`Real/Expr` : Implementation of an Exact Computation Package

Kouji Ouchi

January 16, 1997

Abstract

The `Real/Expr` package is a `C++` project to support the precision-driven approach to exact computation of geometric algorithms. The package is built on top of the class `Real` that encompasses a variety of numerical representations. The class `Expr` captures a set of algebraic expressions on which any comparison can be done precisely.

The software libraries described here are available via the Web page `http://simulation.nyu.edu/projects/exact/`.

1 Introduction

Robust implementation of geometric algorithms is difficult to achieve. The main problem arises from the use of *fixed-precision arithmetic* such as machine floating-point arithmetic. To overcome this, *the exact computation method* which uses arbitrary-precision arithmetic has been proposed. However, its naive interpretation, namely, computing all numerical quantities exactly, is too inefficient. Notice that what needs to be exact is a combinatorial structure; but the numerical quantities associated with the combinatorial structure need not be exact. This observation suggests another interpretation of the exact computation, *precision-driven computation*, where numerical quantities will be computed to be precise enough so that decisions for the related combinatorial structure can be made exactly. As a tool for this approach of the exact computation, we would like to introduce the `Real/Expr` package in which users can perform the precision-driven computation over algebraic expressions.

Geometric algorithms characteristically involve geometric data structures. By a geometric data structure, we mean *a combinatorial data structure* together with *numerical quantities*. Moreover, there are implicit consistency constraints governing the relation between the combinatorial structure and its associated numerical quantities. This means that perturbing the numerical values without taking into account the combinatorial structure can lead to qualitatively different or inconsistent states, which often result in catastrophic errors in algorithms.

Many researchers have devised methods to address non-robustness problems within the fixed-precision arithmetic. We believe that non-robustness in geometric algorithms is inherent when one is committed to fixed-precision, and the best general policy for attacking non-robustness is the exact computation, to compute geometrical data structures exactly.

Exact computation has *a naive interpretation*, namely, to compute *every numerical quantity exactly*. This surely guarantees the robustness of algorithms. However, it is too inefficient in general because occasionally huge numerical quantities must be dealt with.

We will take another approach, where *computing exactly* is taken to mean the combinatorial structure must be mathematically correct, but the associated numerical quantities may be approximations that are consistent with the combinatorial structure. This interpretation of exact computation could be realized with much less expensive cost than the naive one.

Now, we compare fixed-precision arithmetic and arbitrary-precision arithmetic.

In fixed-precision arithmetic (e.g. machine floating-point arithmetic), all the numerical objects are limited to some universal fixed-precision. The arithmetic operations are fast, and often there are hardware supports. Since the size of an object is fixed, the memory allocation for a brand new object can be statically done.

In arbitrary-precision arithmetic, there is no limitation for precisions of numerical objects (officially, of course. In practice, there is a limitation based on the limited available resources, etc). The arithmetic operations could be done without causing overflow or underflow, but the speed is slow. From the view point of complexity, if the size of objects becomes larger, the cost for the operations grows at least proportionally to the size of objects (usually, much worse). At execution time, the memory allocation for a newly constructed object is a much more serious problem: since the size of an object is unknown, the memory allocation for the object should be done dynamically.

We must use arbitrary-precision arithmetic. So, somehow we would like to limit the growth of the size of numerical objects. To achieve this goal, we introduce an arbitrary-precision floating-number representation in the following format:

$$(mantissa \pm error) \times BASE^{exponent},$$

where *mantissa* and *exponent* are integers of arbitrary length and *error* is a non-negative integer. A triple $\langle mantissa, error, exponent\rangle$ is interpreted as any (real algebraic) number in the interval

$$\left[(mantissa - error) \times BASE^{exponent}, (mantissa + error) \times BASE^{exponent}\right].$$

Obviously, any (fixed-precision) floating-point number or an integer of arbitrary length can be represented by some arbitrary-precision floating-point number. But, we need more: to perform the exact computation, we must deal with rational numbers, or much more generally, algebraic numbers. Here, an algebraic number is defined to be a root of some integer coefficient polynomial. Any rational or algebraic number can also be represented by our arbitrary-precision floating-point number representation with an error component $error \geq 0$. Note that the correspondence between rational (or algebraic) numbers and our arbitrary-precision floating-point representations is not bijective. In fact, any arbitrary-precision floating-point representation with non-zero *error* contains infinitely many rational (or algebraic) numbers.

Using this arbitrary-precision floating-point number representation, we could realize our interpretation of exact computation with reduced size of numerical objects. For example, to determine the sign of a non-zero rational number, we simply approximate it in terms of our arbitrary-precision floating-point representation which does not contain 0. If the rational number has a numerator and a denominator of length $\mathcal{O}(n)$ and $\mathcal{O}(d)$ bits, respectively, then we need consider a arbitrary-precision floating-point number whose *mantissa* is of length 1 bit and *exponent* is of length $\mathcal{O}(\lg|n-d|)$ bits.

Furthermore, to minimize inefficiency, we restrict the range of *error* so that it fits some fixed-precision number representation (e.g. machine unsigned long integer). Whenever an object happens to have *error* which is out of range, we truncate *error* as well as *mantisaa* so that *error* will fall into the standard range. This way, we prevent *mantissa* from growing rapidly.

Given this arbitrary-precision floating-point number representation, we now introduce our `Expr` package which embodies our interpretation of exact computation: "precision-driven computation". The `Expr` package captures a set of algebraic expressions involving $+$, $-$, $\cdot$, $/$ and $\sqrt{\ }$ over rational numbers. An expression is expressed as a rooted DAG (directed acyclic graph), and maintains an approximation of the expression. When the precision of the root is specified, we recursively drive the precision of each of the children nodes, so that if the subexpression rooted at the child node is approximated to that precision then we could get the approximation of the root to the required precision. For these approximations, we use our arbitrary-precision floating-point numbers, and thus, `Expr` package returns the interval to which the value of the expression belongs while the width of the interval is controlled by the specified precision. The precision could be set explicitly by users, or internal function calls such as calls to the equality operators.

Many fundamental predicates of geometric algorithms are expressed by algebraic expressions. For example, "P is left of the directed line segment $\overrightarrow{QR}$" is expressed as a sign of the signed volume (the determinant of the matrix whose entries are coordinates of P, Q and R and 1's) of $\triangle PQR$. For these predicates, our `Expr` package is best applicable. We construct the expression for the signed volume, and approximate its value precisely enough so that we can determine its sign; but we never compute the value itself.

In this paper, we describe the design, the algorithms, and the implementation techniques of our package.

2 Overview

In this section, we introduce the basic elements in our `Real/Expr` package and raise the issues to be addressed in this paper.

The package is written in the `C++` language, and is realized as a set of `C++` class libraries. There are three major classes: the class `Expr`, the class `Real` and the class `BigFloat`.

2.1 The Class `Expr`

The class `Expr` captures a set of algebraic expressions.

Formally, an instance of `Expr` is a rooted DAG where each leaf can store some value in $\mathbb{Q}$ and each internal node represents one of the operations $+$, (unary and binary) $-$, $\cdot$, $/$ and $\sqrt{\ }$. If every leaf of the tree rooted at e stores a value in $\mathbb{Q}$ then e can be viewed as an element of a real algebraically closed field $\mathbb{D}$ which contains $\mathbb{Q}$. We call this element in $\mathbb{D}$ *the exact value* of e. Note that the exact value of e is not a data member of `Expr`.

Each instance e of `Expr` maintains some real value x and precision p such that x approximates the exact value of e to precision p. The precision p is set explicitly by the user, or implicitly by the package. For example, the comparison operation $e > 0$ will set the necessary precision p to determine the sign of the exact value of e. To get an approximation x of e to p, we drive the precision top-down from the node e to its descendent leaves, and collect approximations bottom-up from leaves to the node e. In this case, the precision of an instance is set by its parent node. Setting the necessary precisions is the main algorithmic issue of `Expr`.

Another important issue is the semantics of `Expr`. We would like users to use our package as a tool for symbolic computation. For this reason, we define the special semantics for assignments that is different from the standard grammar of `C++`. Since the "pass-by-value" rule cannot be taken, the realization of our scheme is a non-trivial issue in the implementation of `Expr`.

2.2 The Class Real

Instances of the class Real are used for the approximate values of instances of Expr and the exact values in leaves of some Expr trees.

The class Real encompasses a variety of number representations: machine integers (int, long), machine (double-precise) floating-point numbers (double), integers of arbitrary length (BigInt) and rational numbers (Rational), as well as our arbitrary-precision floating-point representation BigFloat. Currently, we use GNU's Integer and Rational for BigInt and Rational, respectively.

The main algorithmic issue here is how to define the operations $+$, (unary and binary) $-$, $\cdot$, $/$ and $\sqrt{\ }$. More specifically, the way to determine the type of the result of binary operators applied to arguments of different types and the way to define operations without causing overflow or underflow become important topics.

The implementation issue is how to realize the class that has an ability to capture various types. We would like to implement Real operations in an object-oriented way, that is, operations are implemented so that, given specific operand(s), the compiler can choose the correct algorithm depending on the type(s) of the operand(s).

2.3 The Class BigFloat

The class BigFloat realizes arbitrary-precision floating-point number representation with the error component. Instances of BigFloat are intended to approximate real numbers. If an instance of BigFloat has a non-zero error then it is actually an interval and approximates any real number which belongs to that interval.

There are two algorithmic issues for BigFloat.

One is the design of an approximation algorithm: given a rational number and precision, find a BigFloat which approximates the rational number to that precision.

The other is the design of the arithmetic operations and the function $\sqrt{\ }$ for BigFloat. Since an instance of BigFloat represents an interval, the operations must be defined so that they are valid for any real number in that interval.

The class BigFloat has a member *mantissa* which is declared to be an integer of arbitrary length. Accessing *mantissa* slows down the execution speed of the package seriously. We show how the use of the "letter-envelope" technique helps to reduce unnecessarily accesses to the *mantissa* components.

3 BigFloat

In this section, we describe our arbitrary-precision floating-point package BigFloat implemented as a class library in C++. In addition to the standard libraries of C++, we assume that we have a class library of integers of arbitrary length such as GNU's Integer.

Some basic ideas are described in [DY93].

3.1 Definition

Fix any positive integer c and let $B = 2^c$. For the implementation, it is convenient to set c as follows; if the largest `unsigned long` is $2^L - 1$ then $c = \left\lfloor \frac{L}{2} \right\rfloor - 2$, e.g., $L = 32 \Rightarrow c = 14$.

Each `BigFloat` number is a triple $\langle m, err, exp \rangle$ where

- mantissa $m \in \mathbb{Z} = \{0, \pm 1, \pm 2, \ldots\}$,
- error $err \in \mathbb{N} = \{0, 1, 2, \ldots\}$,
- exponent $exp \in \mathbb{Z}$.

We say the `BigFloat` $\langle m, err, exp \rangle$ is *error-normalized* (or simply *normalized*) if

$$err \in \{0, 1, \ldots, 4B - 1\}.$$

Unless otherwise specified, we assume `BigFloat` numbers are normalized.

`BigFloat` numbers are intended to be approximations for real numbers. A real number X is said to *belong to* a `BigFloat` number $\langle m, err, exp \rangle$ if

$$X \in [(m - err)B^{exp}, (m + err)B^{exp}].$$

Let $(r, a) \in \mathbb{N} \times \mathbb{Z}$. A `BigFloat` $\langle m, err, exp \rangle$ is said to have *an error-bound* $[r, a]$ if

$$\begin{array}{rcl} err & \leq & |m|\, 2^{-r} \\ & \text{OR} & \\ err B^{exp} & \leq & 2^{-a}. \end{array}$$

3.2 Approximation

Let X be a real number and $(r, a) \in \mathbb{N} \times \mathbb{Z}$.

We say a real number $\widehat{X}$ *approximates* X *to precision* $[r, a]$ and write

$$\widehat{X} \cong X[r, a]$$

if

$$\left| X - \widehat{X} \right| \leq \max\left\{ |X|\, 2^{-r},\ 2^{-a} \right\}.$$

Intuitively, this notation suggests that $\widehat{X}$ is the "output" for input X and $[r, a]$. Here, r and a specify relative and absolute bounds on the error.

We say a `BigFloat` $x = \langle m_x, err_x, exp_x \rangle$ *approximates* X *to precision* $[r, a]$ and write

$$x \cong X[r, a]$$

if X belongs to x and

$$err_x B^{exp_x} \leq \max\left\{ |X|\, 2^{-r},\ 2^{-a} \right\}.$$

If x approximates X then $m_x B^{exp_x} \cong X[r, a]$, i.e.,

$$|X - m_x B^{exp_x}| \leq \max\left\{ |X|\, 2^{-r},\ 2^{-a} \right\}.$$

3.2.1 Approximation Algorithm

Given $R \in \mathbb{Q}$ and $(r, a) \in \mathbb{N} \times \mathbb{Z}$, we would like to compute a `BigFloat` $x = \langle m_x, err_x, exp_x \rangle$ with the error-bound $[r, a]$ such that R belongs to x. Suppose

$$R = \frac{N}{D}$$

where $(N, D) \in \mathbb{Z} \times \mathbb{Z}_{\neq 0}$. Then, x will be computed by the function $div\,(N, D, r, a)$.

We now describe the algorithm for $div\,(N, D, r, a)$.

If $N = 0$ then it returns the `BigFloat` *zero*:

$$\begin{aligned} m_x &\leftarrow 0 \\ err_x &\leftarrow 0 \\ exp_x &\leftarrow 0. \end{aligned}$$

Now, assume $N \neq 0$. Basically, the mantissa m_x is computed by performing the division $|N| / |D|$ (up to the sign of ND denoted $\operatorname{sgn}(ND)$). The mantissa m_x is an integer which must be long enough to have the required error-bound. Thus, we actually shift $|N|$ left or right and invoke the integer division so that we may control the length of the quotient. Shifting must be done chunk by chunk, that is, c bits by c bits.

First, suppose $|N|$ is shifted left $s \geq 0$ chunks. Then, the integer division $|N| B^s / |D|$ yields the equality

$$|N| B^s \;=\; \left\lfloor \frac{|N| B^s}{|D|} \right\rfloor |D| + remainder \quad \text{where } 0 \leq remainder < |D|.$$

Thus

$$\left\lfloor \frac{|N| B^s}{|D|} \right\rfloor B^{-s} \;\leq\; \left|\frac{N}{D}\right| = \left(\left\lfloor \frac{|N| B^s}{|D|} \right\rfloor + \frac{remainder}{|D|} \right) B^{-s} \;<\; \left(\left\lfloor \frac{|N| B^s}{|D|} \right\rfloor + 1 \right) B^{-s}.$$

Note $remainder = 0$ iff $|D|$ divides $|N| B^s$. Hence, we set

$$\begin{aligned} m_x &\leftarrow \operatorname{sgn}(ND) \left\lfloor \frac{|N| B^s}{|D|} \right\rfloor \\ err_x &\leftarrow \begin{cases} 0 & \text{if } |D| \text{ divides } |N| B^s \\ 1 & \text{otherwise} \end{cases} \\ exp_x &\leftarrow -s. \end{aligned}$$

Next, suppose $|N|$ is shifted right $t > 0$ chunks. In this case, $|N|$ is actually truncated and the the quotient $\left\lfloor \frac{|N|}{|D| B^t} \right\rfloor$ of the integer division $\left\lfloor \frac{|N|}{B^t} \right\rfloor / |D|$ satisfies

$$\left\lfloor \frac{|N|}{|D| B^t} \right\rfloor \;\leq\; \frac{|N|}{|D| B^t} \;<\; \left\lfloor \frac{|N|}{|D| B^t} \right\rfloor + 1.$$

Thus

$$\left\lfloor \frac{|N|}{|D| B^t} \right\rfloor B^t \;\leq\; \left|\frac{N}{D}\right| \;<\; \left(\left\lfloor \frac{|N|}{|D| B^t} \right\rfloor + 1 \right) B^t.$$

Hence, we set

$$\begin{aligned} m_x &\leftarrow \operatorname{sgn}(ND)\left\lfloor \frac{|N|}{|D|B^t} \right\rfloor \\ err_x &\leftarrow 1 \\ exp_x &\leftarrow t. \end{aligned}$$

Note err_x is always set to be 1 in this case. We could have a slightly more precise algorithm if the integer division $\left\lfloor \frac{|N|}{B^t} \right\rfloor / |D|$ (i.e. truncate $|N|$ by t chunks before dividing by $|D|$) is replaced by the integer division $|N| / |D| B^t$ (i.e. divide $|N|$ by $|D| B^t$). Then, like the previous case, err_x might be set to be 0 if $|D| B^t$ divides $|N|$. Unfortunately, this is inefficient, since we must perform the integer division with larger operands.

It is convenient to put two cases together. We are able to do so by setting $s = -t$ in the first case. Therefore, we set

$$\begin{aligned} m_x &\leftarrow \operatorname{sgn}(ND)\left\lfloor \frac{|N|}{|D|B^t} \right\rfloor \\ err_x &\leftarrow \begin{cases} 0 & \text{if } t \le 0 \text{ and } |D| \text{ divides } |N| B^{-t} \\ 1 & \text{otherwise} \end{cases} \\ exp_x &\leftarrow t. \end{aligned}$$

Now, we need to determine the value of t (may or may not be non-negative) which satisfies

$$err_x \le |m_x| 2^{-r} \quad \text{OR} \quad err_x\, 2^{ct} \le 2^{-a}. \tag{1}$$

Since $err_x = 0$ OR 1, a sufficient condition for (1) is

$$1 \le \left\lfloor \frac{|N|}{|D|2^{ct}} \right\rfloor 2^{-r} \quad \text{OR} \quad 2^{ct} \le 2^{-a}.$$

We claim that it suffices to set

$$t \leftarrow \max\left\{ \left\lfloor \frac{-r + \lfloor \lg|N| \rfloor - \lfloor \lg|D| \rfloor - 1}{c} \right\rfloor, \left\lfloor \frac{-a}{c} \right\rfloor \right\}. \tag{2}$$

To see that (2) is correct, first, suppose $t = \left\lfloor \frac{-r+\lfloor \lg|N| \rfloor - \lfloor \lg|D| \rfloor -1}{c} \right\rfloor$. Then

$$\begin{aligned} ct &\le -r + \lfloor \lg|N| \rfloor - \lfloor \lg|D| \rfloor - 1 \\ &\le -r + \lfloor \lg|N| - \lg|D| \rfloor \\ &= -r + \left\lfloor \lg\left|\frac{N}{D}\right| \right\rfloor. \end{aligned}$$

Since $r \ge 0$, $ct \le \left\lfloor \lg\left|\frac{N}{D}\right| \right\rfloor \le \lg\left|\frac{N}{D}\right|$ or equivalently

$$\frac{|N|}{|D|2^{ct}} \ge 1.$$

It is not hard to show that $\lg\lfloor x \rfloor \ge \lfloor \lg x \rfloor \quad \forall x \ge 1$. Using this fact,

$$r \le \left\lfloor \lg\left|\frac{N}{D}\right| \right\rfloor - ct = \left\lfloor \lg \frac{|N|}{|D|2^{ct}} \right\rfloor \le \lg\left\lfloor \frac{|N|}{|D|2^{ct}} \right\rfloor$$

or equivalently

$$1 \le \left\lfloor \frac{|N|}{|D| 2^{ct}} \right\rfloor 2^{-r}.$$

Next, suppose $t = \left\lfloor \frac{-a}{c} \right\rfloor$. Immediately

$$2^{ct} \le 2^{-a}.$$

3.2.2 Properties

Fix $R \in \mathbb{Q}$ and $(r, a) \in \mathbb{N} \times \mathbb{Z}$. Let $x = \langle m_x, err_x, exp_x \rangle$ be the `BigFloat` computed by our approximation algorithm on input R and $[r, a]$.

Proposition 1

1.

$$|m_x| B^{exp_x} \le |R|.$$

2.

$$x \cong R[r, a].$$

In particular, $m_x B^{exp_x} \cong R[r,a]$, *i.e.,* $|R - m_x B^{exp_x}| \le \max\{|R| 2^{-r}, 2^{-a}\}$.

Proof.

1. The claim is obvious, since
$$|m_x| = \left\lfloor \frac{|R|}{B^{exp_x}} \right\rfloor.$$

2. By definition
$$err_x B^{exp_x} \le \max\left\{ |m_x| B^{exp_x} 2^{-r}, 2^{-a} \right\},$$
and we have just seen $|m_x| B^{exp_x} \le |R|$.

Q.E.D.

Proposition 2 *If* $m_x \neq 0$ *then*

$$\lfloor \lg |m_x| \rfloor + c \cdot exp_x = \lfloor \lg |R| \rfloor.$$

Proof. If $m_x \neq 0$ then $R \neq 0$. Thus, the right hand side of the formula is well-defined, and $|m_x| = \left\lfloor \frac{|R|}{2^{c \cdot exp_x}} \right\rfloor$. Hence

$$\lfloor \lg |m_x| \rfloor \;=\; \left\lfloor \lg \left\lfloor \frac{|R|}{2^{c \cdot exp_x}} \right\rfloor \right\rfloor \;=\; \left\lfloor \lg \frac{|R|}{2^{c \cdot exp_x}} \right\rfloor \;=\; \lfloor \lg |R| \rfloor - c \cdot exp_x$$

where the second equality holds since $\frac{|R|}{2^{c \cdot exp_x}} \ge 1$. **Q.E.D.**

3.3 Error-Normalization

To keep the representation efficient, we would like to normalize our `BigFloat` number, i.e., maintain the error err in the range $0 \le err < 4B$.

Let $\langle m', err', exp' \rangle$ be a `BigFloat` not necessarily normalized. We could define *the normalization of* $\langle m', err', exp' \rangle$ to be a `BigFloat` $\langle m, err, exp \rangle$ which satisfies

(a) $0 \le err < 4B$,

(b) $[(m - err)B^{exp}, (m + err)B^{exp}] \supseteq \left[(m' - err')B^{exp'}, (m' + err')B^{exp'}\right]$,

and

(c') $errB^{exp}$ is minimized subject to (a) and (b).

The condition (b) states that any real number which belongs to the original `BigFloat` number must also belong to the normalized `BigFloat` number.

Since (c') is somewhat hard to guarantee, we shall officially replace it by:

(c) $errB^{exp} \le 2\, err' B^{exp'}$.

3.3.1 Error-Normalization Algorithm

If $err' < 4B$ then there is nothing to do. Otherwise, let $f \ge 1$ be the integer which satisfies

$$2B^f \le err' < 2B^{f+1}$$

or equivalently, $f = \left\lfloor \frac{\lfloor \lg err' \rfloor - 1}{c} \right\rfloor$. Set

$$\begin{aligned} m &\leftarrow \operatorname{sgn}(m') \left\lfloor \frac{|m'|}{B^f} \right\rfloor \\ err &\leftarrow \left\lfloor \frac{err'}{B^f} \right\rfloor + 2 \\ exp &\leftarrow exp' + f. \end{aligned}$$

The requirement (a) is satisfied, since

$$\left\lfloor \frac{err'}{B^f} \right\rfloor + 2 \quad < \quad 2B + 2 \quad < \quad 4B.$$

For the requirement (b), if $m' \ge 0$ then

$$\begin{aligned} (m - err)B^{exp} &< \left(\left\lfloor \frac{|m'|}{B^f} \right\rfloor - \left(\left\lfloor \frac{err'}{B^f} \right\rfloor + 1 \right) \right) B^{exp'+f} \\ &< (m' - err')B^{exp'} \end{aligned}$$

and

$$\begin{aligned} (m + err)B^{exp} &= \left(\left(\left\lfloor \frac{|m'|}{B^f} \right\rfloor + 1 \right) + \left(\left\lfloor \frac{err'}{B^f} \right\rfloor + 1 \right) \right) B^{exp'+f} \\ &> (m' + err')B^{exp'}. \end{aligned}$$

If $m' < 0$ then

$$\begin{aligned}(m - err)B^{exp} &= \left(-\left(\left\lfloor\frac{|m'|}{B^f}\right\rfloor + 1\right) - \left(\left\lfloor\frac{err'}{B^f}\right\rfloor + 1\right)\right) B^{exp'+f} \\ &< (m' - err')B^{exp'}\end{aligned}$$

and

$$\begin{aligned}(m + err)B^{exp} &> \left(\left(-\left\lfloor\frac{|m'|}{B^f}\right\rfloor\right) + \left(\left\lfloor\frac{err'}{B^f}\right\rfloor + 1\right)\right) B^{exp'+f} \\ &> (m' + err')B^{exp'}.\end{aligned}$$

Finally, the requirement (c) is satisfied, because when $err' \geq 4B$

$$\begin{aligned}err B^{exp} &= \left(\left\lfloor\frac{err'}{B^f}\right\rfloor + 2\right) B^{exp'+f} \\ &\leq \left(err' + 2B^f\right) B^{exp'} \\ &\leq 2\, err' B^{exp'}.\end{aligned}$$

3.4 Unary Minus Operator

Let $x = \langle m_x, err_x, exp_x\rangle$ be a `BigFloat`. Define $-x$ to be a `BigFloat` $y = \langle m_y, err_y, exp_y\rangle$ such that if a real X belongs to x then $-X$ belongs to y.

