Reliable Computing (2005) 11: 313–319
DOI: 10.1007/s11155-005-6893-9

A Constructive Criticism of the C/C++ Proposal for Complex Arithmetic

FRANKY BACKELJAUW and ANNIE CUYT
Department of Mathematics and Computer Science, University of Antwerp, Campus Middelheim, Middelheim 1, B–2020 Antwerpen, Belgium, e-mail: {franky.backeljauw,annie.cuyt}@ua.ac.be

(Received: 28 June 2004; accepted: 9 January 2005)

Abstract. The IEEE 754 and 854 standards regulate the behaviour of real floating-point arithmetic, as implemented in most current hard- and software systems. Although a myriad of libraries for complex floating-point arithmetic is available and in use, there is no general consensus on their implementation. The International C Standard describes in its Annex G guidelines for the implementation of complex arithmetic, in order to achieve a similar behaviour of complex floating-point arithmetic across C-language compliant implementations. In Section 2 we summarize its recommendations and outline the problems inherent to this approach. In Section 3 we describe how the lack of reliability, when computing certain complex-valued expressions, can be overcome. Throughout the discussion the rounding mode is assumed to be round-to-nearest, as in Annex G.

1. Introduction

The IEEE standards for real floating-point arithmetic are a formal model for floating-point arithmetic in which as many properties as possible have successfully been transferred from real arithmetic. These standards realize a closed system for the basic operations, remainder and square root function, including a limited number of properties (such as commutativity) and identities (such as $1 / (1 / x) = x$). We want to analyze whether the Annex G proposal for complex arithmetic [1] is equally successful. The interested reader is also referred to [3]–[5] for introductory material on complex floating-point arithmetic.

We assume to have at our disposal an IEEE compliant implementation of floating-point arithmetic, in base β, of precision t and with the exponent of the normalized numbers ranging between L and U. The signed zeroes and denormal numbers carry exponent $L - 1$. Overflow resulting in a signed infinity or an invalid expression resulting in Not-a-Number (NaN) give rise to results carrying exponent $U + 1$. Based on this floating-point implementation we realize implementations for purely imaginary, purely real and true complex arithmetic, following the Annex G guidelines. Nevertheless it is not difficult to construct examples of simple but problematic complex-valued expressions. Take for instance, with $2 \otimes x \geq \beta^{U+1}$,

$$\bigl(\mathrm{rem}(x, +0) + 2\mathtt{i}\bigr) \otimes (x + 1\mathtt{i}) = ?$$

Clearly this expression should evaluate to a complex NaN, and indeed, using plain IEEE compliant floating-point arithmetic,

$$(\mathrm{NaN} \ominus 2) + (\mathrm{NaN} \oplus +\infty)\mathtt{i} = \mathrm{NaN} + \mathrm{NaN}\mathtt{i} = \text{complex NaN}.$$

But when using the recommendation for the implementation of the product of complex numbers as found in Annex G,

```
#include <math.h>
#include <complex.h>

/* Multiply z * w ... */
double complex _Cmultd(double complex z, double complex w)

{ #pragma STDC FP_CONTRACT OFF
  double a, b, c, d, ac, bd, ad, bc, x, y;
  a = creal(z); b = cimag(z);
  c = creal(w); d = cimag(w);
  ac = a * c; bd = b * d;
  ad = a * d; bc = b * c;
  x = ac - bd; y = ad + bc;
  if (isnan(x) && isnan(y)) {
    /* Recover infinities that computed as NaN+iNaN ... */
    int recalc = 0;
    if (isinf(a) || isinf(b)) { // z is infinite
      /* "Box" the infinity and change NaNs in the other factor to 0 */
      a = copysign(isinf(a) ? 1.0 : 0.0, a);
      b = copysign(isinf(b) ? 1.0 : 0.0, b);
      if (isnan(c)) c = copysign(0.0, c);
      if (isnan(d)) d = copysign(0.0, d);
      recalc = 1;
    }
    if (isinf(c) || isinf(d)) { // w is infinite
      /* "Box" the infinity and change NaNs in the other factor to 0 */
      c = copysign(isinf(c) ? 1.0 : 0.0, c);
      d = copysign(isinf(d) ? 1.0 : 0.0, d);
      if (isnan(a)) a = copysign(0.0, a);
      if (isnan(b)) b = copysign(0.0, b);
      recalc = 1;
    }
```

```
    if (!recalc && (isinf(ac) || isinf(bd) || isinf(ad) || isinf(bc)))
    {
      /* Recover infinities from overflow by changing NaNs to 0 ... */
      if (isnan(a)) a = copysign(0.0, a);
      if (isnan(b)) b = copysign(0.0, b);
      if (isnan(c)) c = copysign(0.0, c);
      if (isnan(d)) d = copysign(0.0, d);
      recalc = 1;
    }
    if (recalc) {
      x = INFINITY * ( a * c - b * d );
      y = INFINITY * ( a * d + b * c );
    }
  }
  return x + I * y; }
```

