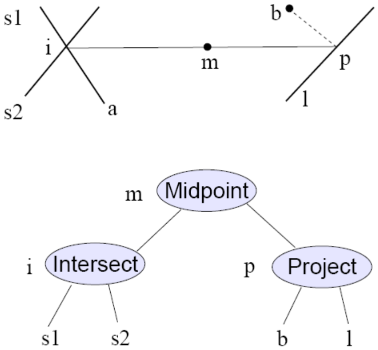
\+ : adaptive

\- : time lost in DAG management

\- : high memory consumption

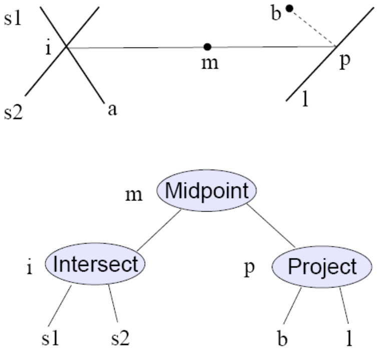
Lazy Kernel

- DAG nodes for constructions
- DAG nodes for predicates



Lazy Kernel

- ▶ DAG nodes for constructions
- ▶ DAG nodes for predicates
- \+ reduce management cost
- \+ reduce memory consumption
- \+ reduce rounding mode changes



(Simplified) Overview CGAL Kernel

- CGAL::Cartesian$<$double$>$: fast but not exact
- CGAL::Cartesian$< \mathbb{Q} >$: exact but slow
- CGAL::Filtered_kernel$< K >$
  - uses constructions of kernel K
  - dynamic filter for all predicates
  - semi-static filter for some predicates
  - predicates are exact

Predefined kernels:

- Exact_predicates_inexact_constructions_kernel
  = Filtered_kernel$<$ Cartesian$<$double$>>$
- Exact_predicates_exact_constructions_kernel
  $\simeq$ Lazy_exact_kernel$<$ Cartesian$< \mathbb{Q} >>$

Exact Expression Evaluation using Separation Bounds

`LEDA::real` and `CORE::Expr`

Allow:

- addition, substraction, mulitplication
- division
- k-th root
- algebraic numbers

Recall Lazy Evaluation

Lazy Number Type

- compute double interval first
- also store history in a DAG*
  ⇒ can compute exact if needed

*DAG = Directed Acyclic Graph

+ : adaptive
- : time lost in DAG management
- : high memory consumption

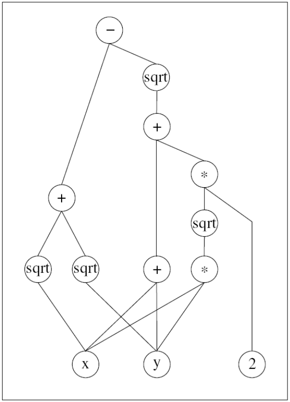


Fig. 3. Example DAG: $\sqrt{x} + \sqrt{y} - \sqrt{x + y + 2\sqrt{xy}}$.

Possible Variant:

Use multi-precision floating point intervals

- try with doubles first
- otherwise try with more precision if needed
- and so on ...

Possible Variant:

Use multi-precision floating point intervals

- try with doubles first
- otherwise try with more precision if needed
- and so on ...
- .. an expression that is zero leads to an infinite loop !

Possible Variant:

Use multi-precision floating point intervals

- try with doubles first
- otherwise try with more precision if needed
- and so on ...
- .. an expression that is zero leads to an infinite loop !

Simple solution:

- just stop at some high precision

Can we do better ?

Suppose the expression is just made of:

- integers (in the leaves of the DAG)
- operations: $\{+, -, *\}$
- Example: $E = 23 \cdot 60 \cdot 234 + 634 \cdot 234 \cdot 12 - 87633 \cdot 24$

Can we do better ?

Suppose the expression is just made of:

- integers (in the leaves of the DAG)
- operations: $\{+, -, *\}$
- Example: $E = 23 \cdot 60 \cdot 234 + 634 \cdot 234 \cdot 12 - 87633 \cdot 24$

Yes we can !

- The value of E must be an integer ($val(E) \in \mathbb{Z}$)

$\Rightarrow$ Compute interval I with increasing precision until:

  - $0 \notin I$: return $sign(I)$;
  - $I \cap \mathbb{Z} = \{0\}$: return 0;

Can we do better ?

Suppose the expression is just made of:

- integers (in the leaves of the DAG)
- operations: $\{+, -, *\}$
- Example: $E = 23 \cdot 60 \cdot 234 + 634 \cdot 234 \cdot 12 - 87633 \cdot 24$

Or in other words:

- 0 is separated from all other possible values by 1, the separation bound of E, $sep(E) = 1$
- The process stops once the width of I is less than 1, $\Delta(I) < 1 = sep(E)$

Extend set of operations by $\sqrt[k]{\cdot}$

Definition

An algebraic integer is a root of a polynomial with integer coefficients and leading coefficient one.