Set

$$\begin{aligned}m_y &\leftarrow -m_x \\ err_y &\leftarrow err_x \\ exp_y &\leftarrow exp_x.\end{aligned}$$

The correctness is obvious.

Note y does not need to be normalized, since $err_y < 4B$.

3.5 Arithmetic Operators

In the following subsections, we describe how arithmetic operations are done over `BigFloat` numbers.

Let $x = \langle m_x, err_x, exp_x\rangle$ and $y = \langle m_y, err_y, exp_y\rangle$ be `BigFloat`. For $@ \in \{+, -, \cdot, /\}$, we would like to define $x@y$ to be a `BigFloat` $z = \langle m_z, err_z, exp_z\rangle$ which satisfies

(a) if a real X belongs to x and a real Y belongs to y then $X@Y$ belongs to z,

and

(b') $err_z B^{exp_z}$ is minimized subject to (a).

As (b') is difficult to ensure, our algorithms will only guarantee upper bounds for $err_z B^{exp_z}$.

In our algorithms, we first define a `BigFloat` $z' = \langle m', err', exp'\rangle$ whose normalization would be z.

3.5.1 Addition and Subtraction

We would like to compute $z = x \pm y$. By symmetry, we may assume $exp_x \geq exp_y$.

1. If $exp_x = exp_y$ then

$$\begin{aligned} m'_z &\leftarrow m_x \pm m_y \\ err'_z &\leftarrow err_x + err_y \\ exp'_z &\leftarrow exp_x. \end{aligned}$$

   The correctness is obvious.

   If $err_x = 0$ or $err_y = 0$ then $err'_z < 4B$ and z' does not need to be normalized.

2. If $exp_x > exp_y$ and $err_x = 0$ then we shift m_x left by $exp_x - exp_y$ chunks and add it to m_y to get m'_z so that we may avoid throwing away the error-free bits of m_y:

$$\begin{aligned} m'_z &\leftarrow m_x B^{exp_x - exp_y} \pm m_y \\ err'_z &\leftarrow err_y \\ exp'_z &\leftarrow exp_y. \end{aligned}$$

   The correctness for this case is also obvious.

   Since $err'_z < 4B$, z' does not need to be normalized.

3. If $exp_x > exp_y$ and $err_x > 0$ then err_x "hides" some insignificant bits of m_y and err_y. We shift m_y right by $exp_x - exp_y$ chunks and add it to m_x to get m'_z, and add 5 to err_x to get err'_z; 1 for covering the truncated bits of m_y and 4 for err_y:

$$\begin{aligned} m'_z &\leftarrow m_x \pm \operatorname{sgn}(m_y) \left\lfloor \frac{|m_y|}{B^{expx - expy}} \right\rfloor \\ err'_z &\leftarrow err_x + 5 \\ exp'_z &\leftarrow exp_x. \end{aligned}$$

   If $m_y \geq 0$ then the addition and the subtraction are correct, since

$$\begin{aligned} &(m'_z - err'_z)B^{exp'_z} \\ &= \left(\left(m_x \pm \left\lfloor \frac{|m_y|}{B^{expx-expy}} \right\rfloor\right) - (err_x + 5)\right) B^{exp_x} \\ &= (m_x - err_x)B^{exp_x} \pm \left(\left\lfloor \frac{|m_y|}{B^{expx-expy}} \right\rfloor \mp 5\right) B^{exp_x - exp_y} B^{exp_y} \\ &< (m_x - err_x)B^{exp_x} \pm (m_y \mp err_y)B^{exp_y} \end{aligned}$$

   and

$$\begin{aligned} &(m'_z + err'_z)B^{exp'_z} \\ &= \left(\left(m_x \pm \left\lfloor \frac{|m_y|}{B^{expx-expy}} \right\rfloor\right) + (err_x + 5)\right) B^{exp_x} \\ &= (m_x + err_x)B^{exp_x} \pm \left(\left\lfloor \frac{|m_y|}{B^{expx-expy}} \right\rfloor \pm 5\right) B^{exp_x - exp_y} B^{exp_y} \\ &> (m_x + err_x)B^{exp_x} \pm (m_y \pm err_y)B^{exp_y}. \end{aligned}$$

   The correctness for the case $m_y < 0$ is similar.

Note the result of addition or subtraction of two error-free `BigFloat` numbers is also error-free.

Proposition 3

$$err_z B^{exp_z} \quad \leq \quad 6 \max\{err_x B^{exp_x},\, err_y B^{exp_y}\}.$$

Proof.

1. If $exp_x = exp_y$ then

$$\begin{aligned} err_z B^{exp_z} &\leq 2\, err'_z B^{exp'_z} \\ &= 2\,(err_x B^{exp_x} + err_y B^{exp_y}) \\ &\leq 4 \max\{err_x B^{exp_x},\, err_y B^{exp_y}\}. \end{aligned}$$

2. If $exp_x > exp_y$ and $err_x = 0$ then

$$\begin{aligned} err_z B^{exp_z} &\leq 2\, err'_z B^{exp'_z} \\ &= 2\, err_y B^{exp_y}. \end{aligned}$$

3. If $exp_x > exp_y$ and $err_x = 1$ or 2 then

$$\begin{aligned} err_z B^{exp_z} &= err'_z B^{exp'_z} && (\text{since } err'_z = err_x + 5 < 4B) \\ &\leq 6\, err_x B^{exp_x}. && (\text{since } err'_z \leq 6\, err_x) \end{aligned}$$

4. If $exp_x > exp_y$ and $err_x \geq 3$ then $err'_z = err_x + 5 < 3\, err_x$. Thus

$$\begin{aligned} err_z B^{exp_z} &\leq 2\, err'_z B^{exp'_z} \\ &< 6\, err_x B^{exp_x}. \end{aligned}$$

Q.E.D.

3.5.2 Multiplication

To compute $z = x \cdot y$, we let

$$\begin{aligned} m'_z &\leftarrow m_x\, m_y \\ err'_z &\leftarrow |m_x|\, err_y + err_x\, |m_y| + err_x\, err_y \\ exp'_z &\leftarrow exp_x + exp_y. \end{aligned}$$

To see that this is correct, it is enough to show

$$(m'_z - err'_z)\, B^{exp'_z} \quad \leq \quad \min\left\{ \begin{array}{l} (m_x + err_x)\, B^{exp_x}\, (m_y + err_y)\, B^{exp_y} \\ (m_x + err_x)\, B^{exp_x}\, (m_y - err_y)\, B^{exp_y} \\ (m_x - err_x)\, B^{exp_x}\, (m_y + err_y)\, B^{exp_y} \\ (m_x - err_x)\, B^{exp_x}\, (m_y - err_y)\, B^{exp_y} \end{array} \right\} \qquad (3)$$

and

$$(m'_z + err'_z)\, B^{exp'_z} \quad \geq \quad \max \left\{ \begin{array}{l} (m_x + err_x)\, B^{exp_x}\, (m_y + err_y)\, B^{exp_y} \\ (m_x + err_x)\, B^{exp_x}\, (m_y - err_y)\, B^{exp_y} \\ (m_x - err_x)\, B^{exp_x}\, (m_y + err_y)\, B^{exp_y} \\ (m_x - err_x)\, B^{exp_x}\, (m_y - err_y)\, B^{exp_y} \end{array} \right\}. \tag{4}$$

The inequalities (3) and (4) amount to the relatively obvious inequalities

$$m'_z - err'_z \quad \leq \quad \left\{ \begin{array}{l} (m_x + err_x)\, (m_y + err_y) \\ (m_x + err_x)\, (m_y - err_y) \\ (m_x - err_x)\, (m_y + err_y) \\ (m_x - err_x)\, (m_y - err_y) \end{array} \right\} \quad \leq \quad m'_z + err'_z. \tag{5}$$

It is easy to see that (5) holds whatever the signs of m_x and m_y.

Note the result of multiplication of two error-free `BigFloat` numbers is also error-free. In particular, z' does not need to be normalized.

Proposition 4

$$err_z B^{exp_z} \quad \leq \quad 6 \max \left\{ \begin{array}{l} |m_x|\, B^{exp_x} err_y B^{exp_y} \\ err_x B^{exp_x}\, |m_y|\, B^{exp_y} \\ err_x B^{exp_x} err_y B^{exp_y} \end{array} \right\}.$$

Proof.

$$\begin{aligned} & err_z B^{exp_z} \\ & \leq \; 2\, err'_z B^{exp'_z} \\ & = \; 2\left(|m_x|\, B^{exp_x} err_y B^{exp_y} + err_x B^{exp_x}\, |m_y|\, B^{exp_y} + err_x B^{exp_x} err_y B^{exp_y}\right) \\ & \leq \; 6 \max\left\{|m_x|\, B^{exp_x} err_y B^{exp_y},\; err_x B^{exp_x}\, |m_y|\, B^{exp_y},\; err_x B^{exp_x} err_y B^{exp_y}\right\}. \end{aligned}$$

Q.E.D.

3.5.3 Division

We would like to compute $z = x/y$. We may assume $|m_y| > err_y$, because otherwise 0 belongs to y and (by definition) the operator is not defined.

The mantissa of the result is computed by calling the function $div\,(m_x, m_y, r)$ for some $r \in \mathbb{N}$. Here, $div\,(m_x, m_y, r)$ is defined to be $div\,(m_x, m_y, r, \infty)$ which was defined in Section 3.2.1 *Approximation Algorithm*.

Let

$$I_{x/y} = \{X/Y \mid X \text{ and } Y \text{ are real, } X \text{ belongs to } x \text{ and } Y \text{ belongs to } y\}.$$

If $err_x > 0$ or $err_y > 0$ then $I_{x/y}$ is not a singleton and we can estimate the size of the interval $I_{x/y}$. Using this estimate, we choose a suitable r for the function call $div\,(m_x, m_y, r)$.

If $err_x = err_y = 0$ then $I_{x/y} = \{Z_0\}$, a singleton, and (unless m_y divides m_x) it is impossible to find an error-free `BigFloat` z to which Z_0 belongs. In this case, r is artificially specified to be some global constant $r_{\texttt{default}}$ which users can change.

We now describe the algorithm in several cases.

(a) CASE $err_x = err_y = 0$:
Let $\langle m''_z, err''_z, exp''_z \rangle = div\,(m_x, m_y, r_{\texttt{default}})$. Then, we set

$$\begin{array}{rcl} m'_z & \leftarrow & m''_z \\ err'_z & \leftarrow & err''_z \\ exp'_z & \leftarrow & exp_x - exp_y + exp''_z. \end{array}$$

Note z' does not need to be normalized, since $err'_z \leq 1$.

(b) CASE $err_x > 0$ or $err_y > 0$:

(b-1) CASE $|m_x| \leq err_x$:
If $|m_x| \leq err_x$ then 0 belongs to x. Thus, we set

$$\begin{array}{rcl} m'_z & \leftarrow & 0 \\ err'_z & \leftarrow & \left\lceil \frac{|m_x|+err_x}{|m_y|-err_y} \right\rceil \\ exp'_z & \leftarrow & exp_x - exp_y. \end{array}$$

The correctness follows from

$$\left| \frac{(m_x \pm err_x) B^{exp_x}}{(m_y \pm err_y) B^{exp_y}} \right| \quad \leq \quad \frac{(|m_x| + err_x) B^{exp_x}}{(|m_y| - err_y) B^{exp_y}} \quad \leq \quad err'_z B^{exp'_z}.$$

(b-2) CASE $|m_x| > err_x$:
Let $\langle m''_z, err''_z, exp''_z \rangle = div\,(m_x, m_y, r)$.
First, we will show how to estimate r. In this case,

$$I_{x/y} = \left[\frac{(|m_x| - err_x) B^{exp_x}}{(|m_y| + err_y) B^{exp_y}}, \frac{(|m_x| + err_x) B^{exp_x}}{(|m_y| - err_y) B^{exp_y}} \right].$$

Since $\frac{B^{exp_x}}{B^{exp_y}}$ does not affect the choice of r, it is convenient to use

$$J_{x/y} = I_{x/y} \frac{B^{exp_y}}{B^{exp_x}} = \left[\frac{|m_x| - err_x}{|m_y| + err_y}, \frac{|m_x| + err_x}{|m_y| - err_y} \right]$$

instead of $I_{x/y}$. Note $\frac{|m_x|}{|m_y|} \in J_{x/y}$. By Proposition 1,

$$\left| \frac{|m_x|}{|m_y|} - |m''_z|\, B^{exp''_z} \right| \quad \leq \quad \frac{|m_x|}{|m_y|} 2^{-r}.$$

Thus, r specifies an upper bound for the distance between $\frac{|m_x|}{|m_y|}$ and $|m''_z|\, B^{exp''_z}$. It is enough to choose r so that

$$|m''_z|\, B^{exp''_z} \in J_{x/y}, \tag{6}$$

because a larger r does not decrease the error of z' substantially. To ensure (6), it suffices to have

$$\frac{|m_x|}{|m_y|} 2^{-r} \quad \leq \quad \frac{|m_x|}{|m_y|} - \frac{|m_x| - err_x}{|m_y| + err_y}, \tag{7}$$

because

$$\frac{|m_x|}{|m_y|} - \frac{|m_x|}{|m_y|} 2^{-r} \leq |m''_z| B^{exp''_z} = \left\lfloor \frac{|m_x|}{|m_y| B^{exp''_z}} \right\rfloor B^{exp''_z} \leq \frac{|m_x|}{|m_y|} < \frac{|m_x|+err_x}{|m_y|-err_y}.$$

Hence, we are interested in the smallest r so that (7) holds. As getting the "smallest" r is difficult, we shall only compute some upper bound. Since

$$\begin{aligned}
\frac{|m_x|}{|m_y|} - \frac{|m_x|-err_x}{|m_y|+err_y} &= \frac{|m_x|err_y+err_x|m_y|}{|m_y|(|m_y|+err_y)} \\
&= \frac{|m_x|}{|m_y|+err_y}\left(\frac{err_x}{|m_x|} + \frac{err_y}{|m_y|}\right) \\
&\geq \frac{|m_x|}{2|m_y|}\left(\frac{err_x}{|m_x|} + \frac{err_y}{|m_y|}\right) \\
&\geq \frac{|m_x|}{2|m_y|} \max\left\{\frac{err_x}{|m_x|}, \frac{err_y}{|m_y|}\right\},
\end{aligned}$$

it suffices to have

$$2^{-r} \leq \begin{cases} \frac{1}{2}\frac{1}{|m_y|} & \text{if } err_x = 0 \text{ and } err_y > 0 \\ \frac{1}{2}\frac{1}{|m_x|} & \text{if } err_x > 0 \text{ and } err_y = 0 \\ \frac{1}{2}\max\left\{\frac{1}{|m_x|}, \frac{1}{|m_y|}\right\} & \text{if } err_x > 0 \text{ and } err_y > 0. \end{cases}$$

Hence, we set

$$r \leftarrow \begin{cases} \lfloor \lg |m_y| \rfloor + 2 & \text{if } err_x = 0 \text{ and } err_y > 0 \\ \lfloor \lg |m_x| \rfloor + 2 & \text{if } err_x > 0 \text{ and } err_y = 0 \\ \min\{\lfloor \lg |m_x| \rfloor, \lfloor \lg |m_y| \rfloor\} + 2 & \text{if } err_x > 0 \text{ and } err_y > 0 \end{cases} \tag{8}$$

and

$$\begin{aligned} m'_z &\leftarrow m''_z \\ exp'_z &\leftarrow exp_x - exp_y + exp''_z. \end{aligned}$$

Next, we compute err'_z. [1] It must satisfy

$$(m'_z - err'_z) B^{exp'_z} \leq \frac{(m_x \pm err_x)B^{exp x}}{(m_y \pm err_y)B^{exp y}} \leq (m'_z + err'_z) B^{exp'_z}$$

or equivalently

$$m'_z - err'_z \leq \frac{m_x \pm err_x}{(m_y \pm err_y)B^{exp''_z}} \leq m'_z + err'_z. \tag{9}$$

[1]We could have

$$err'_z \leftarrow \left(\frac{|m_x|+err_x}{|m_y|-err_y} - \frac{|m_x|-err_x}{|m_y|+err_y}\right)\frac{B^{exp x}}{B^{exp y}} = \frac{2(|m_x|err_y+err_x|m_y|)}{|m_y|^2-err_y^2}\frac{B^{exp x}}{B^{exp y}}.$$

But this error-bound is unnecessarily large and expensive to compute.

Since we assume that $|m_x| > err_x$ and $|m_y| > err_y$, the signs of $m_x \pm err_x$ and $m_y \pm err_y$ are the same as those of m_x and m_y, respectively. Dividing both sides of (9) by $\operatorname{sgn}(m_z) = \operatorname{sgn}(m_x m_y)$,

$$|m'_z| - err'_z \quad \leq \quad \frac{|m_x| \pm err_x}{(|m_y| \pm err_y) B^{exp''_z}} \quad \leq \quad |m'_z| + err'_z. \tag{10}$$

We claim that it suffices to set

$$err'_z \quad \leftarrow \quad \left\lceil \frac{\left\lfloor \frac{|m_x|}{B^{exp''_z}} \right\rfloor + \left\lfloor \frac{err_x}{B^{exp''_z}} \right\rfloor + \delta - |m_y|\,|m'_z| + err_y\,|m'_z|}{|m_y| - err_y} \right\rceil \tag{11}$$

where

$$\delta = \begin{cases} 0 & \text{if } exp''_z \leq 0 \\ 2 & \text{if } exp''_z > 0. \end{cases}$$

Note the quantity $\left\lfloor \frac{|m_x|}{B^{exp''_z}} \right\rfloor - |m_y|\,|m'_z|$ is the remainder of the integer division $\left\lfloor \frac{|m_x|}{B^{exp''_y}} \right\rfloor / |m_y|$, and it has already been computed by $div(m_x, m_y, r)$. If $\left\lfloor \frac{|m_x|}{B^{exp''_z}} \right\rfloor + \left\lfloor \frac{err_x}{B^{exp''_z}} \right\rfloor$ is replaced by $\left\lfloor \frac{|m_x| + err_x}{B^{exp''_z}} \right\rfloor$ then we could have $\delta \leq 1$, but we must perform another integer division to get it.

To see that (11) implies (10), we have

$$\begin{aligned} err'_z \quad &\geq \quad \frac{\left\lfloor \frac{|m_x|}{B^{exp''_z}} \right\rfloor + \left\lfloor \frac{err_x}{B^{exp''_z}} \right\rfloor + \delta - |m_y||m'_z| + err_y|m'_z|}{|m_y| - err_y} \\ &\geq \quad \frac{|m_x| + err_x}{(|m_y| - err_y) B^{exp''_z}} - |m'_z| \\ &\geq \quad \left| \frac{|m_x| \pm err_x}{(|m_y| \pm err_y) B^{exp''_z}} - |m'_z| \right| \end{aligned}$$

where the last inequality is proven as follows:

Let

$$D^+ = \left| \frac{|m_x| + err_x}{(|m_y| - err_y) B^{exp''_z}} - |m'_z| \right| \quad \text{and} \quad D^- = \left| \frac{|m_x| - err_x}{(|m_y| + err_y) B^{exp''_z}} - |m'_z| \right|.$$

Actually,

$$D^+ = \frac{|m_x| + err_x}{(|m_y| - err_y) B^{exp''_z}} - |m'_z|$$

since $\frac{|m_x| + err_x}{(|m_y| - err_y) B^{exp''_z}} > \frac{|m_x|}{|m_y| B^{exp''_z}} \geq \left\lfloor \frac{|m_x|}{|m_y| B^{exp''_z}} \right\rfloor = |m'_z|$, and

$$D^- = |m'_z| - \frac{|m_x| - err_x}{(|m_y| + err_y) B^{exp''_z}}$$

because of our choice of r. Obviously

$$\max\left\{D^+,\, D^-\right\} \quad \geq \quad \left| \frac{|m_x| \pm err_x}{(|m_y| \pm err_y) B^{exp''_z}} - |m'_z| \right|,$$

but $\max\{D^+, D^-\} = D^+$, since

$$\begin{aligned}
D^+ - D^- &= \frac{|m_x|+err_x}{(|m_y|-err_y)B^{exp''_z}} + \frac{|m_x|-err_x}{(|m_y|+err_y)B^{exp''_z}} - 2\,|m'_z| \\
&= \frac{2(|m_x||m_y|+err_x err_y)}{\left(|m_y|^2-err_y^2\right)B^{exp''_z}} - 2\,|m'_z| \\
&\geq \frac{2|m_x||m_y|}{|m_y|^2 B^{exp''_z}} - 2\,|m'_z| \\
&= 2\,\frac{|m_x|}{|m_y|B^{exp''_z}} - 2\left\lfloor \frac{|m_x|}{|m_y|B^{exp''_z}} \right\rfloor \\
&\geq 0.
\end{aligned}$$

Proposition 5

1. *If $err_x = err_y = 0$ then*

$$err_z B^{exp_z} \;\leq\; \frac{|m_x|B^{exp_x}}{|m_y|B^{exp_y}}\, 2^{-r_{\texttt{default}}}.$$

2. *If either $err_x = 0$ and $\frac{|m_y|}{2} \geq err_y > 0$*
   or $|m_x| > err_x > 0$ and $err_y = 0$
   or $|m_x| > err_x > 0$ and $\frac{|m_y|}{2} \geq err_y > 0$ then

$$err_z B^{exp_z} \;\leq\; 12\,\frac{|m_x|B^{exp_x}}{|m_y|B^{exp_y}} \max\left\{\frac{err_x}{|m_x|},\, \frac{err_y}{|m_y|}\right\}.$$

Proof.

1. This is immediate from definition for $div\,(m_x, m_y, r_{\texttt{default}})$ (see Section 3.2.1 *Approximation Algorithm*).

2. Since $err_z B^{exp_z} \leq 2\,err'_z B^{exp'_z}$, it is enough to show

$$err'_z B^{exp'_z} \;\leq\; 6\,\frac{|m_x|B^{exp_x}}{|m_y|B^{exp_y}} \max\left\{\frac{err_x}{|m_x|},\, \frac{err_y}{|m_y|}\right\}.$$

   We will consider three cases. The cases follow the logic of the algorithms for division.

   (a) Suppose $m_x = err_x = 0$ and $\frac{|m_y|}{2} \geq err_y > 0$.
   (This is the case (b-1) of the algorithm.)
   Then, $err'_z = 0$.

   (b) Suppose either $|m_x| > err_x > 0$ and $err_y = 0$
   or $|m_x| > err_x > 0$ and $\frac{|m_y|}{2} \geq err_y > 0$ and $\lfloor \lg |m_x| \rfloor \leq \lfloor \lg |m_y| \rfloor$.
   (This is the case (b-2) of the algorithm when r is set to be $\lfloor \lg |m_x| \rfloor + 2$ in (8).)
   In this case, we see from (2) that

$$exp''_z = \left\lfloor \frac{-\lfloor \lg |m_y| \rfloor - 3}{c} \right\rfloor.$$

Then

$$c \cdot exp''_z \;\; \leq \;\; -\left(\lfloor \lg |m_y| \rfloor + 1\right) - 2 \;\; < \;\; 0.$$

Thus

$$B^{exp''_z} = 2^{c \cdot exp''_z} < \frac{1}{4|m_y|}.$$

Hence

$$B^{exp'_z} = \frac{B^{expx} B^{exp''_z}}{B^{expy}} < \frac{|m_x| B^{expx}}{|m_y| B^{expy}} \frac{1}{4|m_x|}. \tag{12}$$

From (11)

$$\begin{aligned}
err'_z &< \frac{|m_x|+err_x}{(|m_y|-err_y)B^{exp''_z}} - |m''_z| + 1 && \text{(since } \delta = 0) \\
&< \frac{|m_x|+err_x}{(|m_y|-err_y)B^{exp''_z}} - \frac{|m_x|}{|m_y|B^{exp''_z}} + 2 \\
&= \frac{|m_x|}{(|m_y|-err_y)B^{exp''_z}} \left(\frac{|m_x|+err_x}{|m_x|} - \frac{|m_y|-err_y}{|m_y|} \right) + 2 \\
&\leq 2 \frac{|m_x|}{|m_y|B^{exp''_z}} \left(\frac{err_x}{|m_x|} + \frac{err_y}{|m_y|} \right) + 2.
\end{aligned}$$

Therefore

$$\begin{aligned}
err'_z B^{exp'_z} &< 2 \frac{|m_x|}{|m_y|B^{exp''_z}} B^{exp'_z} \left(\frac{err_x}{|m_x|} + \frac{err_y}{|m_y|} \right) + 2B^{exp'_z} \\
&< 2 \frac{|m_x|B^{expx}}{|m_y|B^{expy}} \left(\frac{err_x}{|m_x|} + \frac{err_y}{|m_y|} \right) + 2 \frac{|m_x|B^{expx}}{|m_y|B^{expy}} \frac{1}{4|m_x|} && \text{(by (12))} \\
&= 2 \frac{|m_x|B^{expx}}{|m_y|B^{expy}} \left(\frac{err_x}{|m_x|} + \frac{1}{4|m_x|} + \frac{err_y}{|m_y|} \right) \\
&\leq 6 \frac{|m_x|B^{expx}}{|m_y|B^{expy}} \max\left\{ \frac{err_x}{|m_x|}, \frac{err_y}{|m_y|} \right\}.
\end{aligned}$$

(c) Suppose either $|m_x| > err_x = 0$ and $\frac{|m_y|}{2} \geq err_y > 0$
or $|m_x| > err_x > 0$ and $\frac{|m_y|}{2} \geq err_y > 0$ and $\lfloor \lg |m_x| \rfloor > \lfloor \lg |m_y| \rfloor$.
(This is the case (b-2) of the algorithm when r is set to be $\lfloor \lg |m_y| \rfloor + 2$ in (8).)
In this case, (2) gives us

$$exp''_z = \left\lfloor \frac{\lfloor \lg |m_x| \rfloor - 2\lfloor \lg |m_y| \rfloor - 3}{c} \right\rfloor.$$

Then

$$c \cdot exp''_z \;\; \leq \;\; \lfloor \lg |m_x| \rfloor - 2\left(\lfloor \lg |m_y| \rfloor + 1\right) - 1.$$

Thus

$$B^{exp''_z} = 2^{c \cdot exp''_z} < \frac{|m_x|}{2|m_y|^2}.$$

Hence

$$\frac{B^{exp''_z}}{|m_x|} < \frac{1}{2|m_y|^2} \leq \frac{1}{4|m_y|} \tag{13}$$

where the last inequality follows from $|m_y| \geq 2$. Also

$$B^{exp'_z} = \frac{B^{expx} B^{exp''_z}}{B^{expy}} < \frac{|m_x| B^{expx}}{|m_y| B^{expy}} \frac{1}{2|m_y|}. \tag{14}$$

From (11)

$$\begin{aligned}
&err'_z \\
&< \frac{|m_x|+err_x}{(|m_y|-err_y)B^{exp''_z}} + \frac{\delta}{|m_y|-err_y} - |m''_z| + 1 \\
&< \frac{|m_x|+err_x}{(|m_y|-err_y)B^{exp''_z}} + \frac{2}{|m_y|-err_y} - \frac{|m_x|}{|m_y|B^{exp''_z}} + 2 \\
&= \frac{|m_x|}{(|m_y|-err_y)B^{exp''_z}} \left(\frac{|m_x|+err_x}{|m_x|} + \frac{2B^{exp''_z}}{|m_x|} - \frac{|m_y|-err_y}{|m_y|} \right) + 2 \\
&\le 2\,\frac{|m_x|}{|m_y|B^{exp''_z}} \left(\frac{err_x}{|m_x|} + \frac{1}{2|m_y|} + \frac{err_y}{|m_y|} \right) + 2. \qquad \text{(by (13))}
\end{aligned}$$

Therefore

$$\begin{aligned}
&err'_z B^{exp'_z} \\
&< 2\,\frac{|m_x|B^{exp_x}}{|m_y|B^{exp_y}} \left(\frac{err_x}{|m_x|} + \frac{1}{2|m_y|} + \frac{err_y}{|m_y|} \right) + 2\,\frac{|m_x|B^{exp_x}}{|m_y|B^{exp_y}} \frac{1}{2|m_y|} \qquad \text{(by (14))} \\
&= 2\,\frac{|m_x|B^{exp_x}}{|m_y|B^{exp_y}} \left(\frac{err_x}{|m_x|} + \frac{err_y}{|m_y|} + \frac{1}{|m_y|} \right) \\
&\le 6\,\frac{|m_x|B^{exp_x}}{|m_y|B^{exp_y}} \max\left\{ \frac{err_x}{|m_x|}, \frac{err_y}{|m_y|} \right\}.
\end{aligned}$$

This proposition does not cover the cases $0 = |m_x| < err_x$ or $0 < |m_x| \le err_x$ since there is no upper bound for err'_z in terms of x and y, nor does it cover the cases $2\,err_y > |m_y| > err_y > 0$ since we may assume that non-exact y can be recomputed so that it will have the error-bound $[r, a]$ with $r \ge 1$ and $a \ge -\lfloor \lg |m_y| \rfloor + 1$.

Q.E.D.

3.6 Squareroot

Let $x = \langle m_x, err_x, exp_x \rangle$ be a BigFloat with $m_x \ge 0$. Define $sqrt\,(x)$ to be a BigFloat $y = \langle m_y, err_y, exp_y \rangle$ such that $\forall$ real $X \ge 0$, if X belongs to x then $\sqrt{X}$ belongs to y.

For $x = \langle m_x, err_x, exp_x \rangle$ with $m_x < 0$, $sqrt\,(x)$ is not defined.

3.6.1 Algorithm for $sqrt\,(x)$

Let $x = \langle m_x, err_x, exp_x \rangle$ be a BigFloat with $m_x \ge 0$. We would like to compute a BigFloat $y = sqrt\,(x)$.

Define the function $sqrt\,(X, A)$ as follows:

For $X \in \mathbb{N}$ and $A \in \mathbb{Z}$, $sqrt\,(X, A)$ returns an error-free BigFloat $z = \langle m_z, 0, exp_z \rangle$ such that $\left|\sqrt{X} - m_z B^{exp_z}\right| \le 2^{-A}$.

The function $sqrt\,(x)$ will be computed by calling $sqrt\,(X, A)$ for some X and A. Let

$$I_{\sqrt{x}} = \left\{ \sqrt{\xi} \mid \xi \text{ is non-negative real and } \xi \text{ belongs to } x \right\}.$$

If $err_x > 0$ then $I_{\sqrt{x}}$ is not a singleton and we can estimate the size of the interval $I_{\sqrt{x}}$. Using this estimate, we choose a suitable A for the function call $sqrt\,(X, A)$.

If $err_x = 0$ then $I_{\sqrt{x}} = \{\zeta_0\}$, a singleton, and (unless m_x is a prefect square) it is impossible to find an error-free `BigFloat` y to which ζ_0 belongs. In this case, A is artificially specified by some default precision.

Let

$$\delta = \begin{cases} 0 & \text{if } exp_x \text{ is even} \\ 1 & \text{if } exp_x \text{ is odd.} \end{cases}$$

We now describe the algorithm in several cases. In each case, we first define a `BigFloat` $y' = \left\langle m'_y, err'_y, exp'_y \right\rangle$ whose normalization would be y.

(a) CASE $m_x \le err_x$:

If $m_x \le err_x$ then 0 belongs to x. Thus, we set

$$\begin{aligned} m'_y &\leftarrow 0 \\ err'_y &\leftarrow \begin{cases} 0 & \text{if } err_x = 0 \\ 2\left(\lfloor \texttt{sqrt}\,((\texttt{double})\,err_x)\rfloor + 1\right) 2^{\delta \lceil \frac{c}{2} \rceil} & \text{if } err_x > 0. \end{cases} \\ exp'_y &\leftarrow \frac{exp_x - \delta}{2} \end{aligned}$$

Here, `double sqrt` (`double d`) is the function in the standard library of `C++` to compute $\sqrt{\texttt{d}}$ for $\texttt{d} \ge 0$ which is correctly rounded as outlined in the IEEE 754 floating-point standard [PH90] [Gol91].

The correctness follows from

$$\begin{aligned} err'_y B^{exp'_y} &> 2\sqrt{err_x}\sqrt{B^\delta}\sqrt{\frac{B^{exp_x}}{B^\delta}} \\ &> \sqrt{2\,err_x B^{exp_x}} \\ &\ge \sqrt{(m_x + err_x)\,B^{exp_x}}. \end{aligned}$$

(b) CASE $m_x > err_x = 0$:

We would like to compute y' such that

$$err'_y B^{exp'_y} \quad \le \quad 2^{-a_{\texttt{default}}-1}$$

where $a_{\texttt{default}}$ is some global constant which users can change.

Let $\langle m_z, 0, exp_z \rangle = sqrt\left(m_x B^\delta, a_{\texttt{default}} + 1 + c \cdot \frac{exp_x - \delta}{2}\right)$ and

$$p \quad = \quad a_{\texttt{default}} + 1 + c \cdot \frac{exp_x - \delta}{2} + c \cdot exp_z.$$

If $p \le 0$ then $2^{-p} \ge 1$. Thus, we set

$$\begin{aligned} m'_y &\leftarrow m_z \\ err'_y &\leftarrow 2^{-p} \\ exp'_y &\leftarrow exp_z + \frac{exp_x - \delta}{2}. \end{aligned}$$

The correctness is obvious.

If $p > 0$ then $2^{-p} < 1$. We shift 2^{-p} left $\left\lceil \frac{p}{c} \right\rceil$ chunks to get err'_y so that $err'_y \geq 1$. Thus

$$\begin{aligned} m'_y &\leftarrow m_z B^{\lceil \frac{p}{c} \rceil} \\ err'_y &\leftarrow 2^{-p} B^{\lceil \frac{p}{c} \rceil} \\ exp'_y &\leftarrow exp_z + \tfrac{exp_x - \delta}{2} - \left\lceil \frac{p}{c} \right\rceil = -\left\lceil \frac{a_{\mathtt{default}}+1}{c} \right\rceil . \end{aligned}$$

The correctness is also obvious.

(c) CASE $m_x > err_x > 0$**:**

Let $\langle m_z, 0, exp_z \rangle = sqrt\left(m_x B^\delta, A\right)$ for some A.

First, we will show how to estimate A. In this case,

$$I_{\sqrt{x}} = \left[\sqrt{(m_x - err_x)\, B^{exp_x}}, \sqrt{(m_x + err_x)\, B^{exp_x}}\right].$$

It is convenient to use

$$J_{\sqrt{x}} = \frac{I_{\sqrt{x}}}{\sqrt{B^{exp_x - \delta}}} = \left[\sqrt{(m_x - err_x)\, B^\delta}, \sqrt{(m_x + err_x)\, B^\delta}\right]$$

instead of $I_{\sqrt{x}}$. Note $\sqrt{m_x B^\delta} \in J_{\sqrt{x}}$. By definition for $sqrt\left(m_x B^\delta, A\right)$, A specifies an upper bound for the distance between $\sqrt{m_x B^\delta}$ and $m_z B^{exp_z}$. It is enough to choose A so that

$$m_z B^{exp_z} \in J_{\sqrt{x}}, \tag{15}$$

because a larger A does not decrease the error of y' substantially. Since $m_x > err_x$,

$$\sqrt{m_x B^\delta} - \sqrt{(m_x - err_x)\, B^\delta} \quad > \quad \sqrt{(m_x + err_x)\, B^\delta} - \sqrt{m_x B^\delta}. \tag{16}$$

Hence, to ensure (15), it suffices to have

$$\sqrt{(m_x + err_x)\, B^\delta} - \sqrt{m_x B^\delta} \quad \geq \quad 2^{-A}. \tag{17}$$

We are interested in the smallest A such that (17) holds. As getting the "smallest" A is difficult, we shall only compute some upper bound. Since $m_x > err_x > 0$,

$$1 + \frac{err_x}{m_x} \quad > \quad 1 + \frac{err_x}{2\,m_x} + \frac{err_x^2}{16\,m_x^2} \quad = \quad \left(1 + \frac{err_x}{4\,m_x}\right)^2 .$$

Hence

$$\sqrt{1 + \frac{err_x}{m_x}} - 1 \quad > \quad \frac{err_x}{4\,m_x}.$$

Therefore

$$\begin{aligned} \sqrt{(m_x + err_x)\, B^\delta} - \sqrt{m_x B^\delta} &= \sqrt{m_x B^\delta}\left(\sqrt{1 + \frac{err_x}{m_x}} - 1\right) \\ &> \frac{err_x \sqrt{B^\delta}}{4\sqrt{m_x}} \\ &= 2^{\lg err_x - 2 - \frac{1}{2}\lg m_x + \delta \frac{c}{2}} \\ &> 2^{\lfloor \lg err_x \rfloor - \left\lfloor \frac{1}{2}(\lfloor \lg m_x \rfloor + 1 - \delta c) \right\rfloor - 3}. \end{aligned} \tag{18}$$

Hence, we set

$$A \;\leftarrow\; -\lfloor \lg err_x \rfloor + \left\lfloor \frac{1}{2}\left(\lfloor \lg m_x \rfloor + 1 - \delta\, c\right)\right\rfloor + 3.$$

Next, we compute err'_y. It must satisfy

$$\left(m'_y - err'_y\right) B^{exp'_y} \;\le\; \sqrt{(m_x \pm err_x)\, B^{exp_x}} \;\le\; \left(m'_y + err'_y\right) B^{exp'_y}.$$

Considering (16), it suffices to set err'_y so that

$$\sqrt{m_x B^{exp_x}} - \sqrt{(m_x - err_x)\, B^{exp_x}} + 2^{-A}\sqrt{\frac{B^{exp_x}}{B^{\delta}}} \;\le\; err'_y B^{exp_z}.$$

Since $1 > 1 - \frac{err_x}{m_x} > 0$,

$$1 - \sqrt{1 - \frac{err_x}{m_x}} \;<\; 1 - \left(1 - \frac{err_x}{m_x}\right) \;=\; \frac{err_x}{m_x}.$$

Hence

$$\begin{aligned}
&\sqrt{m_x B^{exp_x}} - \sqrt{(m_x - err_x)\, B^{exp_x}} + 2^{-A}\sqrt{\frac{B^{exp_x}}{B^{\delta}}} \\
&= \sqrt{m_x B^{exp_x}}\left(1 - \sqrt{1 - \frac{err_x}{m_x}}\right) + \frac{err_x\sqrt{B^{\delta}}}{4\sqrt{m_x}}\sqrt{\frac{B^{exp_x}}{B^{\delta}}} \quad \text{(by (18))} \\
&< \frac{err_x\sqrt{B^{exp_x}}}{\sqrt{m_x}} + \frac{err_x\sqrt{B^{exp_x}}}{4\sqrt{m_x}} \\
&< 2\,\frac{err_x\sqrt{B^{exp_x}}}{\sqrt{m_x}}.
\end{aligned}$$

Therefore, err'_y must be set so that

$$2\,\frac{err_x\sqrt{B^{exp_x}}}{\sqrt{m_x}} \;\le\; err'_y B^{exp'_y}.$$

Let

$$q \;=\; -1 - \lceil \lg err_x \rceil + \left\lfloor \frac{1}{2}\lfloor \lg m_x \rfloor \right\rfloor - \delta\left\lceil \frac{c}{2} \right\rceil + c \cdot exp_z.$$

Note if $q \le 0$ then $\frac{2\, err_x\sqrt{B^{\delta}}}{\sqrt{m_x}B^{exp_z}} \ge 1$.

If $q \le 0$ then we set

$$\begin{aligned}
m'_y &\;\leftarrow\; m_z \\
err'_y &\;\leftarrow\; 2\, err_x\, 2^{-\left\lfloor \frac{1}{2}\lfloor \lg m_x \rfloor \right\rfloor + \delta\left\lceil \frac{c}{2}\right\rceil - c\cdot exp_z} \\
exp'_y &\;\leftarrow\; exp_z + \frac{exp_x - \delta}{2}.
\end{aligned}$$

This is correct, because

$$err'_y \;\ge\; 2\, err_x \frac{1}{\sqrt{m_x}}\sqrt{B^{\delta}}\frac{1}{B^{exp_z}} \;\ge\; 1$$

and

$$err'_y B^{exp'_y} \;\ge\; 2\,\frac{err_x\sqrt{B^{\delta}}}{\sqrt{m_x}B^{exp_z}}\,\frac{B^{exp_z}\sqrt{B^{exp_x}}}{\sqrt{B^{\delta}}} \;=\; 2\,\frac{err_x\sqrt{B^{exp_x}}}{\sqrt{m_x}}.$$

If $q > 0$ then $2^{-q} < 1$. We shift 2^{-q} left $\left\lceil \frac{q}{c} \right\rceil$ chunks to get err'_y so that $err'_y \geq 1$. Thus

$$\begin{array}{rcl} m'_y & \leftarrow & m_z B^{\left\lceil \frac{q}{c} \right\rceil} \\ err'_y & \leftarrow & 2^{-q} B^{\left\lceil \frac{q}{c} \right\rceil} \\ exp'_y & \leftarrow & exp_z + \frac{exp_x - \delta}{2} - \left\lceil \frac{q}{c} \right\rceil = \frac{exp_x - \delta}{2} - \left\lceil \frac{-1-\lceil \lg err_x \rceil + \left\lfloor \frac{1}{2} \lfloor \lg m_x \rfloor \right\rfloor - \delta \left\lceil \frac{c}{2} \right\rceil}{c} \right\rceil . \end{array}$$

The correctness is similar to the previous case. Note y' does not need to be normalized, since $err'_y < B$.

Proposition 6

1. *If* $err_x = 0$ *then*

$$err_y B^{exp_y} \quad \leq \quad 2^{-a_{\texttt{default}}}.$$

2. *If* $err_x > 0$ *then* [2]

$$err_y B^{exp_y} \quad \leq \quad 16\sqrt{err_x B^{exp_x}}.$$

Proof.

1. If $m_x = err_x = 0$ then $err'_y = 0$.

   If $m_x > err_x = 0$ then

$$err_y B^{exp_y} \quad \leq \quad 2\, err'_y B^{exp'_y} \quad \leq \quad 2 \cdot 2^{-a_{\texttt{default}}-1}.$$

2. If $m_x \leq err_x$ and $err_x > 0$ then

$$\begin{array}{rcl} err_y B^{exp_y} & \leq & 2\, err'_y B^{exp'_y} \\ & \leq & 2\left(\sqrt{err_x} + 2\right) 2^{\delta \left\lceil \frac{c}{2} \right\rceil} \sqrt{\frac{B^{exp_x}}{B^{\delta}}} \\ & \leq & 2\left(\sqrt{err_x} + 2\sqrt{err_x}\right) 2\sqrt{B^{\delta}} \sqrt{\frac{B^{exp_x}}{B^{\delta}}} \\ & = & 12\sqrt{err_x B^{exp_x}}. \end{array}$$

   If $m_x > err_x > 0$ and $q = -1 - \lceil \lg err_x \rceil + \left\lfloor \frac{1}{2} \lfloor \lg m_x \rfloor \right\rfloor - \delta \left\lceil \frac{c}{2} \right\rceil + c \cdot exp_z \leq 0$ then

$$\begin{array}{rcl} err_y B^{exp_y} & \leq & 2\, err'_y B^{exp'_y} \\ & = & 2 \cdot 2\, err_x\, 2^{-\left\lfloor \frac{1}{2} \lfloor \lg m_x \rfloor \right\rfloor + \delta \left\lceil \frac{c}{2} \right\rceil - c \cdot exp_z} \frac{B^{exp_z} \sqrt{B^{exp_x}}}{\sqrt{B^{\delta}}} \\ & < & 2 \cdot 2\, err_x \frac{2}{\sqrt{m_x}}\, 2\sqrt{B^{\delta}} \frac{1}{B^{exp_z}} \frac{B^{exp_z} \sqrt{B^{exp_x}}}{\sqrt{B^{\delta}}} \\ & = & 16\sqrt{\frac{err_x}{m_x}} \sqrt{err_x B^{exp_x}} \\ & < & 16\sqrt{err_x B^{exp_x}}. \end{array}$$

[2]If c is even then we could replace 16 by 8, because $\left\lceil \frac{c}{2} \right\rceil = \frac{c}{2}$, i.e., $2^{\delta \left\lceil \frac{c}{2} \right\rceil} = \sqrt{B^{\delta}}$. Note we set $c = \left\lfloor \frac{L}{2} \right\rfloor - 2$ where $2^L - 1$ is the largest `unsigned long`. In almost all systems, L is some positive power of 2 (typically $L = 32$), and hence, c is even.

If $m_x > err_x > 0$ and $q > 0$ then

$$\begin{aligned}
err_y B^{exp_y} &= err'_y B^{exp'_y} \\
&= 2^{1+\lceil \lg err_x \rceil - \left\lfloor \frac{1}{2} \lfloor \lg m_x \rfloor \right\rfloor + \delta \left\lceil \frac{c}{2} \right\rceil - c \cdot exp_z} B^{\left\lceil \frac{q}{c} \right\rceil} \frac{B^{exp_z} \sqrt{B^{exp_x}}}{\sqrt{B^\delta} B^{\left\lceil \frac{q}{c} \right\rceil}} \\
&< 2 \cdot 2\, err_x \frac{2}{\sqrt{m_x}}\, 2\sqrt{B^\delta} \frac{1}{B^{exp_z}} \frac{B^{exp_z} \sqrt{B^{exp_x}}}{\sqrt{B^\delta}} \\
&= 16 \sqrt{\frac{err_x}{m_x}} \sqrt{err_x B^{exp_x}} \\
&< 16 \sqrt{err_x B^{exp_x}}.
\end{aligned}$$

Q.E.D.

3.6.2 Algorithm for $sqrt\,(X, A)$

We now describe the algorithm for $sqrt\,(X, A)$ defined above.

We simulate Newton's method to compute $\sqrt{X}$ over `BigFloat` numbers. First, we introduce some functions which will be used in our algorithm.

For $X \in \mathbb{N}$, error-free `BigFloat` numbers $y = \langle m_y, 0, exp_y \rangle$ with $m_y > 0$ and $z = \langle m_z, 0, exp_z \rangle$ with $m_z \geq 0$, and $A \in \mathbb{Z}$, define

$$\begin{aligned}
Q_X(y, A) &= \left\langle m'_q, 0, exp'_q - exp_y \right\rangle \\
&\quad \text{where } \left\langle m'_q, err'_q, exp'_q \right\rangle = div\,(X, m_y, \infty, A - c \cdot exp_y)\,, \\
H(z) &= \left\langle \left\lfloor \frac{m_z}{2} \right\rfloor, 0, exp_z \right\rangle, \\
N_X(y, A) &= H\,(y + Q_X(y, A))\,, \\
S_X(y) &= \frac{1}{2} \left(m_y B^{exp_y} + \frac{X}{m_y B^{exp_y}} \right).
\end{aligned}$$

Note the function $S_X(y)$ yields a fraction, not a `BigFloat`.

Lemma 7

1.

$$\frac{X}{m_y B^{exp_y}} - 2^{-A} \;\leq\; \frac{X}{m_y B^{exp_y}} - B^{\left\lfloor \frac{-A}{c} \right\rfloor} \;<\; Q_X(y, A) \;\leq\; \frac{X}{m_y B^{exp_y}}. \tag{19}$$

2.

$$S_X(y) - 2^{-A} \;\leq\; S_X(y) - B^{\left\lfloor \frac{-A}{c} \right\rfloor} \;<\; N_X(y, A) \;\leq\; S_X(y). \tag{20}$$

Proof.