It follows that this is also the case for its minimal polynomial.

Example: $X^2 - 2 = (X - \sqrt{2})(X + \sqrt{2})$ or $X^k - a$

Remark I: An integer is an algebraic integer.

Remark II: Algebraic integers are closed under $op \in \{+, -, *\}$

For algebraic integers α and β consider the minimal polynomials:

- $P_A(X) = X^n + \prod_{i=0}^{n-1} a_i X^i = \prod_{i=1}^{n} (X - \alpha_i) \in \mathbb{Z}[X]$
- $P_B(X) = X^m + \prod_{j=0}^{m-1} b_j X^i = \prod_{j=1}^{m} (X - \beta_j) \in \mathbb{Z}[X]$,

where α is a root of $P_A(X)$ and β is a root of $P_B(X)$.

The result of α *op* β, with *op* $\in \{+, -, *\}$ is the root of

$$P_{A\ op\ B}(X) = \prod_{i=1}^{n} \prod_{j=1}^{m} (X - (\alpha_i\ op\ \beta_j)) \in \mathbb{Z}[X],$$

which is a monic polynomial of degree $n \cdot m$.

(*) The α_i are the algebraic conjugates of α.

(**) The degree of $P_A(X)$ is the algebraic degree of α.

Lemma

Let α be an algebraic integer and let deg(α) be its algebraic degree. If $U > 0$ is an upper bound on the absolute values of all algebraic conjugates of α, then

$$|\alpha| \geqslant 1/U^{deg(\alpha)-1}.$$

Proof.

Consider the minimal polynomial $P_\alpha = \prod_{i=1}^{n}(X - \alpha_i) \in \mathbb{Z}[X]$. The constant coefficient is $\prod_{i=1}^{n} \alpha_i$ which is at least one, since it is in $\mathbb{Z}$.

$\Rightarrow |\alpha| \cdot U^{deg(\alpha)-1} \geqslant 1$ □

We obtain algebraic integers by expressions that are made of:

- integers (in the leaves of the DAG)
- operations: $\{+, -, *, \sqrt[k]{\cdot}\}$

An upper bound on the

- algebraic degree $D(E)$ is the product of all occurring k.
- the bound $U(E)$ on absolute value of the algebraic conjugates is given by the following recursive table:

E	$U(E)$	D(E)
$n \in \mathbb{Z}$	$\|n\|$	1
$X \pm Y$	$U(X) + U(Y)$	$D(X) \cdot D(Y)$
$X \cdot Y$	$U(X) \cdot U(Y)$	$D(X) \cdot D(Y)$
$\sqrt[k]{X}$	$\sqrt[k]{U(X)}$	$k \cdot D(x)$

If $\tilde{E} < 1/U(E)^{D(E)-1} \Rightarrow E = 0$

Introducing devisions

Devision destroys algebraic integer property !
$\Rightarrow$ Treat numerator and denominator separately

$$\frac{A_n}{A_d} \pm \frac{B_n}{B_d} \Rightarrow \frac{A_n B_d \pm B_n A_d}{A_d B_d}, \ldots, \sqrt[k]{\frac{A_n}{A_d}} \Rightarrow \frac{\sqrt[k]{A_n A_d^{k-1}}}{A_d}$$

we obtain the following table:

E	$U_n(E)$	$U_d(E)$
$n \in \mathbb{Z}$	$\lvert n \rvert$	1
$X \pm Y$	$U_n(X)U_d(Y) + U_n(Y)U_d(X)$	$U_d(X)U_d(Y)$
$X \cdot Y$	$U_n(X) \cdot U_n(Y)$	$U_d(X) \cdot U_d(Y)$
X/Y	$U_n(X) \cdot U_d(Y)$	$U_d(X) \cdot U_n(Y)$
$\sqrt[k]{X}$	$\sqrt[k]{U_n(X)U_d^{k-1}}$	$U_d(X)$

If $|\tilde{E}| \cdot U_d(E) < 1/U_n(E)^{D(E)-1} \Rightarrow E = 0$

Final Remarks

- `leda::real` and `CORE::Expr` are essentially the same
- both also allow to define a value as the root of a polynomial

Final Remarks

- `leda::real` and `CORE::Expr` are essentially the same
- both also allow to define a value as the root of a polynomial

Advantages & Disadvantages

+ : Allow cascaded constructions
+ : Lazy evaluation
- : time lost in DAG management
- : high memory consumption

Final Remarks

- `leda::real` and `CORE::Expr` are essentially the same
- both also allow to define a value as the root of a polynomial

Advantages & Disadvantages

+ : Allow cascaded constructions
+ : Lazy evaluation
- : time lost in DAG management
- : high memory consumption

General guidelines:

- never use them as your main type
- try to produce balanced expressions
- try to simplify expressions
- do you really need to use $\sqrt{\cdot}$?
- avoid unnecessary test against zero