1. By definition (see Section 3.2.1 *Approximation Algorithm*),

$$Q_X(y, A) = \left\lfloor \frac{X}{m_y B^{\left\lfloor \frac{-A}{c} \right\rfloor + exp_y}} \right\rfloor B^{\left\lfloor \frac{-A}{c} \right\rfloor}.$$

Thus

$$\begin{aligned}
\frac{X}{m_y B^{exp_y}} - B^{\lfloor \frac{-A}{c} \rfloor} &= \left(\frac{X}{m_y B^{\lfloor \frac{-A}{c} \rfloor + exp_y}} - 1 \right) B^{\lfloor \frac{-A}{c} \rfloor} \\
&< Q_X(y, A) \\
&\leq \frac{X}{m_y B^{\lfloor \frac{-A}{c} \rfloor + exp_y}} B^{\lfloor \frac{-A}{c} \rfloor} \\
&= \frac{X}{m_y B^{exp_y}}.
\end{aligned}$$

2. By definition of `BigFloat` addition,

$$\begin{aligned}
\frac{1}{2}\left(m_y B^{exp_y} + Q_X(y, A)\right) - \frac{1}{2} B^{\min\left\{exp_y, \lfloor \frac{-A}{c} \rfloor\right\}} &\leq H\left(y + Q_X(y, A)\right) \\
&\leq \frac{1}{2}\left(m_y B^{exp_y} + Q_X(y, A)\right).
\end{aligned}$$

By (19)

$$\begin{aligned}
&\frac{1}{2}\left(m_y B^{exp_y} + Q_X(y, A)\right) - \frac{1}{2} B^{\min\left\{exp_y, \lfloor \frac{-A}{c} \rfloor\right\}} \\
&\quad > \frac{1}{2}\left(m_y B^{exp_y} + \frac{X}{m_y B^{exp_y}} - B^{\lfloor \frac{-A}{c} \rfloor}\right) - \frac{1}{2} B^{\min\left\{exp_y, \lfloor \frac{-A}{c} \rfloor\right\}} \\
&\quad \geq \frac{1}{2}\left(m_y B^{exp_y} + \frac{X}{m_y B^{exp_y}}\right) - B^{\lfloor \frac{-A}{c} \rfloor}
\end{aligned}$$

and

$$\frac{1}{2}\left(m_y B^{exp_y} + Q_X(y, A)\right) \leq \frac{1}{2}\left(m_y B^{exp_y} + \frac{X}{m_y B^{exp_y}}\right).$$

Q.E.D.

Lemma 8

1. $\sqrt{X}$ lies between $m_y B^{exp_y}$ and $\frac{X}{m_y B^{exp_y}}$, i.e.,

$$\left(m_y B^{exp_y} - \sqrt{X}\right)\left(\frac{X}{m_y B^{exp_y}} - \sqrt{X}\right) \leq 0. \tag{21}$$

2. If $m_y B^{exp_y} \geq \sqrt{X}$ then

$$m_y B^{exp_y} \geq S_X(y) \geq \sqrt{X} \tag{22}$$

where the equalities hold iff $m_y B^{exp_y} = \sqrt{X}$.

Proof.

1. If $m_y B^{exp_y} \geq \sqrt{X}$ then
$$\frac{X}{m_y B^{exp_y}} \leq \frac{X}{\sqrt{X}} = \sqrt{X},$$
and if $m_y B^{exp_y} < \sqrt{X}$ then
$$\frac{X}{m_y B^{exp_y}} > \frac{X}{\sqrt{X}} = \sqrt{X}.$$

2. Note $S_X(y)$ is the midpoint between $m_y B^{exp_y}$ and $\frac{X}{m_y B^{exp_y}}$, but
$$\sqrt{X} - \min\left\{m_y B^{exp_y}, \frac{X}{m_y B^{exp_y}}\right\} \quad \leq \quad \max\left\{m_y B^{exp_y}, \frac{X}{m_y B^{exp_y}}\right\} - \sqrt{X}$$
since
$$\begin{aligned}\left(\max\left\{m_y B^{exp_y}, \frac{X}{m_y B^{exp_y}}\right\} - \sqrt{X}\right) - \left(\sqrt{X} - \min\left\{m_y B^{exp_y}, \frac{X}{m_y B^{exp_y}}\right\}\right)&\\ = \; & m_y B^{exp_y} - 2\sqrt{X} + \frac{X}{m_y B^{exp_y}} \\ = \; & \frac{m_y^2 B^{2exp_y} - 2\, m_y B^{exp_y} \sqrt{X} + X}{m_y B^{exp_y}} \\ = \; & \frac{\left(m_y B^{exp_y} - \sqrt{X}\right)^2}{m_y B^{exp_y}} \\ \geq \; & 0.\end{aligned}$$

Q.E.D.

Now, we present the algorithm for $sqrt\,(X, A)$. Since the case $X = 0$ or 1 is trivial, assume $X \geq 2$. Set
$$\begin{aligned} y_0 &= \langle X, 0, 0\rangle \\ y_i &= N_X\,(y_{i-1}, A) \quad i = 1, 2, \ldots. \end{aligned}$$
Note $m_{y_0} B^{exp_{y_0}} \geq \sqrt{X}$.

The iteration continues until we will find the smallest $i \in \mathbb{N}$ such that
$$Q_X\,(y_i, A) + 2^{-A} \quad \geq \quad m_{y_i} B^{exp_{y_i}}, \tag{23}$$
and $\langle m_{y_i}, 0, exp_{y_i}\rangle$ will be returned. The correctness follows from the following lemmas.

Lemma 9 *The condition (23) is sufficient to have* $\left|\sqrt{X} - m_{y_i} B^{exp_{y_i}}\right| \leq 2^{-A}$.

Proof. There are two cases.

1. If $Q_X\,(y_i, A) + 2^{-A} \geq m_{y_i} B^{exp_{y_i}} \geq \sqrt{X}$ then
$$0 \leq m_{y_i} B^{exp_{y_i}} - \sqrt{X} \leq 2^{-A},$$
since
$$\begin{aligned} Q_X\,(y_i, A) + 2^{-A} \quad &\geq \quad m_{y_i} B^{exp_{y_i}} && \text{(by assumption)} \\ &\geq \quad \sqrt{X} && \text{(by assumption)} \\ &\geq \quad \frac{X}{m_{y_i} B^{exp_{y_i}}} && \text{(by (21))} \\ &\geq \quad Q_X\,(y_i, A)\,. && \text{(by (19))} \end{aligned}$$

2. Suppose

$$Q_X\left(y_i, A\right) + 2^{-A} \geq m_{y_i} B^{exp_{y_i}} \quad \text{and} \quad \sqrt{X} > m_{y_i} B^{exp_{y_i}}. \tag{24}$$

Actually, the first condition of (24) is redundant, i.e., (24) is equivalent to $\sqrt{X} > m_{y_i} B^{exp_{y_i}}$. In fact, if $\sqrt{X} > m_{y_i} B^{exp_{y_i}}$ then

$$\begin{array}{rcll} Q_X\left(y_i, A\right) + 2^{-A} & > & \frac{X}{m_{y_i} B^{exp_{y_i}}} & \text{(by (19))} \\ & \geq & \sqrt{X} & \text{(by (21))} \\ & > & m_{y_i} B^{exp_{y_i}}. & \text{(by assumption)} \end{array}$$

If $\sqrt{X} > m_{y_i} B^{exp_{y_i}}$ and $m_{y_j} B^{exp_{y_j}} > \sqrt{X}$ for $j = 0, 1, 2, \ldots, i-1$ then

$$0 < \sqrt{X} - m_{y_i} B^{exp_{y_i}} \leq 2^{-A},$$

since

$$\begin{array}{rcll} S_X\left(y_{i-1}\right) & > & \sqrt{X} & \text{(by (22))} \\ & > & m_{y_i} B^{exp_{y_i}} = N_X\left(y_{i-1}, A\right) & \text{(by (21))} \\ & > & S_X\left(y_{i-1}\right) - 2^{-A}. & \text{(by (20))} \end{array}$$

Q.E.D.

Lemma 10 *There exists $i \in \mathbb{N}$ such that (23) holds.*

Proof. If $m_{y_j} B^{exp_{y_j}} \geq \sqrt{X}$ then $m_{y_j} B^{exp_{y_j}} \geq m_{y_{j+1}} B^{exp_{y_{j+1}}}$, because

$$\begin{array}{rcll} m_{y_j} B^{exp_{y_j}} & \geq & S_X\left(y_j\right) & \text{(by (22))} \\ & \geq & N_X\left(y_j, A\right) & \text{(by (20))} \\ & = & m_{y_{j+1}} B^{exp_{y_{j+1}}}. & \end{array}$$

Since $m_{y_0} B^{exp_{y_0}} \geq \sqrt{X}$, the sequence $\left\{m_{y_j} B^{exp_{y_j}}\right\}$ is non-increasing for small j. Depending on the behavior of $\left\{m_{y_j} B^{exp_{y_j}}\right\}$, we consider several cases.

1. If $\exists i \in \mathbb{N}$ such that $m_{y_i} B^{exp_{y_i}} = m_{y_{i+1}} B^{exp_{y_{i+1}}}$ and $m_{y_j} B^{exp_{y_j}} > m_{y_{j+1}} B^{exp_{y_{j+1}}}$ for $j = 0, 1, 2, \ldots, i-1$ then

   $$m_{y_i} B^{exp_{y_i}} = S_X\left(y_i\right) = N_X\left(y_i, A\right) = m_{y_{i+1}} B^{exp_{y_{i+1}}}.$$

   By (19)

   $$m_{y_i} B^{exp_{y_i}} = \frac{X}{m_{y_i} B^{exp_{y_i}}} < Q_X\left(y_i, A\right) + 2^{-A}.$$

   Note, in this case, $m_{y_i} B^{exp_{y_i}} = \sqrt{X}$.

2. Suppose $\left\{m_{y_j} B^{exp_{y_j}}\right\}$ is strictly decreasing for all small j.

(a) If $\exists i \in \mathbb{N}_{>0}$ such that $\sqrt{X} > m_{y_i} B^{exp_{y_i}}$ and $m_{y_j} B^{exp_{y_j}} > \sqrt{X}$ for $j = 0, 1, 2, \ldots, i-1$ then we have already seen in Lemma 9 that $Q_X(y_i, A) + 2^{-A} > m_{y_i} B^{exp_{y_i}}$.

(b) Otherwise, $\forall j \in \mathbb{N}$ $m_{y_j} B^{exp_{y_j}} > \sqrt{X}$. By (22), $\forall j \in \mathbb{N}$

$$m_{y_j} B^{exp_{y_j}} > S_X(y_j) \geq m_{y_{j+1}} B^{exp_{y_{j+1}}} > \sqrt{X}.$$

Thus, both $\left\{m_{y_j} B^{exp_{y_j}}\right\}$ and $\{S_X(y_j)\}$ are strictly decreasing for all j and bounded from below by $\sqrt{X}$. Hence, both of them converge. Moreover, $\lim_{j\to\infty} m_{y_j} B^{exp_{y_j}} = \lim_{j\to\infty} S_X(y_j)$. Hence,

$$\lim_{j\to\infty} m_{y_j} B^{exp_{y_j}} = \sqrt{X},$$

and in particular, $\exists i \in \mathbb{N}$ such that (23) holds.

Q.E.D.

Finally, we claim that the convergence of $\left\{m_{y_j} B^{exp_{y_j}}\right\}$ is quadratic as the original Newton's method. Let i be the smallest non-negative integer such that (23) holds. By Lemma 10, for $j = 1, 2, \ldots, i-1$, $m_{y_j} B^{exp_{y_j}} > \sqrt{X}$. Then

$$\begin{array}{lll}
m_{y_j} B^{exp_{y_j}} - \sqrt{X} & & \\
= & N_X(y_{j-1}, A) - \sqrt{X} & \\
\leq & S_X(y_{j-1}) - \sqrt{X} & \text{(by (20))} \\
= & \frac{1}{2}\left(m_{y_{j-1}} B^{exp_{y_{j-1}}} + \frac{X}{m_{y_{j-1}} B^{exp_{y_{j-1}}}}\right) - \sqrt{X} & \\
= & \frac{m_{y_{j-1}}^2 B^{2exp_{y_{j-1}}} - 2\, m_{y_{j-1}} B^{exp_{y_{j-1}}} \sqrt{X} + X}{2\, m_{y_{j-1}} B^{exp_{y_{j-1}}}} & \\
< & \frac{1}{2\sqrt{X}}\left(m_{y_{j-1}} B^{exp_{y_{j-1}}} - \sqrt{X}\right)^2. & \text{(since } m_{y_{j-1}} B^{exp_{y_{j-1}}} > \sqrt{X}\text{)}
\end{array}$$

Summarizing:

```
if $X = 0$ then return $\langle 0, 0, 0 \rangle$
else if $X = 1$ then return $\langle 1, 0, 0 \rangle$
else
    $y \leftarrow \langle X, 0, 0 \rangle$
    loop
        $q \leftarrow Q_X(y, A)$
        if $y \leq q + 2^{-A}$ then return $\langle m_y, 0, exp_y \rangle$
        $y \leftarrow H(y + q)$
    end loop
```

3.7 Implementation

We use the well-known "letter-envelope" technique [Cop92]. Any `BigFloat` number is associated with two instances, one belongs to the "envelope" class `BigFloat` and the other belongs to the "letter" class `BigFloatRep`.

The "envelope" class `BigFloat` is defined as follows:

```
class  BigFloat
{
  BigFloatRep*  rep;

  //  private member functions come here.

public:

  //  public member functions come here.
};
```

The "letter" class `BigFloatRep` is defined as follows:

```
class  BigFloatRep
{
friend  class  BigFloat;

  BigInt          m;
  unsigned long   err;
  long            exp;

  unsigned int    refCount;

  //  private member functions come here.
};
```

Here, `BigInt` is the class of integers of arbitrary length. In the previous sections, we assume that the exponent $exp \in \mathbb{Z}$. In the implementation, however, we declare `exp` to be `long` for efficiency.

No member of `BigFloatRep` is declared to be public. Thus, the members of `BigFloatRep` can be accessed only via `BigFloat`.

The only data member of `BigFloat` is a pointer `rep` to the "letter" where the values of components are stored. The member or friend functions of `BigFloat` are implemented as implicit calls to the corresponding member functions of `BigFloatRep`. For example,

```
BigFloat x, y;
x + y;
```

is compiled as follows (see Figure 1):

The binary operator

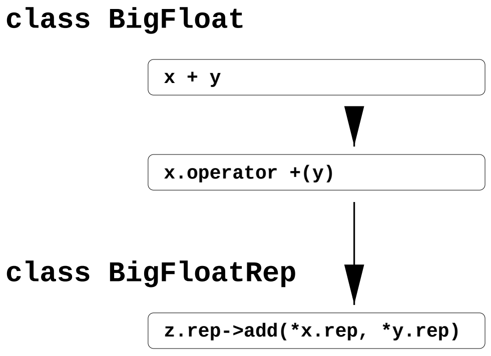


Figure 1: The flow for `x + y`.

```
BigFloat BigFloat :: operator +(const BigFloat) const
```

is called with the implicit argument `x` and the explicit argument `y`. It constructs `BigFloat z` where the result is going to be stored, and calls the member function of `BigFloatRep`

```
void BigFloatRep :: add(const BigFloatRep, const BigFloatRep)
```

with the implicit argument `*z.rep` and the explicit arguments `*x.rep` and `*y.rep`.

The "letter-envelope" technique allows us to do memory-management efficiently since multiple "envelopes" can share the single "letter". For example, consider the function

```
BigFloat BigFloat :: abs() const
```

which returns the absolute value of the `*this`. If `BigFloat x` has a non-negative mantissa then the values of the components of `x` and `x.abs()` are the same. Thus, `x.abs()` could be implemented as a `BigFloat` whose `rep` is identical to `x.rep` rather than a copy of `x`. Compared to a brute-force implementation, we now reduce the number of `BigInt` instances by one (see Figure 2).

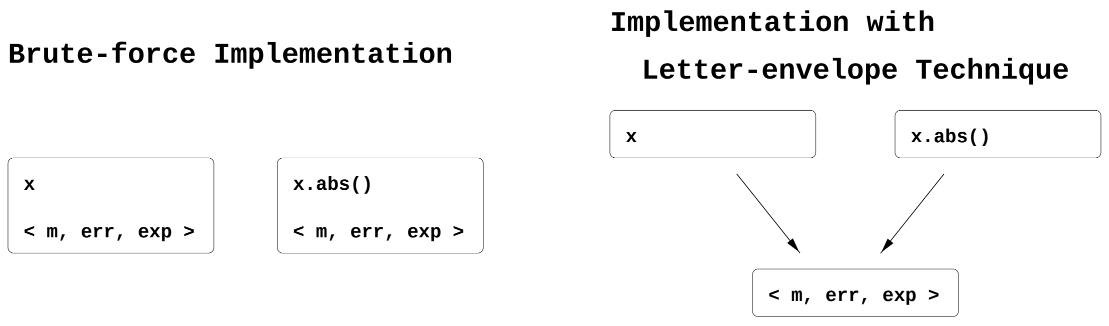

Brute-force Implementation
x
< m, err, exp >
x.abs()
< m, err, exp >
Implementation with Letter-envelope Technique
x
x.abs()
< m, err, exp >

Figure 2: If a brute-force implementation is used, the components of `x` and `x.abs()` are distinct, even though their components store the same values. If the "letter-envelope" technique is used, `x` and `x.abs()` can share their components.

The class `BigFloatRep` has the private data member `refCount` which counts how many "envelopes" points to `*this`. When an instance of `BigFloat x` is about to be destroyed, `x.rep->refCount` is decremented, and if it reaches 0 then `*x.rep` is also destroyed.

4 Real

In this section, we describe the `Real` package implemented as a class library in `C++`. In addition to the standard libraries of `C++`, we assume that we have a class library of integers of arbitrary length and a class library of rational numbers such as `GNU`'s `Integer` and `Rational` as well as our class `BigFloat`.

The class `Real` is used to represent numerical objects in the `Expr` package. The class has the ability to capture various types of number representations, namely, built-in machine types and some arbitrary length number types including `BigFloat`.

4.1 Definition

4.1.1 Construction

Define

$$\texttt{KernelType} = \{\texttt{int}, \texttt{long}, \texttt{double}, \texttt{BigInt}, \texttt{Rational}, \texttt{BigFloat}\}.$$

Here, `int`, `long` and `double` are the standard `C++` types, `BigInt` is the type of arbitrary length integers and `Rational` is the type of rational numbers.

A `Real` x is defined to be a numerical object X of type $t \in \texttt{KernelType}$.

We say X is *the kernel* of x.

4.1.2 Semantics

Any `Real` number is associated with a triple $\langle T, V, Err\rangle$ where

- $T \in \texttt{RealType} = (\texttt{KernelType} \setminus \{\texttt{BigFloat}\}) \cup \{\texttt{ExBigFloat}, \texttt{AppBigFloat}\}$,
- $V \in \mathbb{Q}$,
- $Err \in \mathbb{Q}_{\geq 0}$.

Here, `ExBigFloat` is the type of error-free `BigFloat` numbers and `AppBigFloat` is the type of `BigFloat` numbers with positive errors. As one would expect, there is a natural type coercion among the types in `RealType`. It is as follows:

$$\texttt{int} < \texttt{long} < \left\{ \begin{array}{c} \texttt{double} \\ \texttt{BigInt} \end{array} \right\} < \texttt{ExBigFloat} < \texttt{Rational} < \texttt{AppBigFloat}. \qquad (25)$$

Note `double` resides between `long` and `ExBigFloat`, that is, a `double` can be a kernel of some `Real` only if it is *exact*.

For any `Real` x, $\langle T_x, V_x, Err_x\rangle$ is set as follows:

- If the kernel of x is the `BigFloat` $X = \langle m_X, err_X, exp_X\rangle$ then

$$\begin{aligned} T_x &= \left\{ \begin{array}{ll} \texttt{ExBigFloat} & \text{if } err_X = 0 \\ \texttt{AppBigFloat} & \text{if } err_X > 0 \end{array} \right. \\ V_x &= m_X B^{exp_X} \\ Err_x &= err_X B^{exp_X}. \end{aligned}$$

- If the kernel of x is X whose type is $t \neq$ `BigFloat` then

$$\begin{aligned} T_x &= t \\ V_x &= X \\ Err_x &= 0. \end{aligned}$$

A `Real` x is said to be *error-free* if $Err_x = 0$. Hence, a `Real` x is error-free, unless T_x is `AppBigFloat`.

On the other hand, a `Real` x with $T_x =$ `AppBigFloat` is intended to be an approximation for some real number. A real number X is said to *belong to* a `Real` x if

$$X \in [V_x - Err_x, V_x + Err_x].$$

Let $(r,a) \in \mathbb{N} \times \mathbb{Z}$. A `Real` x is said to have *an error-bound* $[r,a]$ if

$$Err_x \leq \max\left\{|V_x|\, 2^{-r},\, 2^{-a}\right\}.$$

The most significant bit (MSB) μ_x of a `Real` x is defined to be

$$\begin{cases} \lfloor \lg |V_x| \rfloor & \text{if } V_x \neq 0 \\ -\infty & \text{if } V_x = 0. \end{cases}$$

4.2 Approximation

Let X be a real number and $(r,a) \in \mathbb{N} \times \mathbb{Z}$. We say a `Real` x *approximates* X *to precision* $[r,a]$ and write

$$x \cong X[r,a]$$

if X belongs to x and

$$Err_x \leq \max\left\{|X|\, 2^{-r},\, 2^{-a}\right\}.$$

If x approximates X then $V_x \cong X[r,a]$, i.e.,

$$|X - V_x| \leq \max\left\{|X|\, 2^{-r},\, 2^{-a}\right\}.$$

4.2.1 Approximation Algorithm

Given $R \in \mathbb{Q}$ and $(r,a) \in \mathbb{N} \times \mathbb{Z}$, we would like to compute a `Real` x with the error-bound $[r,a]$ such that R belongs to x.

If $r = a = \infty$ then x is the `Real` whose kernel is R.

Otherwise, x is the `Real` whose kernel is the `BigFloat` X with error-bound $[r,a]$ such that R belongs to x (defined in Section 3.2.1).

4.2.2 Properties

Proposition 11 *Fix $R \in \mathbb{Q}$ and $(r, a) \in \mathbb{N} \times \mathbb{Z}$. Let x be the* `Real` *computed by our approximation algorithm on input R and $[r, a]$.*

1.
$$|V_x| \leq |R|\,. \tag{26}$$

2.
$$x \cong R[r, a]. \tag{27}$$
In particular, $V_x \cong R[r, a]$, i.e., $|R - V_x| \leq \max\{|R|\,2^{-r},\, 2^{-a}\}$.

3. *If $R \neq 0$ then*
$$\mu_x = \lfloor \lg |R| \rfloor\,. \tag{28}$$

Proof. If $T_x \neq$ `ExBigFloat` or $T_x \neq$ `AppBigFloat` then the claims are trivial.

If $T_x =$ `ExBigFloat` or $T_x =$ `AppBigFloat` then the claims follow from Proposition 1 and 2. **Q.E.D.**

4.3 Arithmetic Operators and Squareroot

Over `Real`, the arithmetic operators $+$, (unary and binary) $-$, $\cdot$ and $/$ as well as the function $sqrt()$ are defined.

4.3.1 Unary Minus

Let x be `Real` whose kernel is X. We define $-x$ as follows:

1. Suppose $T_x =$ `int`. Note, in this case, $-X$ may not fit in `int`. Thus, $-x$ is the `Real` whose kernel is [3]
$$\begin{cases} -X & \text{if } -X \text{ fits in } \texttt{int} \\ -(\texttt{long})X & \text{if } -X \text{ does not fit in } \texttt{int}. \end{cases}$$
Here, $(t)X$ stands for X to which the casting operator () is applied so that the result has type t.

2. Suppose $T_x =$ `long`. Again, $-X$ may not fit in `long`. Thus, $-x$ is the `Real` whose kernel is
$$\begin{cases} -X & \text{if } -X \text{ fits in } \texttt{long} \\ -(\texttt{BigInt})X & \text{if } -X \text{ does not fit in } \texttt{long}. \end{cases}$$

3. Suppose $T_x \geq$ `double` or `BigInt`. Then, $-x$ is the `Real` whose kernel is $-X$.

[3] By definition, `int` $\subseteq$ `long`. If a system assumes that `int` $=$ `long` then `long` must be replaced by `BigInt`.

4.3.2 Unifier

We need the concept of unifiers to define the binary operators.

Let s and $t \in$ `RealType`. We say $u \in$ `RealType` *unifies* s and t if $s \leq u$ and $t \leq u$ where $\leq$ is defined as in (25). If u unifies s and t, u is called *a unifier* of s and t. For example, both `ExBigFloat` and `AppBigFloat` are the unifiers of `double` and `BigInt`.

The most general unifier (MGU) of s and t is defined to be $u \in$ `RealType` such that

- u unifies s and t,
- $\forall$ unifier u' of s and t, $u' \not< u$.

For example, the MGU of `double` and `BigInt` is `ExBigFloat`.

Note, as long as both s and $t \in$ `RealType`, their MGU is uniquely determined.

4.3.3 Addition, Subtraction and Multiplication

We now define the binary operators $+$, $-$ and $\cdot$ over `Real`.

Let x and y be `Real` whose kernels are X and Y, respectively. Further, let u be the MGU of T_x and T_y.

For $@ \in \{+, -, \cdot\}$, we define $x@y$ as follows:

1. Suppose $u =$ `int`, i.e., $T_x = T_y =$ `int`. Note @ defined in `int` is not overflow-free. Thus, $x@y$ is the `Real` whose kernel is

$$\begin{cases} X@Y & \text{if } X@Y \text{ fits in } \texttt{int} \\ (\texttt{long})X@(\texttt{long})Y & \text{if } X@Y \text{ does not fit in } \texttt{int} \text{ but fits in } \texttt{long} \\ (\texttt{BigInt})X@(\texttt{BigInt})Y & \text{if } X@Y \text{ does not fit in } \texttt{long}. \end{cases}$$

2. Suppose $u =$ `long`. Again, @ defined in `long` is not overflow-free. Thus, $x@y$ is the `Real` whose kernel is

$$\begin{cases} (\texttt{long})X@(\texttt{long})Y & \text{if } (\texttt{long})X@(\texttt{long})Y \text{ fits in } \texttt{long} \\ (\texttt{BigInt})X@(\texttt{BigInt})Y & \text{if } (\texttt{long})X@(\texttt{long})Y \text{ does not fit in } \texttt{long}. \end{cases}$$

3. Suppose $u =$ `double`. Then, @ defined in `double` is neither overflow-free nor underflow-free. Moreover, even when @ does not cause over/underflow, the result may be rounded (an exception "inexact" is caused). Since we assume that a *non-exact* `double` cannot be a kernel of `Real`, we must use @ in `BigFloat` when the result of @ in `double` is rounded. Thus, $x@y$ is the `Real` whose kernel is

$$\begin{cases} (\texttt{double})X@(\texttt{double})Y \\ \quad \text{if } (\texttt{double})X@(\texttt{double})Y \text{ does not cause exceptions:} \\ \qquad \text{overflow or underflow or inexact} \\ (\texttt{BigFloat})X@(\texttt{BigFloat})Y \\ \quad \text{if } (\texttt{double})X@(\texttt{double})Y \text{ causes an exception:} \\ \qquad \text{overflow or underflow or inexact.} \end{cases}$$

4. Suppose `BigInt` $\leq u \leq$ `Rational`. Set $v \in$ `KernelType` to be

$$\begin{cases} \texttt{BigFloat} & \text{if } u = \texttt{ExBigFloat} \\ u & \text{otherwise.} \end{cases}$$

Then, @ defined in v is overflow-free. Thus, $x@y$ is just the `Real` whose kernel is $(v)X@(v)Y$.

5. Suppose the one of T_x and T_y is `Rational` and the other is `AppBigFloat`. WLOG, we may assume that $T_x =$ `Rational` and $T_y =$ `AppBigFloat`. Although $u =$ `AppBigFloat`, we cannot simply say that $x@y$ is the `Real` whose kernel is (`BigFloat`)X@(`BigFloat`)Y, because `Rational` cannot be casted into `BigFloat`. Instead, we first compute an approximation $\widehat{x}$ of x to some precision so that $T_{\widehat{x}} =$ `AppBigFloat`, then we define $x@y$ to be the `Real` whose kernel is $\widehat{X}@Y$ where $\widehat{X}$ is the kernel of $\widehat{x}$.

   We now state how much precision is specified to get $\widehat{x}$.

   (a) If $@ = +$ or $-$ then $\widehat{x} \cong x\,[\infty, -\lfloor \lg Err_y \rfloor]$.
   Note

$$Err_{\widehat{x}} \quad \leq \quad Err_y, \tag{29}$$

   since $Err_{\widehat{x}} \leq 2^{\lfloor \lg Err_y \rfloor} \leq Err_y$.

   (b) If $@ = \cdot$ then $\widehat{x} \cong x\,[\max\{\lfloor \lg |V_y| \rfloor - \lfloor \lg Err_y \rfloor, 0\} + 1, \infty]$.
   Note, by (26),

$$|V_{\widehat{x}}|\, Err_y \quad \leq \quad |V_x|\, Err_y,$$

   and

$$Err_{\widehat{x}}\, |V_y| \quad \leq \quad |V_x|\, Err_y,$$

   since $Err_{\widehat{x}} \leq |V_x|\, 2^{-(\lfloor \lg |V_y| \rfloor + 1) + \lfloor \lg Err_y \rfloor} \leq \frac{|V_x| Err_y}{|V_y|}$, and

$$Err_{\widehat{x}} Err_y \quad \leq \quad |V_x|\, Err_y.$$

   because $Err_{\widehat{x}} \leq |V_x|\, 2^{-1}$. All together,

$$\max\{|V_{\widehat{x}}|\, Err_y,\ Err_{\widehat{x}}\, |V_y|\,,\ Err_{\widehat{x}} Err_y\} \quad \leq \quad |V_x|\, Err_y. \tag{30}$$

6. If $T_x = T_y =$ `AppBigFloat` then $x@y$ is the `Real` whose kernel is $X@Y$.

For example, if $x = \langle$`ExBigFloat`$, X, 0\rangle$ and $y = \langle$`Rational`$, R, 0\rangle$ then $x + y = \langle$`Rational`$, ($`Rational`$)\,X + R, 0\rangle$ where $+$ is the addition defined in `Rational`.

4.3.4 Division

To define the binary operator / over `Real`, we must clarify the semantics of /.

Let x and y be `Real` whose kernels are X and Y, respectively. We would like to define $z = x/y$ so that $x = y \cdot z$. Unfortunately, in some types, the operator / does not satisfy this condition. For example, if X and Y are both `int` then X/Y is $\operatorname{sgn}(X\,Y)\left\lfloor\left|\frac{X}{Y}\right|\right\rfloor$ which is different from $\frac{X}{Y}$ (unless $Y|X$). Hence, we first cast X and Y to be `Rational` or `BigFloat` where / is appropriately defined.

Let u be the MGU of T_x and T_y. We define x/y as follows:

1. If $u \in \{$`int`, `long`, `BigInt`, `Rational`$\}$ then x/y is the `Real` whose kernel is (`Rational`)X/(`Rational`)Y.

2. Suppose $u =$ `double` or `ExBigFloat`. Then, x/y is the `Real` whose kernel is (`BigFloat`)X/(`BigFloat`)Y. We could use the division in `Rational`, but constructing `Rational` from `double` or `BigFloat` is expensive, in general.

3. Suppose that one of T_x and T_y is `Rational` and the other is `AppBigFloat`. WLOG, we may assume that $T_x =$ `Rational` and $T_y =$ `AppBigFloat`. Again, instead of casting X into `BigFloat`, x must be approximated by $\widehat{x}$ so that $T_{\widehat{x}} =$ `AppBigFloat`. We set $\widehat{x} \cong x\left[\max\{\lfloor\lg|V_y|\rfloor - \lfloor\lg Err_y\rfloor, 0\} + 1, \infty\right]$ and $\widehat{X}$ to be the kernel of $\widehat{x}$. Then, x/y is the `Real` whose kernel is $\widehat{X}/Y$.

   Note

   $$\frac{\left|V_{\widehat{x}}\right|}{|V_y|}\frac{Err_{\widehat{x}}}{\left|V_{\widehat{x}}\right|} \;\le\; \frac{|V_x|}{|V_y|}\frac{Err_y}{|V_y|},$$

   since $Err_{\widehat{x}} \le |V_x|\, 2^{-(\lfloor\lg|V_y|\rfloor+1)+\lfloor\lg Err_y\rfloor} \le \frac{|V_x| Err_y}{|V_y|}$, and

   $$\frac{\left|V_{\widehat{x}}\right|}{|V_y|}\frac{Err_y}{|V_y|} \;\le\; \frac{|V_x|}{|V_y|}\frac{Err_y}{|V_y|},$$

   by (26). Hence

   $$\frac{\left|V_{\widehat{x}}\right|}{|V_y|}\max\left\{\frac{Err_{\widehat{x}}}{\left|V_{\widehat{x}}\right|}, \frac{Err_y}{|V_y|}\right\} \;\le\; \frac{|V_x|}{|V_y|}\frac{Err_y}{|V_y|}. \qquad (31)$$

4. If $T_x = T_y =$ `AppBigFloat` then x/y is the `Real` whose kernel is X/Y.

4.3.5 Squareroot

Let x be `Real` whose kernel is X. Since there is no type where $\sqrt{X}$ can be computed exactly, we use the function `BigFloat` *sqrt* (`BigFloat` $\langle m_X, err_X, exp_X\rangle$) which computes $\sqrt{\langle m_X, err_X, exp_X\rangle}$ for $m_X \ge 0$ (see Section 3.6).

1. Unless $T_x =$ `Rational` then $sqrt\,(x)$ is defined to be the `Real` z whose kernel is $sqrt\,((\texttt{BigFloat})\,X)$.

2. If $T_x =$ `Rational` then, once again, x must be approximated by $\widehat{x}$ so that $T_{\widehat{x}} =$ `AppBigFloat`. We set $\widehat{x} \cong x\,[\infty, 2\,a_{\mathtt{default}} + 8]$ and $\widehat{X}$ to be the kernel of $\widehat{x}$. Then, $sqrt\,(x)$ is defined to be the `Real` z whose kernel is $sqrt\left(\widehat{X}\right)$.

   Note

$$\sqrt{Err_{\widehat{x}}} \;\; \leq \;\; 2^{-a_{\mathtt{default}}-4}. \tag{32}$$

4.3.6 Properties

Proposition 12 *Let x and y be* `Real`.

1. *If a real X belongs to x then $-X$ belongs to $-x$.*
2. *For $@ \in \{+, -, \cdot, /\}$, if a real X belongs to x and a real Y belongs to y then $X@Y$ belongs to $x@y$.*
3. *If a real $X \geq 0$ belongs to x then $\sqrt{X}$ belongs to $sqrt\,(x)$.*

Proof. Clear from definitions. **Q.E.D.**

Proposition 13 *Let x, y and z be* `Real`.

1. *If $z = x \pm y$ then*

$$Err_z \;\; \leq \;\; 6\,\max\{Err_x,\, Err_y\}. \tag{33}$$

2. *If $z = x \cdot y$ then*

$$Err_z \;\; \leq \;\; 6\,\max\{|V_x|\,Err_y,\, Err_x\,|V_y|\,,\, Err_x Err_y\}. \tag{34}$$

3. *Suppose $z = x/y$.*

   (a) *If $Err_x = Err_y = 0$ then*

$$Err_z \;\; \leq \;\; \frac{|V_x|}{|V_y|}\,2^{-r_{\mathtt{default}}} \tag{35}$$

   where $r_{\mathtt{default}}$ is some global constant which users can change.

   (b) *If either $Err_x = 0$ and $\frac{|V_y|}{2} \geq Err_y > 0$*
   or $|V_x| > Err_x > 0$ and $Err_y = 0$
   or $|V_x| > Err_x > 0$ and $\frac{|V_y|}{2} \geq Err_y > 0$ then

$$Err_z \;\; \leq \;\; 12\,\frac{|V_x|}{|V_y|}\max\left\{\frac{Err_x}{|V_x|},\, \frac{Err_y}{|V_y|}\right\}. \tag{36}$$

4. *Suppose $z = sqrt\,(x)$.*

(a) If $Err_x = 0$ then

$$Err_z \leq 2^{-a_{\texttt{default}}} \tag{37}$$

where $a_{\texttt{default}}$ is some global constant which users can change.

(b) If $Err_x > 0$ then

$$Err_z \leq 16\sqrt{Err_x}. \tag{38}$$

Proof. If $T_z \neq$ `ExBigFloat` or $T_z \neq$ `AppBigFloat` then the claims are trivial.

If either $T_z =$ `ExBigFloat` or $T_z =$ `AppBigFloat` but neither T_x nor T_y is `Rational` then the claims follow from Proposition 3, 4, 5 and 6.

Otherwise one of T_x and T_y is `Rational` and the other is `AppBigFloat`. WLOG, we may assume $T_x =$ `Rational` and $T_y =$ `AppBigFloat`. In this case, x is approximated by $\widehat{x}$ and z is defined to be the result of the `BigFloat` operation applied to $\widehat{x}$ and y.

1. If $z = x \pm y$ then

$$\begin{aligned} Err_z &\leq 6 \max\{Err_{\widehat{x}}, Err_y\} && \text{(by Proposition 3)} \\ &\leq 6\, Err_y. && \text{(by (29))} \end{aligned}$$

2. If $z = x \cdot y$ then

$$\begin{aligned} Err_z &\leq 6 \max\{|V_{\widehat{x}}|\, Err_y, Err_{\widehat{x}}\, |V_y|\,, Err_{\widehat{x}} Err_y\} && \text{(by Proposition 4)} \\ &\leq 6\, |V_x|\, Err_y. && \text{(by (30))} \end{aligned}$$

3. If $z = x/y$ and $\frac{|V_y|}{2} \geq Err_y > 0$ then

$$\begin{aligned} Err_z &\leq 12 \frac{|V_{\widehat{x}}|}{|V_y|} \max\left\{\frac{Err_{\widehat{x}}}{|V_{\widehat{x}}|}, \frac{Err_y}{|V_y|}\right\} && \text{(by Proposition 5)} \\ &\leq 12 \frac{|V_x|}{|V_y|} \frac{Err_y}{|V_y|}. && \text{(by (31))} \end{aligned}$$

4. If $z = sqrt\,(x)$ then

$$\begin{aligned} Err_z &\leq 16\sqrt{Err_{\widehat{x}}} && \text{(by Proposition 6)} \\ &\leq 2^{-a_{\texttt{default}}}. && \text{(by (32))} \end{aligned}$$

Q.E.D.

4.4 Implementation

For the implementation, we use the class inheritance scheme of the `C++` language, as well as the "letter-envelope" technique. From the class `Real`, we derive several classes, each of which corresponds to the number types that `Real` incorporates.

```
class Real
{
  Real* rep;

  //  other members here.
};

class RealInt : public class Real
{
friend class RealLong;
friend class RealDouble;
friend class RealBigInt;
friend class RealBigFloat;
friend class RealRational;

  int ker;  // kernel

  //  other members come here.
};

//  other inherited classes come here.
```

Now, consider the following program segment:

```
double X = 1.0;
BigInt Y = 1;
Real   x = X;
Real   y = Y;
x + y;
```

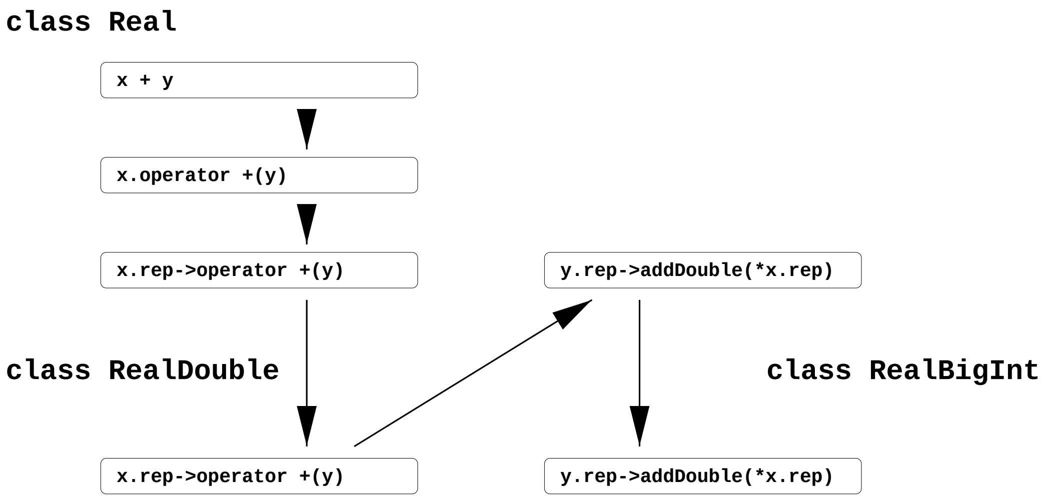


Figure 3: The flow for `x + y`.

Then, `x + y` is compiled as follows (see Figure 3):

1. The binary operator

   ```
   Real Real :: operator +(const Real) const
   ```

   is called with the implicit argument `x` and the explicit argument `y`.

2. Determine T_x. The operator `+` applied to the "envelope" `x` calls the operator `+` for its "letter" `*x.rep`:

   ```
   virtual Real Real :: operator +(const Real) const
   ```

   is called with the implicit argument `*x.rep` and the explicit argument `y`. By the virtual function mechanism, the compiler finds that $T_x =$ `double`, and actually

   ```
   Real RealDouble :: operator +(const Real) const
   ```

   is called.

3. Determine T_y. "Swap" the implicit and explicit arguments and do the same as before: the member operator `+` of `RealDouble` calls

   ```
   virtual Real Real :: addDouble(const RealDouble) const
   ```

   with the implicit argument `*y.rep` and the explicit argument `*x.rep`. Again, by the virtual function mechanism, the compiler finds that $T_y =$ `BigInt`, and actually

   ```
   Real RealBigInt :: addDouble(const RealDouble) const
   ```

   is called.

4. Now, `x + y` turns out to be an addition for `double` and `BigInt`. Since the MGU of `double` and `BigInt` is `ExBigFloat`,

   ```
   Real(BigFloat(*x.rep) + BigFloat(*y.rep)).
   ```

   is returned.

5 Expr

In this section, we describe the `Expr` package implemented as a class library in `C++`. In addition to the standard library of `C++`, we assume that we have our class `Real`.

The class `Expr` captures a set of algebraic expressions.

5.1 Definition

5.1.1 Node of `Expr` Tree and Exact Value

An `Expr` e is a node of some rooted DAG. If `Expr` e is a leaf of some rooted DAG then it is

- *a parameter node* which can store some value in $\mathbb{Q}$.

If `Expr` e is an internal node of some rooted DAG then it is either

- *a unary minus node* which has one child f and represents $-f$, or
- *a binary operator node* which has two children f and g and represents $f@g$ for $@ \in \{+, -, \cdot, /\}$, or
- *a squareroot node* which has one child f and represents $\sqrt{f}$.

A non parameter node is called *an operator node.*

Let e and f be `Expr`. The tree rooted at e and the tree rooted at f could share some subtrees. Hence, a single node may have several parent nodes. This is why we define an instance of `Expr` to be a node of some DAG, and not a node of some simple tree.

Any `Expr` e is associated with *an exact value* $\check{e}$ defined as follows:

1. Suppose e is a parameter node. If e stores a value $x \in \mathbb{Q}$ then $\check{e} = x$. Otherwise, $\check{e}$ is an indeterminate value denoted ω_e.
2. If e is a unary minus node whose child is f then $\check{e} = -\check{f}$.
3. If e is a binary operator node which has two children f and g and represents $f@g$ for $@ \in \{+, -, \cdot, /\}$ then $\check{e} = \check{f}@\check{g}$.
4. If e is a squareroot node whose child is f then $\check{e} = -\check{f}$.

If every leaf of the tree rooted at e has the exact value in $\mathbb{Q}$ then $\check{e}$ is the element of some algebraically closed filed $\mathbb{D}$ containing $\mathbb{Q}$.

In this paper, we may use the same symbol e for an instance of `Expr`, a tree rooted at e and its exact value. The context should make our intent clear. For example, in the statement "if e is a parameter then $e \in \mathbb{Q}$", the first e is meant to be the instance of `Expr` and the second e is the exact value of the instance.

5.1.2 Approximation

Let e be `Expr` and $(r, a) \in \mathbb{N} \times \mathbb{Z}$. We say `Real` $\widehat{e}$ approximates e to precision $[r, a]$ and write

$$\widehat{e} \cong e[r, a]$$

if e belongs to $\widehat{e}$ and

$$Err_{\widehat{e}} \;\; \leq \;\; \max\left\{ |e|\, 2^{-r},\, 2^{-a} \right\}.$$

If $\widehat{e} \cong e[r,a]$ then $V_{\widehat{e}} \cong e[r,a]$, i.e.,

$$|e - V_{\widehat{e}}| \leq \max\left\{|e|\,2^{-r},\, 2^{-a}\right\}.$$

Each `Expr` e maintains

- *an approximate value* $\widehat{e}$ in `Real`, and
- *precision* $[r,a]$ where $r \in \mathbb{N}$ and $a \in \mathbb{Z}$

so that $\widehat{e} \cong e[r,a]$. Precision can be set explicitly by users or implicitly by some function call. Whenever precision $[r,a]$ of `Expr` e is specified, we recompute the approximate value $\widehat{e}$ of e so that $\widehat{e} \cong e[r,a]$.

5.1.3 Semantics of Assignments

In the `Expr` package, assignment is somewhat subtle as we now explain.

The copy rule of `C++` is "pass by value". Thus, the assignment `x = y` assigns the current actual value of `y` to the actual value of `x`. We do not want to apply this copy rule to assignment operators over instances of `Expr`. Consider the following program:

```
Expr a, b, c;
Expr D = b * b - 4 * a * c;
// at this point, the exact value of D is $\omega_b^2 - 4\,\omega_a\omega_c$.

a = 3;
b = 7;
c = 3;
// at this point, the exact value of D is still $\omega_b^2 - 4\,\omega_a\omega_c$,
// although we expect it to be 13.
```

If we follow the standard semantics of `C++`, then the exact value of `D` at the end of the program is $\omega_b^2 - 4\,\omega_a\omega_c$. We would like to have special semantics where the exact value of e becomes 13 at the end of the program.

Define the semantics of the assignment operator `=` for `Expr` so that the following holds:

Fix a scope $\mathcal{S}$. Let e, f and g be `Expr` and x be `Real`. Further, let Φ be an algebraic expression which involves $+$, (unary and binary) $-$, $\cdot$, $/$ and $\sqrt{}$.

1. Suppose, in $\mathcal{S}$, there are statements of the form

$$e = \Phi(f) \qquad (39)$$
$$f = x. \qquad (40)$$

   If (40) precedes (39), and in between (40) and (39) there is no assignment statement whose left operand is f then, as in the standard `C++` semantics, the exact value of e becomes $\Phi(x)$ when (39) is stated. If (39) precedes (40), and in between (39) and (40) there is no assignment statement whose left operand is e then, unlike the standard `C++` semantics, the exact value of e becomes $\Phi(x)$ when (40) is stated.

2. Suppose, in $\mathcal{S}$, there are statements of the form

$$e = \Phi(f) \tag{41}$$
$$f = g. \tag{42}$$

   If (42) precedes (41), and in between (42) and (41) there is no assignment statement whose left operand is f then, as in the standard `C++` semantics, the exact value of e becomes the exact value of $\Phi(g)$ when (41) is stated. If (41) precedes (42), and in between (41) and (42) there is no assignment statement whose left operand is e then, unlike the standard `C++` semantics, the exact value of e becomes the exact value of $\Phi(g)$ when (42) is stated.

Note our new semantics for the assignment operator causes a side-effect:

Fix a scope $\mathcal{S}$. Let e and f be `Expr` and Φ be an algebraic expression which involves $+$, (unary and binary) $-$, $\cdot$, $/$ and $\sqrt{}$. Suppose, in $\mathcal{S}$, there is a statement of the form

$$e = \Phi(f).$$

Then, in the rest of $\mathcal{S}$, until e is assigned to be something else, whenever the assignment operator whose left operand is f is stated, the exact value of e is changed.

The assignment of the form $e = \Phi(e)$ is not defined. Also, the operators `+=`, `-=`, `*=` and `/=` are not defined.

5.2 Implementation

To realize the semantics described above, again, we use the full power of the "letter-envelope" technique.

There are two basic classes, the class `Expr` for "envelopes" and the class `ExprRep` for "letters". From `ExprRep`, we derive three classes, `ParamRep`, `UnaryOpRep` and `BinOpRep`. From `UnaryOpRep`, we derive two classes, `NegRep` and `SqrtRep`. From `BinOpRep`, we derive four classes, `AddRep`, `SubRep`, `MultRep` and `DivRep`. (See Figure 4.) It is clear what each of those classes represents.

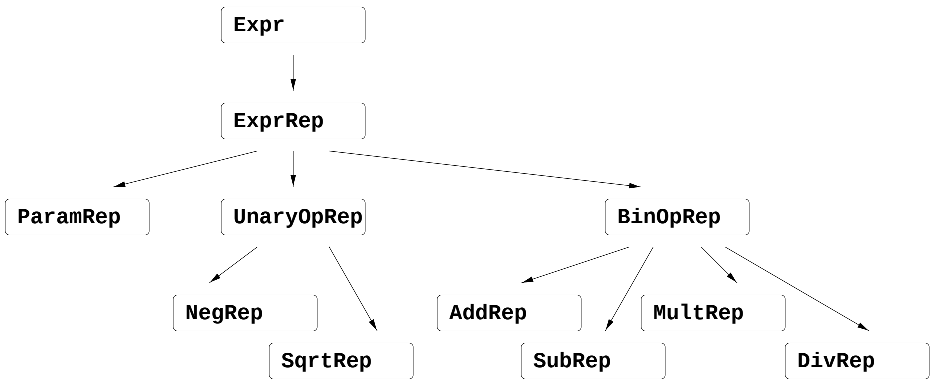


Figure 4: `Expr`, `ExprRep` and the classes inherited from `ExprRep`

The class `ExprRep` is derived from the class `Expr`. Thus, an instance of the class `ExprRep` could point to another instance of the class `ExprRep`.

```
class Expr
{
protected:

  Expr* rep;

  //  other members come here.
};

class ExprRep : public class Expr
{
friend class Expr;
friend class ParamRep;
friend class UnaryOpRep;
friend class BinOpRep;

private:

  unsigned refCount;

protected:

  Real appValue;  //  approximate value

  //  other members come here.
};

//  the inherited classes come here.
```

5.2.1 Node

A node of `Expr` tree is realized as a chain that consists of 0 or 1 "envelope" (`Expr`) followed by 1 or more "letter(s)" (`ExprRep`) (see Figure 5).

The last letter in the chain specifies the type of the node, e.g., if the last letter in the chain is `NegRep` then the chain represents a unary minus node. The approximate value and the precision of the chain reside in the last letter.

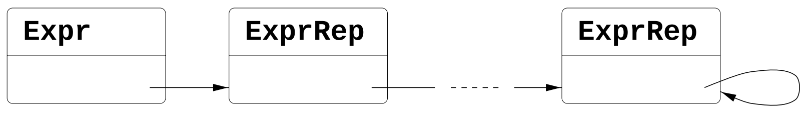
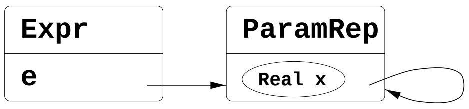


Figure 5: The chain of `Expr`. Arrows indicate `rep` pointers.

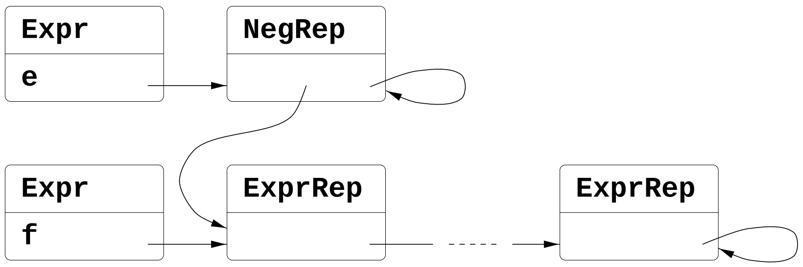
The last letter in the chain is characterized as an `ExprRep` where `rep == this` holds. Hence, given a chain of `Expr`, its last letter is detected as follows: starting from any instance of `Expr` or `ExprRep` in the chain, follow the chain until reaching the instance where `rep == this` holds.

5.2.2 Construction

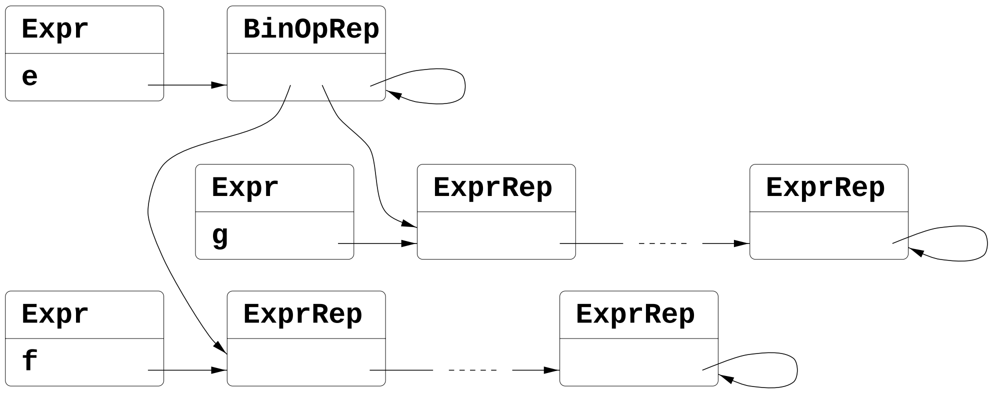
A new parameter node `e` which stores $x \in \mathbb{Q}$ is a chain of one `Expr` and one `ParamRep` which contains `Real` x.



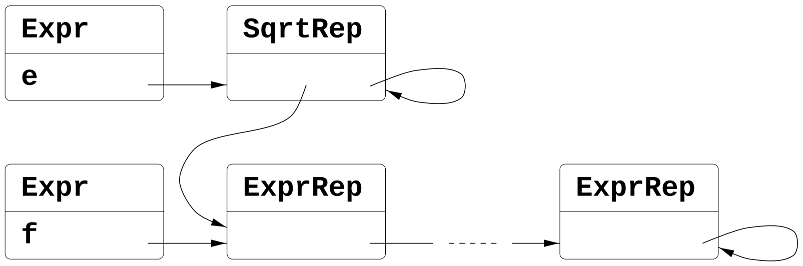
A new unary minus node `e` which represents $-$`f` is a chain of one `Expr` and one `NegRep` whose child is `*f.rep`.



A new binary operator node `e` which represents `f`@`g` is a chain of one `Expr` and one letter of some derived class of `BinOpRep` (depending on @) whose children are `*f.rep` and `*g.rep`.



A new squareroot node `e` which represents $\sqrt{\texttt{f}}$ is a chain of one `Expr` and one `SqrtRep` whose child is `*f.rep`.

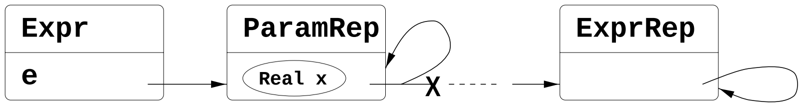


5.2.3 Assignment

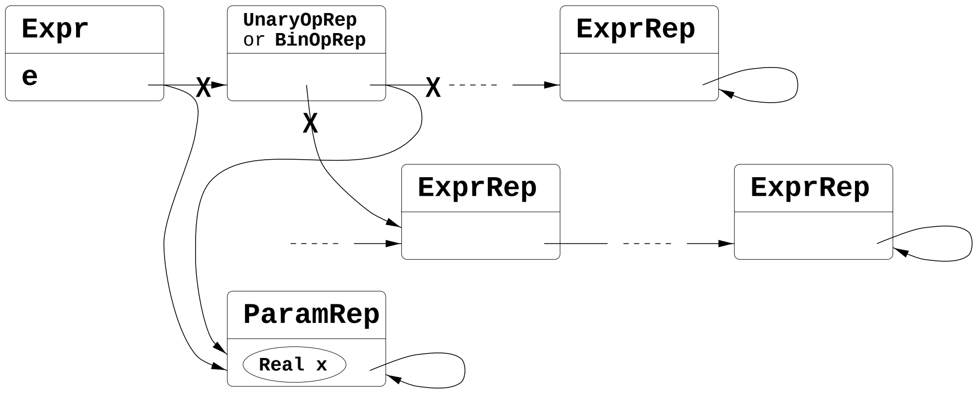
Let e and f be `Expr` and x be `Real`.

The assignment operation `e = x` is done as follows:

1. Suppose `*e.rep` is `ParamRep`. Then, `*e.rep` can store `x`. We cut the chain headed by e at `*e.rep` and destroy the instances in the chain headed by `e.rep->rep` (if exist), and set `e.rep->exValue` to be `x`.

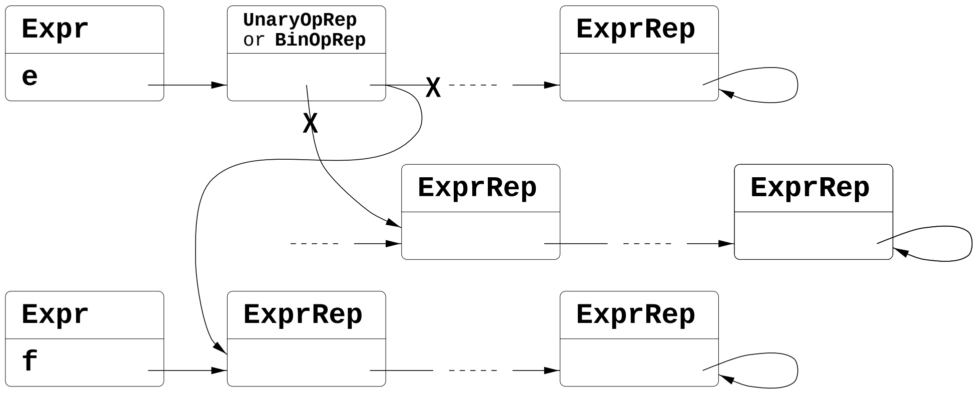


2. Suppose `*e.rep` is `UnaryOpRep` or `BinOpRep`. Then, `e.rep` cannot store `x`. First, we cut the chain headed by e at the `*e.rep` and destroy the instances in the chain headed by `e.rep->rep` (if exist). Then, we cut the link(s) from to its child(ren) (if exists) and destroy the chain(s) headed by `*e.rep`'s child(ren). Finally, we construct a new `ParamRep` which will store `x`, and make both `e` and the current `*e.rep` point to this newly constructed `ParamRep`.



The assignment operation `e = f` is done as follows:

First, we cut the chain headed by e at the `*e.rep` and destroy the instances in the chain headed by `e.rep->rep` (if exist). Then, we cut the link(s) from to its child(ren) (if exist(s)) and destroy the chain(s) headed by `*e.rep`'s child(ren). Then, we set `e.rep->rep` to be `*f.rep`.



6 Root Bound

In this section, we describe our algorithm to determine whether or not a given `Expr` is *exactly* 0. The algorithm is based on the theory of the root bounds for polynomials over an algebraically closed field. The missing proofs for the theorems are found in [Yap97].

6.1 Notations

Fix an algebraically closed field $\mathbb{D}$. Any `Expr` e can be viewed as an element of $\mathbb{D}$, i.e., $\exists E(X) \in \mathbb{D}[X]$ such that $E(e) = 0$.

Write

$$E(X) = \sum_{i=0}^{m} e_i X^i \qquad \text{where } e_m \neq 0.$$

We say $E(X)$ is of *degree* m and write $\deg E = m$. *The leading coefficient* of $E(X)$ is e_m.

Let $\alpha_1, \ldots, \alpha_m \in \mathbb{D}$ (not necessarily distinct) be all the roots of $E(X)$. Then

$$E(X) = e_m \prod_{i=1}^{m} (X - \alpha_i).$$

Define

$$\begin{aligned} \|E\|_1 &= \sum_{i=0}^{m} |e_i| \\ \|E\|_2 &= \sqrt{\sum_{i=0}^{m} |e_i|^2} \\ \|E\|_\infty &= \max\{|e_0|, \cdots, |e_m|\}. \end{aligned}$$

Note $\|E\|_1 \geq \|E\|_2 \geq \|E\|_\infty$.

6.2 Root Bound

In this subsection, we describe Landau's root bound theorem and introduce our algorithm to determine whether or not a given `Expr` e is *exactly* 0, provided the 2-norm of $E(X) \in \mathbb{D}[X]$ such that $E(e) = 0$ is known.

6.2.1 Landau's Root Bound

We start from Landau's root bound theorem which gives us an upper bound for the magnitude of any root of $E(X)$:

Theorem 14 (Landau) *For any root α of $E(X) = \sum_{i=0}^{m} e_i X^i \in \mathbb{D}[X]$ with $e_m \neq 0$,*

$$|\alpha| \leq \frac{\|E\|_2}{|e_m|}.$$

Let $E(X) = \sum_{i=0}^{m} e_i X^i \in \mathbb{D}[X]$ with $e_m \neq 0$. Define *the tail coefficient* of $E(X)$ to be e_t which satisfies

$$e_t \neq 0 \quad \text{and} \qquad e_i = 0 \quad i = 0, \ldots, t-1.$$

Such t always exists, since $e_m \neq 0$. Obviously, $t \leq m$.

Lemma 15 *Let $E(X) = \sum_{i=0}^{m} e_i X^i$ with $e_m \neq 0$ be a polynomial in $\mathbb{D}[X]$ whose tail coefficient is e_t. Then, $E(X)$ has a non-zero root in $\mathbb{D}$ iff $t < m$.*

Proof. We show that the only root of $E(X)$ is 0 iff $t = m$.

If the only root of $E(X)$ is 0 then $E(X) = e_m X^m$.

Conversely, the equation $e_m X^m = 0$ where $e_m \neq 0$ has the only solution 0 over the integral domain $\mathbb{D}$. **Q.E.D.**

Theorem 16 *Let $E(X) = \sum_{i=0}^{m} e_i X^i$ with $e_m \neq 0$ be a polynomial in $\mathbb{D}[X]$ whose tail coefficient is e_t. For any non-zero root α of $E(X)$*

$$|\alpha| \geq \frac{|e_t|}{||E||_2}.$$

Proof. Define

$$F(X) = X^m E\left(\frac{1}{X}\right) = X^m \sum_{i=t}^{m} e_i \left(\frac{1}{X}\right)^i = \sum_{j=0}^{m-t} e_{m-j} X^j.$$

Suppose $E(X)$ has a non-zero root. By lemma 15, $t < m$. Hence, $\deg F = m - t \geq 1$, and for any non-zero root α of $E(X)$, $\frac{1}{\alpha}$ is a root of $F(X)$.

Since $||F||_2 = ||E||_2$ and the leading coefficient of $F(X)$ is the tail coefficient of $E(X)$, applying Landau's root bound for $\frac{1}{\alpha}$ yields

$$\left|\frac{1}{\alpha}\right| \leq \frac{||E||_2}{|e_t|},$$

or equivalently

$$|\alpha| \geq \frac{|e_t|}{||E||_2}.$$

Q.E.D.

Corollary 17 *Let α be a root of $E(X) \in \mathbb{Z}[X]$. Then*

$$\alpha \neq 0 \quad \textit{iff} \quad |\alpha| \geq \frac{1}{||E||_2}.$$

Proof. The sufficient condition is trivial.

The necessary condition is immediate from Theorem 16. **Q.E.D.**

6.2.2 Algorithm

Let e be `Expr` and $E(X) \in \mathbb{Z}[X]$ such that $E(e) = 0$. Define *a length bound* l_e of e to be a positive integer which satisfies

$$l_e \geq \lfloor \lg ||E||_2 \rfloor .$$

Note $\frac{1}{||E||_2} > 2^{-l_e - 1}$.

Proposition 18 *$e = 0$ iff 0 belongs to $\widehat{e}$ where $\widehat{e} \cong e\,[\infty, l_e + 2]$.*

Proof. The necessary condition is trivial.

Suppose 0 belongs to $\widehat{e}$, i.e., $|V_{\widehat{e}}| \leq Err_{\widehat{e}}$. Then

$$|e| \;\leq\; |e - V_{\widehat{e}}| + |V_{\widehat{e}}| \;\leq\; 2Err_{\widehat{e}}.$$

Since $\widehat{e} \cong e\,[\infty, l_e + 2]$, $Err_{\widehat{e}} \leq 2^{-l_e - 2}$. Hence

$$|e| \;\leq\; 2^{-l_e - 1} \;<\; \frac{1}{||E||_2}.$$

By Corollary 17, $e = 0$. **Q.E.D.**

Proposition 18 suggests the algorithm to determine whether or not a given `Expr` e is *exactly* 0. We simply compute $\widehat{e} \cong e\,[\infty, l_e + 2]$. If $|V_{\widehat{e}}| \leq Err_{\widehat{e}}$ then e is *exactly* 0. Otherwise, $e \neq 0$.

6.3 Resultant

To invoke our algorithm to determine whether or not a given `Expr` e is *exactly* 0, we must calculate the length bound l_e of $E(X) \in \mathbb{Z}[X]$ such that $E(e) = 0$. In this subsection, we describe the method of finding such an $E(X)$.

6.3.1 Sylvester Resultant

Let $F(X)$ and $G(X)$ be polynomials in $\mathbb{D}[X]$ of degree m and n, respectively. Write $F(X) = \sum_{i=0}^{m} f_i X^i$ and $G(X) = \sum_{i=0}^{n} g_i X^i$ where $f_m g_n \neq 0$.

The Sylvester matrix $\mathrm{syl}_X(F, G)$ of F and G with respect to X is the $(m+n)$ dimensional square matrix which is defined to be

$$\begin{pmatrix}
f_m & f_{m-1} & & & \cdots & f_0 & & & & \\
 & f_m & & & \cdots & f_1 & f_0 & & & \\
 & & \ddots & & \cdots & & & & \ddots & \\
 & & & f_m & \cdots & & & & & f_0 \\
g_n & g_{n-1} & & & \cdots & & & g_0 & & \\
 & g_n & & & \cdots & & & g_1 & g_0 & \\
 & & \ddots & & \cdots & & & & & \ddots \\
 & & & g_n & \cdots & & & & & g_0
\end{pmatrix}$$

The Sylvester resultant $\operatorname{res}_X(F,G)$ of F and G with respect to X is defined to be

$$\det\left(\operatorname{syl}_X(F,G)\right).$$

Let $\beta_1,\ldots,\beta_m$ and $\gamma_1,\ldots,\gamma_n$ be the roots of F and G, respectively.

The following lemma is a well-known property of the Sylvester resultant which is sometimes used as an alternative definition of the Sylvester resultant.

Lemma 19 (Poisson's Definition for Resultant)

$$\operatorname{res}_X(F,G) = f_m^n \prod_{i=1}^{m} G(\beta_i) = g_n^m \prod_{j=1}^{n} F(\gamma_j) = f_m^n g_n^m \prod_{i=1}^{m}\prod_{j=1}^{n}(\beta_i-\gamma_j).$$

Theorem 20

1.

$$\operatorname{res}_Y\left(F(X\mp Y),G(Y)\right) = f_m^n g_n^m \prod_{i=1}^{m}\prod_{j=1}^{n}\left(X-(\beta_i\pm\gamma_j)\right). \tag{43}$$

2.

$$\operatorname{res}_Y\left(Y^m F\left(\frac{X}{Y}\right),G(Y)\right) = f_m^n g_n^m \prod_{i=1}^{m}\prod_{j=1}^{n}(X-\beta_i\gamma_j). \tag{44}$$

Proof.

1.

$$\begin{aligned}
\operatorname{res}_Y\left(F(X\mp Y),G(Y)\right) &= g_n^m \prod_{j=1}^{n} F(X\mp\gamma_j)\\
&= g_n^m \prod_{j=1}^{n}\left(f_m\prod_{i=1}^{m}(X\mp\gamma_j-\beta_i)\right)\\
&= f_m^n g_n^m \prod_{i=1}^{m}\prod_{j=1}^{n}\left(X-(\beta_i\pm\gamma_j)\right).
\end{aligned}$$

2.

$$\begin{aligned}
\operatorname{res}_Y\left(Y^m F\left(\frac{X}{Y}\right),G(Y)\right) &= g_n^m \prod_{j=1}^{n}\gamma_j^m F\left(\frac{X}{\gamma_j}\right)\\
&= g_n^m \prod_{j=1}^{n}\left(\gamma_j^m f_m \prod_{i=1}^{m}\left(\frac{X}{\gamma_j}-\beta_i\right)\right)\\
&= f_m^n g_n^m \prod_{j=1}^{n}\prod_{i=1}^{m}\left(\gamma_j\left(\frac{X}{\gamma_j}-\beta_i\right)\right)\\
&= f_m^n g_n^m \prod_{i=1}^{m}\prod_{j=1}^{n}(X-\beta_i\gamma_j).
\end{aligned}$$

Q.E.D.

Proposition 21 *Algebraic numbers are closed under taking the inverse, addition and multiplication. In fact, algebraic numbers form a field.*

Proof. Let β and $\gamma \in \mathbb{D}$. Then, $\exists F(X)$ and $G(X) \in \mathbb{Z}[X]$ so that $F(\beta) = G(\gamma) = 0$. If $\beta \neq 0$ then $\frac{1}{\beta}$ is a root of

$$X^{\deg F} F\left(\tfrac{1}{X}\right) \tag{45}$$

(see the proof for Theorem 16). By (43), $\beta \pm \gamma$ is a root of

$$\mathrm{res}_Y \left(F(X \mp Y), G(Y)\right). \tag{46}$$

By (44), $\beta\gamma$ is a root of

$$\mathrm{res}_Y \left(Y^{\deg F} F\left(\tfrac{X}{Y}\right), G(Y)\right). \tag{47}$$

Finally, if $F(X)$ and $G(X)$ are both in $\mathbb{Z}[X]$ then so are (45), (46) and (47), because of their constructions. **Q.E.D.**

6.3.2 Algorithms

Fix any `Expr` e. We would like to find $E(X) \in \mathbb{Z}[X]$ such that $E(e) = 0$. They are computed recursively by traversing the `Expr` tree e bottom-up from the leaves to the root e.

1. Suppose e is a leaf. Then, the exact value $e \in \mathbb{Q}$ is known. Writing $e = \frac{p}{q}$ where $(p, q) \in \mathbb{Z} \times \mathbb{Z}_{\neq 0}$ with $\gcd(p, q) = 1$, we find that

$$E(X) = qX - p. \tag{48}$$

2. Suppose e is of the form $e = -f$ for some `Expr` f. By assumption, $F(X) = \sum_{i=0}^{m} f_i X^i \in \mathbb{Z}[X]$ with $f_m \neq 0$ such that $F(f) = 0$ is known. Then

$$E(X) = F(-X) = \sum_{i=0}^{m} (-1)^i f_i X^i. \tag{49}$$

   The correctness is obvious.

3. Suppose e is of the form $e = f@g$ for some `Expr` f and g and for some $@ \in \{+, -, \cdot, /\}$. By assumption, $F(X)$ and $G(X) \in \mathbb{Z}[X]$ such that $F(f) = G(g) = 0$ are known.

   (a) If $e = f + g$ then

$$E(X) = \mathrm{res}_Y \left(F(X - Y), G(Y)\right) \tag{50}$$

   or

$$E(X) = \mathrm{res}_Y \left(G(X - Y), F(Y)\right). \tag{51}$$

   The correctness is immediate from (46).

(b) If $e = f - g$ then

$$E(X) = \operatorname{res}_Y (F(X+Y), G(Y)) \tag{52}$$

or

$$E(X) = H(-X) \quad \text{where} \quad H(X) = \operatorname{res}_Y (G(X+Y), F(Y)). \tag{53}$$

The correctness for (52) is immediate from (46). Also, by (46), $g - f$ is a root of $H(X)$. Hence, $f - g = -(g - f)$ is a root of $H(-X)$.

(c) If $e = f \cdot g$ then [4]

$$E(X) = \operatorname{res}_Y \left(Y^{\deg F} F\left(\frac{X}{Y}\right), G(Y) \right). \tag{54}$$

The correctness is immediate from (47).

(d) If $g \neq 0$ and $e = f/g$ then

$$E(X) = \operatorname{res}_Y \left(Y^{\deg F} F\left(\frac{X}{Y}\right), Y^{\deg G} G\left(\frac{1}{Y}\right) \right). \tag{55}$$

The correctness is proven as follows:
Since $g \neq 0$, by (45), $\frac{1}{g}$ is a root of $X^{\deg G} G\left(\frac{1}{X}\right)$. By (47), $\frac{f}{g} = f\frac{1}{g}$ is a root of (55).

4. Suppose e is of the form $e = sqrt\,(f)$ for some `Expr` f with $f \geq 0$. By assumption, $F(X) = \sum_{i=0}^{m} f_i X^i \in \mathbb{Z}[X]$ with $f_m \neq 0$ such that $F(f) = 0$ is known. Then

$$E(X) = F\left(X^2\right) = \sum_{i=0}^{m} f_i X^{2i}. \tag{56}$$

This is correct, since

$$E(e) = \sum_{i=0}^{m} f_i \left(\sqrt{f}\right)^{2i} = \sum_{i=0}^{m} f_i f^i = 0.$$

6.4 Degree-Length Bound

In the previous subsection, we described the method, for a given `Expr` e, of finding $E(X) \in \mathbb{Z}[X]$ such that $E(e) = 0$. We would like to compute the length bound l_e of $E(X)$. The naive approach is just to calculate all the coefficients of $E(X)$ and use them to get $\lfloor \lg ||E||_2 \rfloor$. This is inefficient both in terms of space and time: To find $E(X)$, for each descendent f of e, we must find a polynomial $F(X) \in \mathbb{Z}[X]$ such that $F(f) = 0$, but the degree and the magnitudes of the coefficients of those polynomials easily become huge. To avoid these problems, we compute an upper bound for $\lfloor \lg ||E||_2 \rfloor$ which may not be tight, but could be gotten much more efficiently. In fact, we compute it without knowing any coefficient of $E(X)$.

[4]We could also have

$$E(X) = \operatorname{res}_Y \left(Y^{\deg G} G\left(\frac{X}{Y}\right), F(Y) \right).$$

But unlike the addition or subtraction, having this alternative choice will not affect our algorithm. Hence, we can safely ignore it.

6.4.1 Generalized Hadamard Bound

We are going to use the generalized version of Hadamard Bound [GG74] which gives us an upper bound for the 2-norm of the determinant of a matrix over $\mathbb{C}[X]$:

Theorem 22 (Generalized Hadamard Bound) *Let $P(X) = [P_{j,k}(X)]$ be an n dimensional square matrix over $\mathbb{C}[X]$. Then*

$$||\det P(X)||_2 \leq \prod_{j=1}^{n} \sqrt{\sum_{k=1}^{n} ||P_{j,k}(X)||_1^2}. \tag{57}$$

Proposition 23 *Let $F(X) = \sum_{j=0}^{m} f_j X^j$ and $G(X) = \sum_{j=0}^{n} g_j X^j \in \mathbb{Z}[X]$ with $f_m g_n \neq 0$.*

1. *If $E(X) = \mathrm{res}_Y\left(F(X \mp Y), G(Y)\right)$ then*

$$\deg E \leq m\,n \quad \textit{and} \quad ||E||_2 \leq \left(||F||_2\, 2^{m+1}\right)^n ||G||_2^m. \tag{58}$$

2. *If $E(X) = \mathrm{res}_Y\left(Y^m F\left(\frac{X}{Y}\right), G(Y)\right)$ then*

$$\deg E \leq m\,n \quad \textit{and} \quad ||E||_2 \leq ||F||_2^n\, ||G||_2^m. \tag{59}$$

Proof.

1.

$$\begin{aligned} F(X \mp Y) &= \sum_{j=0}^{m} f_j (X \mp Y)^j \\ &= \sum_{j=0}^{m} f_j \sum_{k=0}^{j} \binom{j}{k} X^{j-k} (\mp Y)^k \\ &= \sum_{k=0}^{m} \left((\mp 1)^k \sum_{j=k}^{m} f_j \binom{j}{k} X^{j-k} \right) Y^k. \end{aligned}$$

For each of the upper n rows of $\mathrm{syl}_Y\left(F(X \mp Y), G(Y)\right)$, any non-zero element in the row is a polynomial (in X) of degree at most m, and for each of the lower m rows, all the elements in the row are constants. Hence, $\deg E \leq m\,n$.

For each of the upper n rows of $\mathrm{syl}_Y\left(F(X \mp Y), G(Y)\right)$, the square root of the sum of the squared 1-norm of the elements in the row is bounded from above as

$$\begin{aligned} \sqrt{\sum_{k=0}^{m} \left|\left| (\mp 1)^k \sum_{j=k}^{m} f_j \binom{j}{k} X^{j-k} \right|\right|_1^2} &\leq \sqrt{\sum_{k=0}^{m} \left(\sum_{j=k}^{m} ||F||_\infty \binom{j}{k} \right)^2} \\ &\leq \sum_{k=0}^{m} \sum_{j=k}^{m} ||F||_\infty \binom{j}{k} \\ &= ||F||_\infty \sum_{j=0}^{m} \sum_{k=0}^{j} \binom{j}{k} \end{aligned}$$

$$\begin{aligned} &= ||F||_\infty \sum_{j=0}^{m} 2^j \\ &\leq ||F||_\infty \, 2^{m+1} \\ &\leq ||F||_2 \, 2^{m+1}. \end{aligned}$$

For each of the lower m rows of $\mathrm{syl}_Y(F(X \mp Y), G(Y))$, the square root of the sum of the squared 1-norm of the elements in the row is

$$\sqrt{\sum_{j=0}^{n} |g_j|^2} = ||G||_2 .$$

Applying (57) to $E(X)$,

$$||E||_2 \leq \left(||F||_2 \, 2^{m+1}\right)^n ||G||_2^m .$$

2. Let f_t be the tail coefficient of $F(X)$. Then

$$Y^m F\left(\frac{X}{Y}\right) = \sum_{j=0}^{m-t} f_{m-j} X^{m-j} Y^j .$$

For each of the upper n rows of $\mathrm{syl}_Y\left(Y^m F\left(\frac{X}{Y}\right), G(Y)\right)$, any non-zero element in the row is a polynomial (in X) of degree at most m, and for each of the lower m rows, all the elements in the row are constants. Hence, $\deg E \leq m\,n$.

For each of the upper n rows of $\mathrm{syl}_Y\left(Y^m F\left(\frac{X}{Y}\right), G(Y)\right)$, the square root of the sum of the squared 1-norm of the elements in the row is

$$\begin{aligned} \sqrt{\sum_{j=0}^{m-t} ||f_{m-j} X^{m-j}||_1^2} &= \sqrt{\sum_{j=0}^{m-t} |f_{m-j}|^2} \\ &= \sqrt{\sum_{k=t}^{m} |f_k|^2} \\ &= ||F||_2 . \end{aligned}$$

For each of the lower m rows of $\mathrm{syl}_Y\left(Y^m F\left(\frac{X}{Y}\right), G(Y)\right)$, the square root of the sum of the squared 1-norm of the elements in the row is

$$\sqrt{\sum_{j=0}^{n} |g_j|^2} = ||G||_2 .$$

Applying (57) to $E(X)$,

$$||E||_2 \leq ||F||_2^n \, ||G||_2^m .$$

Q.E.D.

6.4.2 Algorithms

Let e be `Expr` and $E(X) \in \mathbb{Z}[X]$ such that $E(e) = 0$. Define *a degree-length bound* of e to be a pair $(d_e, l_e) \in \mathbb{N}_{>0} \times \mathbb{N}$ which satisfies

$$d_e \geq \deg E \quad \text{and} \quad l_e \geq \lfloor \lg ||E||_2 \rfloor .$$

By Proposition 18, $e = 0$ iff 0 belongs to $\widehat{e}$ where $\widehat{e} \cong e[\infty, l_e + 2]$.

Fix any `Expr` e. Let $E(X) \in \mathbb{Z}[X]$ such that $E(e) = 0$. We would like to find a degree-length bound (d_e, l_e) of e. The bounds are computed recursively by traversing the `Expr` tree e bottom-up from the leaves to the root e.

1. Suppose e is a leaf. Then, by (48), $E(X) = pX - q$ where $(p, q) \in \mathbb{Z} \times \mathbb{Z}_{\neq 0}$ such that $e = \frac{p}{q}$ and $\gcd(p, q) = 1$. Thus

$$\deg E = 1 \quad \text{and} \quad ||E||_2 = \sqrt{p^2 + q^2}.$$

Hence, we set

$$\begin{array}{rcll} d_e & \leftarrow & 1 & \\ l_e & \leftarrow & \max\{\lfloor \lg |p| \rfloor, \lfloor \lg |q| \rfloor\} + 1 & \geq \left\lfloor \frac{1}{2} + \lg(\max\{|p|, |q|\}) \right\rfloor \\ & & & = \left\lfloor \frac{1}{2}\left(1 + \lg(\max\{|p|, |q|\})^2\right) \right\rfloor \\ & & & = \left\lfloor \frac{1}{2} \lg(2\max\{p^2, q^2\}) \right\rfloor \\ & & & \geq \left\lfloor \lg \sqrt{p^2 + q^2} \right\rfloor . \end{array}$$

2. Suppose e is of the form $e = -f$ for some `Expr` f. Assume (d_f, l_f) is known. Let $F(X) \in \mathbb{Z}[X]$ found in Section 6.3.2 such that $F(f) = 0$. Then, $d_f \geq \deg F$ and $l_f \geq \lfloor \lg ||F||_2 \rfloor$. By (49), $E(X) = F(-X)$. Thus

$$\deg E = \deg F \quad \text{and} \quad ||E||_2 = ||F||_2 .$$

Hence, we set

$$\begin{array}{rcl} d_e & \leftarrow & d_f \\ l_e & \leftarrow & l_f . \end{array}$$

3. Suppose e is of the form $e = f@g$ for some `Expr` f and g and for some $@ \in \{+, -, \cdot, /\}$. Assume (d_f, l_f) and (d_g, l_g) are known. Let $F(X)$ and $G(X) \in \mathbb{Z}[X]$ found in Section 6.3.2 such that $F(f) = G(g) = 0$. Then, $d_f \geq \deg F$, $l_f \geq \lfloor \lg ||F||_2 \rfloor$, $d_g \geq \deg G$ and $l_g \geq \lfloor \lg ||G||_2 \rfloor$.

   (a) Suppose $e = f + g$. If, as (50), $E(X) = \mathrm{res}_Y(F(X - Y), G(Y))$ then, by (58),

$$\deg E \leq \deg F \deg G \quad \text{and} \quad ||E||_2 \leq \left(||F||_2 \, 2^{\deg F + 1}\right)^{\deg G} ||G||_2^{\deg F} .$$

Thus

$$\begin{aligned}
\lfloor \lg ||E||_2 \rfloor &= \lfloor \deg G\,(\lg ||F||_2 + \deg F + 1) + \deg F\ \lg ||G||_2 \rfloor \\
&\leq \lfloor \deg G\,(\lg ||F||_2 + \deg F + 1) \rfloor + \lfloor \deg F\ \lg ||G||_2 \rfloor + 1 \\
&\leq \deg G\,(\lfloor \lg ||F||_2 \rfloor + \deg F + 1) + \deg G - 1 \\
&\quad + \deg F\,\lfloor \lg ||G||_2 \rfloor + \deg F - 1 + 1.
\end{aligned}$$

If, as (51), $E(X) = \mathrm{res}_Y\,(G(X-Y), F(Y))$ then, by (58),

$$\deg E \leq \deg G\ \deg F \quad \text{and} \quad ||E||_2 \leq \left(||G||_2\, 2^{\deg G + 1}\right)^{\deg F} ||F||_2^{\deg G}.$$

Thus

$$\begin{aligned}
\lfloor \lg ||E||_2 \rfloor &\leq \deg F\,(\lfloor \lg ||G||_2 \rfloor + \deg G + 1) + \deg F - 1 \\
&\quad + \deg G\,\lfloor \lg ||F||_2 \rfloor + \deg G - 1 + 1.
\end{aligned}$$

Hence, we set

$$\begin{aligned}
d_e &\leftarrow d_f\, d_g \\
l_e &\leftarrow d_f\, l_g + d_g\, l_f + d_f\, d_g + \min\{d_f, d_g\} + d_f + d_g - 1.
\end{aligned}$$

(b) Suppose $e = f - g$. If, as (52), $E(X) = \mathrm{res}_Y\,(F(X+Y), G(Y))$ then, by (58),

$$\deg E \leq \deg F\ \deg G \quad \text{and} \quad ||E||_2 \leq \left(||F||_2\, 2^{\deg F + 1}\right)^{\deg G} ||G||_2^{\deg F}.$$

If, as (53), $E(X) = H(-X)$ where $H(X) = \mathrm{res}_Y\,(G(X+Y), F(Y))$ then, by (58),

$$\deg E = \deg H \leq \deg G\ \deg F$$

and

$$||E||_2 = ||H||_2 \leq \left(||G||_2\, 2^{\deg G + 1}\right)^{\deg F} ||F||_2^{\deg G}.$$

Hence, we set

$$\begin{aligned}
d_e &\leftarrow d_f\, d_g \\
l_e &\leftarrow d_f\, l_g + d_g\, l_f + d_f\, d_g + \min\{d_f, d_g\} + d_f + d_g - 1.
\end{aligned}$$

(c) If $e = f \cdot g$ then, by (54), $E(X) = \mathrm{res}_Y\left(Y^{\deg F} F\left(\frac{X}{Y}\right), G(Y)\right)$. By (59),

$$\deg E \leq \deg F\ \deg G \quad \text{and} \quad ||E||_2 \leq ||F||_2^{\deg G}\, ||G||_2^{\deg F}.$$

Thus

$$\begin{aligned}
\lfloor \lg ||E||_2 \rfloor &= \lfloor \deg G\ \lg ||F||_2 + \deg F\ \lg ||G||_2 \rfloor \\
&\leq \lfloor \deg G\ \lg ||F||_2 \rfloor + \lfloor \deg F\ \lg ||G||_2 \rfloor + 1 \\
&\leq \deg G\,\lfloor \lg ||F||_2 \rfloor + \deg G - 1 \\
&\quad + \deg F\,\lfloor \lg ||G||_2 \rfloor + \deg F - 1 + 1.
\end{aligned}$$

Hence, we set

$$\begin{aligned}
d_e &\leftarrow d_f\, d_g \\
l_e &\leftarrow d_f\, l_g + d_g\, l_f + d_f + d_g - 1.
\end{aligned}$$

(d) If $g \neq 0$ and $e = f/g$ then, by (55), $E(X) = \operatorname{res}_Y \left(Y^{\deg F} F\left(\frac{X}{Y}\right), H(Y) \right)$ where $H(Y) = Y^{\deg G} G\left(\frac{1}{Y}\right)$. Since $\deg H \leq \deg G$ and $||H||_2 = ||G||_2$, by (59),

$$\deg E \leq \deg F \, \deg H \leq \deg F \, \deg G$$

and

$$||E||_2 \leq ||F||_2^{\deg H} \, ||H||_2^{\deg F} \leq ||F||_2^{\deg G} \, ||G||_2^{\deg F} .$$

Hence, we set

$$\begin{array}{rcl} d_e & \leftarrow & d_f \, d_g \\ l_e & \leftarrow & d_f \, l_g + d_g \, l_f + d_f + d_g - 1. \end{array}$$

4. Suppose e is of the form $e = \mathit{sqrt}\,(f)$ for some `Expr` f with $f \geq 0$. Assume (d_f, l_f) is known. Let $F(X) \in \mathbb{Z}[X]$ found in Section 6.3.2 such that $F(f) = 0$. Then, $d_f \geq \deg F$ and $l_f \geq \lfloor \lg ||F||_2 \rfloor$. By (56), $E(X) = F\left(X^2\right)$. Thus

$$\deg E = \deg^2 F \quad \text{and} \quad ||E||_2 = ||F||_2 .$$

Hence, we set

$$\begin{array}{rcl} d_e & \leftarrow & d_f^2 \\ l_e & \leftarrow & l_f. \end{array}$$

7 Precision-Driven Algorithm

In this section, we describe our algorithm to compute an approximation of a given `Expr` to a given precision.

Let e be `Expr`. Whenever precision p_e is given, for each child f of e, we compute the precision p_f of f so that if f is approximated by $\widehat{f}$ to p_f then $\widehat{e}$ which is computed by applying `Real` operation to $\widehat{f}$ will be an approximation of e to the required precision p_e. Thus, in our algorithms, the precisions are propagated top-down from e to the leaves, whereas approximate values are collected bottom-up from the leaves to e.

7.1 Approximation

Let e be `Expr` and $(r, a) \in \mathbb{N} \times \mathbb{Z}$. We say `Real` $\widehat{e}$ approximates e to precision $[r, a]$ and write

$$\widehat{e} \cong e[r, a]$$

if e belongs to $\widehat{e}$ and

$$\mathit{Err}_{\widehat{e}} \quad \leq \quad \max\left\{ |e| \, 2^{-r}, \, 2^{-a} \right\}.$$

If $\widehat{e} \cong e[r, a]$ then $V_{\widehat{e}} \cong e[r, a]$, i.e.,

$$|e - V_{\widehat{e}}| \quad \leq \quad \max\left\{ |e| \, 2^{-r}, \, 2^{-a} \right\}.$$

7.1.1 Properties

Lemma 24 *Fix* `Expr` e *and* $(r,a) \in \mathbb{N} \times \mathbb{Z}$. *Let* $\widehat{e} \cong e[r,a]$.

1. *If* $a \geq -\lfloor \lg |e| \rfloor$ *then*
$$|V_{\widehat{e}}| \leq 2\,|e|\,. \tag{60}$$
2. *If* $r \geq 1$ *and* $a \geq -\lfloor \lg |e| \rfloor + 1$ *then*
$$|V_{\widehat{e}}| \geq \frac{|e|}{2}. \tag{61}$$
3. *If* $r \geq 1$ *and* $a \geq -\lfloor \lg |e| \rfloor + 1$ *then*
$$|V_{\widehat{e}}| \geq Err_{\widehat{e}}. \tag{62}$$

Proof.

1. If $|V_{\widehat{e}}| \leq |e|$ then there is nothing to prove.

   Suppose $|V_{\widehat{e}}| > |e|$. Since $r \geq 0$ and $a \geq -\lfloor \lg |e| \rfloor$, $\max\{|e|\,2^{-r},\, 2^{-a}\} \leq |e|$. Then
$$|V_{\widehat{e}}| \;\leq\; |e| + Err_{\widehat{e}} \;\leq\; |e| + \max\{|e|\,2^{-r},\, 2^{-a}\} \;\leq\; 2\,|e|\,.$$
2. If $|V_{\widehat{e}}| \geq |e|$ then there is nothing to prove.

   Suppose $|V_{\widehat{e}}| < |e|$. Since $r \geq 1$ and $a \geq -\lfloor \lg |e| \rfloor + 1$, $\max\{|e|\,2^{-r},\, 2^{-a}\} \leq \frac{|e|}{2}$. Then
$$|V_{\widehat{e}}| \;\geq\; |e| - Err_{\widehat{e}} \;\geq\; |e| - \max\{|e|\,2^{-r},\, 2^{-a}\} \;\geq\; \frac{|e|}{2}.$$
3. Again, $\max\{|e|\,2^{-r},\, 2^{-a}\} \leq \frac{|e|}{2}$. Together with (61),
$$Err_{\widehat{e}} \;\leq\; \max\{|e|\,2^{-r},\, 2^{-a}\} \;\leq\; \frac{|e|}{2} \;\leq\; |V_{\widehat{e}}|\,.$$

Q.E.D.

7.2 Most Significant Bit

Let e be `Expr`. *The most significant bit* (MSB) μ_e of e is defined to be
$$\begin{cases} \lfloor \lg |e| \rfloor & \text{if } e \neq 0 \\ -\infty & \text{if } e = 0. \end{cases}$$
Note
$$2^{\mu_e} \leq |e| < 2^{\mu_e + 1}.$$
Here, we mean $2^{-\infty} = 0$ by convention. We write $\sigma_e = \operatorname{sgn}(e)$.

The MSB μ_e of e plays an important role in our precision driven algorithm. If non-trivial μ_e is known then each of the relative and absolute precisions of e could be "translated" to the other. Moreover, a non-trivial μ_e actually tells us whether or not e is *exactly* 0, i.e, if e is *exactly* 0 then $\mu_e = -\infty$. Since we usually do not know the exact value of e, the MSB itself is hard to compute. Instead, we compute an upper bound μ_e^+ and a lower bound μ_e^- for the MSB μ_e of e. We also compute the sign σ_e of e which is helpful to compute μ_e^+ and μ_e^-.

We now describe how to compute μ_e^+, μ_e^- and σ_e.

We will consider two cases:

(a) When e is newly constructed or gets some new substructure.

(b) When e has been approximated at least once to precision $[r, a]$ with $r \geq 1$ and $a \geq -\mu_e + 1$.

This idea comes from the following observation:

If e has never approximated, then we use a static algorithm to compute μ_e^+ and μ_e^-. These bounds may not be tight. Thus, once we get some approximation of e, we try to refine μ_e^+ and μ_e^-.

7.2.1 Algorithms for MSB

Let e be `Expr`. We would like to compute an upper bound μ_e^+ and a lower bound μ_e^- for the MSB μ_e of e, as well as the sign σ_e of e. They are computed recursively by traversing the `Expr` tree e bottom-up from the leaves to the root e.

(a) Suppose e is newly constructed or gets some new substructure.

1. Suppose e is a leaf. Then, the exact value $e \in \mathbb{Q}$ is known. Thus, we set

$$\begin{aligned} \mu_e^+ = \mu_e^- &\leftarrow \lfloor \lg |e| \rfloor \\ \sigma_e &\leftarrow \operatorname{sgn}(e). \end{aligned}$$

2. Suppose e is of the form $e = -f$ for some `Expr` f. By assumption, μ_f^+, μ_f^- and σ_f are known. Then

$$2^{\mu_f} \leq |-f| = |f| < 2^{\mu_f + 1}.$$

   Thus, we set

$$\begin{aligned} \mu_e^+ &\leftarrow \mu_f^+ \\ \mu_e^- &\leftarrow \mu_f^- \\ \sigma_e &\leftarrow -\sigma_f. \end{aligned}$$

3. Suppose e is of the form $e = f@g$ for some `Expr` f and g and for some $@ \in \{+, -, \cdot, /\}$. By assumption, μ_f^+, μ_f^-, σ_f, μ_g^+, μ_g^- and σ_g are known.

   (a) Suppose $e = f \pm g$. There are several cases depending on σ_f and σ_g.

i. If either $e = f + g$ and $\sigma_f \sigma_g > 0$ or $e = f - g$ and $\sigma_f \sigma_g < 0$ then

$$\begin{aligned} 2^{\max\left\{\mu_f, \mu_g\right\}} &< 2^{\mu_f} + 2^{\mu_g} \\ &\leq |f| + |g| = |f \pm g| \\ &< 2^{\mu_f+1} + 2^{\mu_g+1} \leq 2^{\max\left\{\mu_f, \mu_g\right\}+2}. \end{aligned}$$

Thus, we set

$$\begin{aligned} \mu_e^+ &\leftarrow \max\left\{\mu_f^+, \mu_g^+\right\} + 1 \\ \mu_e^- &\leftarrow \max\left\{\mu_f^-, \mu_g^-\right\} \\ \sigma_e &\leftarrow \sigma_f. \end{aligned}$$

ii. If either $e = f + g$ and $\sigma_f \sigma_g < 0$ or $e = f - g$ and $\sigma_f \sigma_g > 0$ then

$$||f| - |g|| = |f \pm g| < 2^{\max\left\{\mu_f+1, \mu_g+1\right\}}.$$

Thus, we set

$$\mu_e^+ \leftarrow \max\left\{\mu_f^+, \mu_g^+\right\}.$$

To get μ_e^- and σ_e, we consider three sub-cases.

A. Suppose $\mu_f^- - \mu_g^+ \geq 2$. Then

$$\begin{aligned} |f \pm g| &= |f| - |g| \\ &> 2^{\mu_f^-} - 2^{\mu_g^+ + 1} \\ &= \left(2^{\mu_f^- - \mu_g^+ - 1} - 1\right) 2^{\mu_g^+ + 1} \\ &\geq 2^{\mu_f^- - \mu_g^+ - 2}\, 2^{\mu_g^+ + 1} \qquad (\text{since } \mu_f^- - \mu_g^+ - 1 \geq 1) \\ &= 2^{\mu_f^- - 1}. \end{aligned}$$

Thus, we set

$$\begin{aligned} \mu_e^- &\leftarrow \mu_f^- - 1 \\ \sigma_e &\leftarrow \sigma_f. \end{aligned}$$

Intuitively, the above means that μ_g is much smaller than μ_f so that, even though $f \pm g$ is performed, g cannot cancel out μ_f.

B. Suppose $\mu_g^- - \mu_f^+ \geq 2$. By a similar argument to the previous case,

$$|f \pm g| = -|f| + |g| > 2^{\mu_g^- - 1}.$$

Thus, we set

$$\begin{aligned} \mu_e^- &\leftarrow \mu_g^- - 1 \\ \sigma_e &\leftarrow \begin{cases} \sigma_g & \text{if } e = f + g \\ -\sigma_g & \text{if } e = f - g. \end{cases} \end{aligned}$$

C. Otherwise, μ_f and μ_g are almost the same, and most (possibly all) of the significant bits of f and g will cancel out with each other. Unfortunately, there is no way to predict how many of them will cancel out, and we cannot find μ_e^- or σ_e by just using statically obtained quantities. We must use the algorithm which will be described later.

iii. If $\sigma_f \neq 0$ but $\sigma_g = 0$ then $f \pm g = f$. Thus, we set

$$\begin{aligned} \mu_e^+ &\leftarrow \mu_f^+ \\ \mu_e^- &\leftarrow \mu_f^- \\ \sigma_e &\leftarrow \sigma_f. \end{aligned}$$

iv. If $\sigma_f = 0$ but $\sigma_g \neq 0$ then $|f \pm g| = |g|$. Thus, we set

$$\begin{aligned} \mu_e^+ &\leftarrow \mu_g^+ \\ \mu_e^- &\leftarrow \mu_g^- \\ \sigma_e &\leftarrow \begin{cases} \sigma_g & \text{if } e = f + g \\ -\sigma_g & \text{if } e = f - g. \end{cases} \end{aligned}$$

v. If $\sigma_f = \sigma_g = 0$ then $f \pm g = 0$.

(b) If $e = f \cdot g$ then

$$2^{\mu_f + \mu_g} \leq |f|\,|g| < 2^{\mu_f + \mu_g + 2}.$$

Thus, we set

$$\begin{aligned} \mu_e^+ &\leftarrow \mu_f^+ + \mu_g^+ + 1 \\ \mu_e^- &\leftarrow \mu_f^- + \mu_g^- \\ \sigma_e &\leftarrow \sigma_f\,\sigma_g. \end{aligned}$$

(c) Suppose $e = f/g$. If $\sigma_g = 0$ then e is not well-defined. Otherwise

$$2^{\mu_f - \mu_g - 1} < \left|\frac{f}{g}\right| < 2^{\mu_f - \mu_g + 1}.$$

Thus, we set

$$\begin{aligned} \mu_e^+ &\leftarrow \mu_f^+ - \mu_g^- \\ \mu_e^- &\leftarrow \mu_f^- - \mu_g^+ - 1 \\ \sigma_e &\leftarrow \sigma_f\,\sigma_g. \end{aligned}$$

4. Suppose e is of the form $e = \mathit{sqrt}(f)$ for some `Expr` f. By assumption, μ_f^+, μ_f^- and σ_f are known. If $\sigma_f = -1$ then e is not well-defined. Otherwise

$$2^{\lfloor \frac{\mu_f}{2} \rfloor} \leq 2^{\frac{\mu_f}{2}} \leq \sqrt{f} < 2^{\frac{\mu_f + 1}{2}} \leq 2^{\lfloor \frac{\mu_f}{2} \rfloor + 1}.$$

Thus, we set

$$\begin{aligned} \mu_e^+ &\leftarrow \left\lfloor \frac{\mu_f^+}{2} \right\rfloor \\ \mu_e^- &\leftarrow \left\lfloor \frac{\mu_f^-}{2} \right\rfloor \\ \sigma_e &\leftarrow \sigma_f. \end{aligned}$$

(b) Suppose e has been approximated by $\widehat{e}$ to precision $[r, a]$ with $r \geq 1$ and $a \geq -\mu_e + 1$.

The following proposition suggests that we could refine μ_e^+ and μ_e^- when a suitable approximation of e is known.

Proposition 25

$$\mu_{\widehat{e}} + 1 \quad \geq \quad \mu_e \quad \geq \quad \mu_{\widehat{e}} - 1.$$

Proof. By (61),

$$|e| \quad \leq \quad 2\,|V_{\widehat{e}}| \quad < \quad 2^{\mu_{\widehat{e}} + 2}.$$

By (60),

$$|e| \quad \geq \quad \frac{|V_{\widehat{e}}|}{2} \quad \geq \quad 2^{\mu_{\widehat{e}} - 1}.$$

Q.E.D.

We now could have the algorithm to refine μ_e^+ and μ_e^-:

$$\begin{aligned} \mu_e^+ \quad &\leftarrow \quad \min\left\{\mu_e^+,\ \mu_{\widehat{e}} + 1\right\} \\ \mu_e^- \quad &\leftarrow \quad \max\left\{\mu_e^-,\ \mu_{\widehat{e}} - 1\right\}. \end{aligned}$$

7.3 Precision-Driven Algorithm

Let `Expr` e and $(r_e, a_e) \in \mathbb{N}\times\mathbb{Z}$. We would like to compute `Real` $\widehat{e}$ such that $\widehat{e} \cong e\,[r_e, a_e]$. There are several cases depending on the type of e.

1. Suppose e is a leaf. Then, the exact value $e \in \mathbb{Q}$ is known. We simply call the approximation algorithm to compute $\widehat{e} \in$ `Real` with the error-bound $[r_e, a_e]$ such that e belongs to $\widehat{e}$. By (27) in Proposition 11, $\widehat{e} \cong e[r, a]$.

2. Suppose e is of the form $e = -f$ for some `Expr` f. The computation of $\widehat{e}$ consists of two phases:

   (a) Set $r_f \leftarrow r_e$ and $a_f \leftarrow a_e$, and make a recursive call to compute $\widehat{f} \cong f\,[r_e, a_e]$.

   (b) Set $\widehat{e} \leftarrow -\widehat{f}$.

   By Proposition 12,

   $$\left|e - \left(-V_{\widehat{f}}\right)\right| \quad = \quad |e - V_{\widehat{e}}| \quad \leq \quad \max\left\{|e|\,2^{-r_e},\ 2^{-a_e}\right\}.$$

3. Suppose e is a binary operator node of the form $e = f@g$ for some `Expr` f and g and for some $@ \in \{+, -, \cdot, /\}$. The computation of $\widehat{e}$ consists of two phases:

   (a) Determine (r_f, a_f) and $(r_g, a_g) \in \mathbb{N} \times \mathbb{Z}$ so that

   $$\left|e - V_{f[r_f, a_f]@g[r_g, a_g]}\right| \quad \leq \quad \max\left\{|e|\,2^{-r_e},\ 2^{-a_e}\right\},$$

   and make recursive calls to compute $\widehat{f} \cong f\,[r_f, a_f]$ and $\widehat{f} \cong g\,[r_g, a_g]$.

(b) Compute $\widehat{f}@\widehat{g}$ to get $\widehat{e}$.

4. Suppose e is of the form $e = sqrt\,(f)$ for some `Expr` f with $f \geq 0$. The computation of $\widehat{e}$ consists of two phases:

   (a) Determine $(r_f, a_f) \in \mathbb{N} \times \mathbb{Z}$ so that

   $$\left| e - sqrt\left(V_{f[r_f, a_f]}\right)\right| \;\leq\; \max\left\{|e|\,2^{-r_e},\, 2^{-a_e}\right\},$$

   and make a recursive call to compute $\widehat{f} \cong f\,[r_f, a_f]$.

   (b) Compute $sqrt\left(\widehat{f}\right)$ to get $\widehat{e}$.

Since phase (b) of the algorithms for binary operator nodes and $sqrt$ nodes is just the `Real` operation (and its correctness immediately follows from Proposition 12), we will concentrate on phase (a).

7.3.1 Addition and Subtraction

Consider an `Expr` of the form $e = f \pm g$. Given $(r_e, a_e) \in \mathbb{N} \times \mathbb{Z}$, we would like to determine (r_f, a_f) and $(r_g, a_g) \in \mathbb{N} \times \mathbb{Z}$ so that

$$\left| e - V_{f[r_f, a_f] \pm g[r_g, a_g]}\right| \;\leq\; \max\left\{|e|\,2^{-r_e},\, 2^{-a_e}\right\}. \tag{63}$$

Proposition 26 *To ensure (63), it suffices to set*

$$\begin{array}{ll} r_f \leftarrow \max\left\{\mu_f^+ - \mu_e^- + r_e + 4,\, 0\right\}, & a_f \leftarrow a_e + 3, \\ r_g \leftarrow \max\left\{\mu_g^+ - \mu_e^- + r_e + 4,\, 0\right\}, & a_g \leftarrow a_e + 3. \end{array} \tag{64}$$

Proof. By (33), it is enough to show

$$6 \max\left\{Err_{\widehat{f}},\, Err_{\widehat{g}}\right\} \;\leq\; \max\left\{|e|\,2^{-r_e},\, 2^{-a_e}\right\}.$$

By symmetry, we may assume $\max\left\{Err_{\widehat{f}},\, Err_{\widehat{g}}\right\} = Err_{\widehat{f}}$. If $Err_{\widehat{f}} \leq |f|\,2^{-r_f}$ then

$$\begin{aligned} 6\,Err_{\widehat{f}} &\leq 6\,|f|\,2^{-r_f} \\ &\leq 6\,|f|\,2^{-\left(\mu_f^+ + 1\right) + \mu_e^- - r_e - 3} \\ &< |e|\,2^{-r_e}. \end{aligned}$$

If $Err_{\widehat{f}} \leq 2^{-a_f}$ then

$$\begin{aligned} 6\,Err_{\widehat{f}} &\leq 6 \cdot 2^{-a_f} \\ &\leq 6 \cdot 2^{-a_e - 3} \\ &\leq 2^{-a_e}. \end{aligned}$$

Q.E.D.

7.3.2 Lower Bound for MSB

We now describe the algorithm to set μ_e^- and σ_e for `Expr` e when e is of the form either $f+g$ with $\sigma_f\,\sigma_g < 0$ or $f-g$ with $\sigma_f\,\sigma_g > 0$, and neither $\mu_f^- - \mu_g^+ \geq 2$ nor $\mu_g^- - \mu_f^+ \geq 2$. In this case, μ_f and μ_g are almost the same, and μ_e becomes very tiny (possibly $-\infty$). To get μ_e^- and σ_e, we must eventually compute an approximation of e to some precision.

Setting

$$\begin{aligned} r_f &\leftarrow \max\left\{\mu_f^+ + l_e + 6,\, 0\right\}, \qquad a_f \leftarrow l_e + 5, \\ r_g &\leftarrow \max\left\{\mu_g^+ + l_e + 6,\, 0\right\}, \qquad a_g \leftarrow l_e + 5, \end{aligned}$$

we compute $\widehat{f} \cong f\,[r_f, a_f]$ and $\widehat{g} \cong g\,[r_g, a_g]$ by our precision-driven algorithm. Note $\max\{|f|\,2^{-r_f},\, 2^{-a_f}\} \leq 2^{-l_e-5}$ and $\max\{|g|\,2^{-r_g},\, 2^{-a_g}\} \leq 2^{-l_e-5}$. Thus,

$$\max\left\{Err_{\widehat{f}},\, Err_{\widehat{g}}\right\} \;\leq\; 2^{-l_e-5}.$$

By (33),

$$Err_{\widehat{e}} \;\leq\; 6\max\left\{Err_{\widehat{f}},\, Err_{\widehat{g}}\right\} \;\leq\; 6\cdot 2^{-l_e-5} \;<\; 2^{-l_e-2}.$$

Hence, $\widehat{e} \cong e\,[\infty, l_e+2]$.

By Proposition 18, if 0 belongs to $\widehat{e}$ then $e = 0$.

Unless 0 belongs to $\widehat{e}$ then $e \neq 0$. Thus, we could set

$$\begin{aligned} \mu_e^- &\leftarrow \left\lfloor \lg\left(\left|V_{\widehat{e}}\right| - Err_{\widehat{e}}\right)\right\rfloor \\ \sigma_e &\leftarrow \operatorname{sgn}\left(V_{\widehat{e}}\right). \end{aligned}$$

7.3.3 Multiplication

Consider an `Expr` of the form $e = f\cdot g$. Given $(r_e, a_e) \in \mathbb{N}\times\mathbb{Z}$, we would like to determine (r_f, a_f) and $(r_g, a_g) \in \mathbb{N}\times\mathbb{Z}$ so that

$$\left|e - V_{f[r_f,a_f]\cdot g[r_g,a_g]}\right| \;\leq\; \max\left\{|e|\,2^{-r_e},\, 2^{-a_e}\right\}. \tag{65}$$

Proposition 27 *To ensure (65), it suffices to set*

$$\begin{aligned} r_f &\leftarrow r_e + 4, \qquad a_f \leftarrow \max\left\{-\mu_f^- + 1,\, \mu_g^+ + a_e + 5\right\}, \\ r_g &\leftarrow r_e + 4, \qquad a_g \leftarrow \max\left\{-\mu_g^- + 1,\, \mu_f^+ + a_e + 5\right\}. \end{aligned} \tag{66}$$

Proof. By (34), it is enough to show

$$6\max\left\{\left|V_{\widehat{f}}\right| Err_{\widehat{g}},\, Err_{\widehat{f}}\left|V_{\widehat{g}}\right|,\, Err_{\widehat{f}}Err_{\widehat{g}}\right\} \;\leq\; \max\left\{|e|\,2^{-r_e},\, 2^{-a_e}\right\}. \tag{67}$$

Observe $r_f \geq 4$ and $a_f \geq -\mu_f^- + 1$. Then, by (62), $Err_{\widehat{f}} \leq \left|V_{\widehat{f}}\right|$. Thus

$$Err_{\widehat{f}}Err_{\widehat{g}} \;\leq\; \left|V_{\widehat{f}}\right| Err_{\widehat{g}}.$$

Hence, to have (67), we only need to show

$$6 \max\left\{\left|V_{\widehat{f}}\right| Err_{\widehat{g}},\, Err_{\widehat{f}}\left|V_{\widehat{g}}\right|\right\} \;\le\; \max\left\{|e|\, 2^{-r_e},\, 2^{-a_e}\right\}.$$

We claim

$$6\left|V_{\widehat{f}}\right| Err_{\widehat{g}} \;\le\; \max\left\{|e|\, 2^{-r_e},\, 2^{-a_e}\right\}.$$

Since $a_f \ge -\mu_f^- + 1$, by (60), $\left|V_{\widehat{f}}\right| \le 2\,|f|$. If $Err_{\widehat{g}} \le |g|\, 2^{-r_g}$ then

$$\begin{aligned} 6\left|V_{\widehat{f}}\right| Err_{\widehat{g}} &\le 6 \cdot 2\,|f|\,|g|\, 2^{-r_g} \\ &\le 12\,|f \cdot g|\, 2^{-r_e - 4} \\ &\le |e|\, 2^{-r_e}. \end{aligned}$$

If $Err_{\widehat{g}} \le 2^{-a_g}$ then

$$\begin{aligned} 6\left|V_{\widehat{f}}\right| Err_{\widehat{g}} &\le 6 \cdot 2\,|f|\, 2^{-a_g} \\ &\le 12\,|f|\, 2^{-\left(\mu_f^+ + 1\right) - a_e - 4} \\ &< 2^{-a_e}. \end{aligned}$$

Similarly

$$6\, Err_{\widehat{f}}\left|V_{\widehat{g}}\right| \;\le\; \max\left\{|e|\, 2^{-r_e},\, 2^{-a_e}\right\}.$$

Q.E.D.

7.3.4 Division

Consider an `Expr` of the form $e = f/g$. Given $(r_e, a_e) \in \mathbb{N} \times \mathbb{Z}$, we would like to determine (r_f, a_f) and $(r_g, a_g) \in \mathbb{N} \times \mathbb{Z}$ so that

$$\left|e - V_{f[r_f, a_f]/g[r_g, a_g]}\right| \;\le\; \max\left\{|e|\, 2^{-r_e},\, 2^{-a_e}\right\}. \tag{68}$$

Proposition 28 *To ensure (68), it suffices to set*

$$\begin{array}{ll} r_f \leftarrow \min\left\{r_e + 7,\, \max\left\{\mu_e^+ + a_e + 8,\, 2\right\}\right\}, & a_f \leftarrow -\mu_f^- + r_f, \\ r_g \leftarrow \min\left\{r_e + 7,\, \max\left\{\mu_e^+ + a_e + 8,\, 2\right\}\right\}, & a_g \leftarrow -\mu_g^- + r_g, \\ r_{\texttt{default}} \leftarrow \max\left\{r_{\texttt{default}},\, \min\left\{r_e + 6,\, \mu_e^+ + a_e + 7\right\}\right\}. & \end{array} \tag{69}$$

Proof. First, note $\max\left\{|f|\, 2^{-r_f},\, 2^{-a_f}\right\} = |f|\, 2^{-r_f}$ and $\max\left\{|g|\, 2^{-r_g},\, 2^{-a_g}\right\} = |g|\, 2^{-r_g}$. Thus, we only need to consider the case where f and g are both bounded by their relative precisions.

Next, observe $r_f \ge 2$, $a_f \ge -\mu_f^- + 2$, $r_g \ge 2$ and $a_g \ge -\mu_g^- + 2$. Then, by (60) and (61),

$$\frac{|f|}{2} \le \left|V_{\widehat{f}}\right| \le 2\,|f| \quad\text{and}\quad \frac{|g|}{2} \le \left|V_{\widehat{g}}\right| \le 2\,|g|. \tag{70}$$

Moreover,

$$Err_{\widehat{f}} \;\leq\; \frac{\left|V_{\widehat{f}}\right|}{2}, \tag{71}$$

because $Err_{\widehat{f}} \leq |f|\,2^{-r_f} \leq \left|V_{\widehat{f}}\right| 2^{-r_f+1} \leq \frac{\left|V_{\widehat{f}}\right|}{2}$. Similarly

$$Err_{\widehat{g}} \;\leq\; \frac{\left|V_{\widehat{g}}\right|}{2}. \tag{72}$$

Now, we show that (68) holds. There are two cases.

1. Suppose $Err_{\widehat{f}} = Err_{\widehat{g}} = 0$. By (35), it is enough to show

   $$12\,\frac{\left|V_{\widehat{f}}\right|}{\left|V_{\widehat{g}}\right|}\,r_{\texttt{default}} \;\leq\; \max\left\{|e|\,2^{-r_e},\,2^{-a_e}\right\}.$$

   But, by (70),

   $$\begin{aligned} 12\,\frac{\left|V_{\widehat{f}}\right|}{\left|V_{\widehat{g}}\right|}\,r_{\texttt{default}} &\leq 12\,\frac{2\,|f|}{\frac{|g|}{2}}\,r_{\texttt{default}} \\ &\leq 48\,|e|\max\left\{2^{-r_e-6},\,2^{-\left(\mu_e^{+}+1\right)-a_e-6}\right\} \\ &\leq \max\left\{|e|\,2^{-r_e},\,2^{-a_e}\right\}. \end{aligned}$$

2. Suppose $Err_{\widehat{f}} > 0$ or $Err_{\widehat{g}} > 0$. Since (71) and (72) hold, by (36), it is enough to show

   $$12\,\frac{\left|V_{\widehat{f}}\right|}{\left|V_{\widehat{g}}\right|}\max\left\{\frac{Err_{\widehat{f}}}{\left|V_{\widehat{f}}\right|},\,\frac{Err_{\widehat{g}}}{\left|V_{\widehat{g}}\right|}\right\} \;\leq\; \max\left\{|e|\,2^{-r_e},\,2^{-a_e}\right\}.$$

   If $Err_{\widehat{f}} \leq |f|\,2^{-r_f}$ and $Err_{\widehat{g}} \leq |g|\,2^{-r_g}$ then, by (70),

   $$\begin{aligned} 12\,\frac{\left|V_{\widehat{f}}\right|}{\left|V_{\widehat{g}}\right|}\max\left\{\frac{Err_{\widehat{f}}}{\left|V_{\widehat{f}}\right|},\,\frac{Err_{\widehat{g}}}{\left|V_{\widehat{g}}\right|}\right\} &\leq 12\,\frac{2\,|f|}{\frac{|g|}{2}}\max\left\{\frac{Err_{\widehat{f}}}{\frac{|f|}{2}},\,\frac{Err_{\widehat{g}}}{\frac{|g|}{2}}\right\} \\ &\leq 96\,|e|\max\left\{2^{-r_e-7},\,2^{-\left(\mu_e^{+}+1\right)-a_e-7}\right\} \\ &\leq \max\left\{|e|\,2^{-r_e},\,2^{-a_e}\right\}. \end{aligned}$$

Q.E.D.

7.3.5 Squareroot

Consider an `Expr` of the form $e = sqrt\,(f)$. Given $(r_e, a_e) \in \mathbb{N} \times \mathbb{Z}$, we would like to determine $(r_f, a_f) \in \mathbb{N} \times \mathbb{Z}$ so that

$$\left| e - sqrt\left(V_{f[r_f,a_f]}\right)\right| \;\leq\; \max\left\{|e|\,2^{-r_e},\, 2^{-a_e}\right\}. \tag{73}$$

Proposition 29 *To ensure (73), it suffices to set*

$$\begin{array}{l} r_f \leftarrow 2\,r_e + 8, \qquad a_f \leftarrow 2\,a_e + 8, \\ a_{\texttt{default}} \leftarrow \max\{a_{\texttt{default}},\, \min\{-\mu_e^- + r_e,\, a_e\}\}. \end{array} \tag{74}$$

Proof. There are two cases.

1. Suppose $Err_{\widehat{f}} = 0$. By (37), it is enough to show

$$2^{-a_{\texttt{default}}} \;\leq\; \max\left\{|e|\,2^{-r_e},\, 2^{-a_e}\right\}.$$

   But

$$2^{-a_{\texttt{default}}} \;\leq\; \max\left\{2^{\mu_e^- - r_e},\, 2^{-a_e}\right\} \;\leq\; \max\{|e|\,2^{-r_e},\, 2^{-a_e}\}.$$

2. Suppose $Err_{\widehat{f}} > 0$. By (38), it is enough to show

$$16\sqrt{Err_{\widehat{f}}} \;\leq\; \max\left\{|e|\,2^{-r_e},\, 2^{-a_e}\right\}.$$

   If $Err_{\widehat{f}} \leq |f|\,2^{-r_f}$ then

$$\begin{aligned} 16\sqrt{Err_{\widehat{f}}} &\leq 16\sqrt{|f|\,2^{-r_f}} \\ &= 16\sqrt{|f|}\,2^{-r_e-4} \\ &= |e|\,2^{-r_e}. \end{aligned}$$

   If $Err_{\widehat{f}} \leq 2^{-a_f}$ then

$$\begin{aligned} 16\sqrt{Err_{\widehat{f}}} &\leq 16\sqrt{2^{-a_f}} \\ &= 16 \cdot 2^{-a_e-4} \\ &= 2^{-a_e}. \end{aligned}$$

Q.E.D.

8 Conclusion

Most geometric algorithms are designed under the assumption that all the numerical quantities are real (algebraic) numbers and they can be computed exactly. Thus, their implementations are quite difficult and often practically impossible.

As an typical example, we consider the problem of the sign determination of determinants of square matrices. Many geometrical predicates such as "left of line" or "on circle" can be reduced into this problem.

For this problem, several robust implementations are proposed. Some of them are based on floating-point arithmetic, and therefore, every implementation can work correctly with some specific inputs and under some limited conditions. A user must carefully choose the appropriate implementation which satisfies his/her request, and probably some adjustments need to be done.

Our `Real/Expr` package may relax these annoying conditions to some extent. By using the `Real/Expr` package, the user can have a simple implementation, namely, expand the determinant to get the algebraic expressions for it, and apply the inequality operator. The elements of the input matrix could be of any type from which an instance of `Real` can be constructed. In particular, the inputs could be arbitrarily long. Moreover, the same implementation can be used for matrices of any dimension although it is not practical for dimensions above 6.

We would also like to say that the algorithms we use to determine the sign of expressions may be more efficient than the other exact computation package where the naive implementation of the exact computation is taken.

We conclude that users can use our `Real/Expr` package to implement the exact algorithms in the very general situation. More specifically, the `Real/Expr` package has the following significant points:

- Users can implement the exact algorithms without being constrained by the restrictions caused by fixed-precision arithmetic.
- Users can expect better performance than the traditional exact computation tools where all numerical quantities are computed exactly.
- Users can deal with algebraic expressions involving the squareroots.

We expect the `Real/Expr` package will be used in the following situations.

Under some circumstances, the `Real/Expr` package may be a primary candidate to implement algorithms. The implementation could be an almost straightforward interpretation of the underlying algorithm.

Nevertheless, floating-point arithmetic is fast. It is quite natural to choose floating-point arithmetic to implement algorithms. Then, the robustness (or exactness) needs to be ensured. In this situation, users can embed the `Real/Expr` package in their implementation at some critical points where exactness is important.

Finally, users may use the `Real/Expr` package as a verifier of floating-point implementation.

9 Acknowledgements

First of all, I would like to thank Prof. Chee Yap for his supervision, and to Prof. Marsha Berger for being a reader of my paper.

Also, I would like to express my appreciation to Arieh Listowsky for his comments, to Catalin Floristean for having discussions, and to Fabian Monrose who kept encouraging me.

References

[Cop92] James O. Coplien. *Advanced C++ programming styles and idioms.* Addison-Wesley Publishers, Co., Reading, MA., 1992.

[DY93] T. Dubé and Chee Yap. A Basis for Implementing Exact Geometric Algorithms (extended abstract), Sep 1993. The electronic copy is available via `ftp://cs.nyu.edu/pub/local/yap/exact/basis.ps.gz`.

[GG74] A. J. Goldstein and R. L. Graham. A Hadamard-type bound on the coefficients of a determinant of polynomials. *SIAM Review*, 16:394–395, 1974.

[Gol91] David Goldberg. What Every Computer Scientist Should Know About Floating Point Arithmetic. *ACM Computing Surveys*, 23(1):5–48, Mar 1991.

[PH90] David A. Patterson and John L. Hennessy. *Computer Architecture: a quantitative approach.* Morgan Kaufmann Publishers, Inc., San Mateo, CA., 1990. (with an appendix on Computer Arithmetic by David Goldberg).

[Yap97] Chee Yap. *Fundamental Problems in Algorithmic Algebra.* Princeton University Press, Princeton, NJ., 1997. The electronic copy is available via `ftp://cs.nyu.edu/pub/local/yap/algebra-bk/`.