then the result needs to be recomputed because one of the four intermediate results, called `ac`, `bd`, `ad`, `bc`, evaluates to either $+\infty$ or $-\infty$. The recomputation delivers the final result

$$(\mathrm{NaN} + 2\mathtt{i}) \otimes (x + 1\mathtt{i}) \rightarrow |\infty| \times [(+0 + 2\mathtt{i}) \otimes (x + 1\mathtt{i})] = -\infty + \infty\mathtt{i}$$

which is incorrect. Let us now analyze the origin of the problem and formulate a possible cure for it.

2. Complex Arithmetic as Detailed in Annex G

As can be expected, the problematic situations arise from the appearance of signed zeroes, signed infinities and NaNs in expressions, especially when at least 4 representations of zero or underflow, namely $(+/-\ 0) + (+/-\ 0)\mathtt{i}$, and 4 representations of the complex Riemann infinity, namely $(+/-\ \infty) + (+/-\ \infty)\mathtt{i}$, exist. The fact that combinations of these special values in the real and imaginary part of a complex number are possible, gives rise to particular problems. For instance, the result of the multiplication

$$(+\infty + \infty\mathtt{i}) \otimes (-\infty - \infty\mathtt{i}) = \mathrm{NaN} - \infty\mathtt{i} \tag{2.1}$$

is interpreted in Annex G as infinity. That this is correct, can be shown as follows. Let us introduce the notation $r\ \exp^{[\theta_1,\theta_2]\mathtt{i}}$ to denote a complex number with modulus r and argument θ satisfying $\theta_1 \leq \theta \leq \theta_2$. Using this notation, it is easy to see that (2.1) signifies

$$\begin{aligned}(+\infty + \infty\mathtt{i}) \otimes (-\infty - \infty\mathtt{i}) &= (|\infty| \exp^{[0,\pi/2]\mathtt{i}}) \times (|\infty| \exp^{[-\pi,-\pi/2]\mathtt{i}}) \\ &= |\infty| \exp^{[-\pi,0]\mathtt{i}} \end{aligned} \tag{2.2}$$

which is a complex number of infinitely large modulus lying in the lower half-plane.

As a consequence of this type of situations, Annex G proposes to also consider the 4 values $(^+/-\infty) + \mathrm{NaN}\mathtt{i}$ and $\mathrm{NaN} + (^+/-\infty)\mathtt{i}$ as representations of the Riemann infinity, in addition to the 4 representations already listed above. These representations actually stand for complex numbers with infinitely large modulus lying in the 4 possible halfplanes:

$$\begin{aligned}(+\infty) + \mathrm{NaN}\mathtt{i} &= |\infty| \exp^{[-\pi/2,\pi/2]\mathtt{i}},\\ (-\infty) + \mathrm{NaN}\mathtt{i} &= |\infty| \exp^{[\pi/2,3\pi/2]\mathtt{i}},\\ \mathrm{NaN} + (+\infty)\mathtt{i} &= |\infty| \exp^{[0,\pi]\mathtt{i}},\\ \mathrm{NaN} + (-\infty)\mathtt{i} &= |\infty| \exp^{[-\pi,0]\mathtt{i}}.\end{aligned} \tag{2.3}$$

A truely invalid result is represented by $\mathrm{NaN} + \mathrm{NaN}\mathtt{i}$.

Of course this situation gives rise to some new problems, such as in the expression

$$(+0) \oslash (+0) \oplus (y + \infty\mathtt{i}) = \mathrm{NaN} + \infty\mathtt{i}$$

which is incorrectly being interpreted as a representation of complex infinity instead of invalid, because of $(+0) \oslash (+0)$.

The fact that now a different number of representations for zero and infinity exists, namely 4 versus 8, leads to violations of the floating-point identity $1/(1/x) = x$. The latter holds for all nonzero x and also for x being either a real signed zero or a real signed infinity. However, it is easy to see that the identity does not hold anymore for the representations (2.3). With $x = \mathrm{NaN} - \infty\mathtt{i}$, which represents $|\infty| \exp^{[-\pi,0]\mathtt{i}}$:

$$\begin{aligned}1 \oslash (\mathrm{NaN} - \infty\mathtt{i}) &= (+0) + (+0)\mathtt{i},\\ 1 \oslash (+0 + 0\mathtt{i}) &= +\infty + \mathrm{NaN}\mathtt{i} \neq \mathrm{NaN} - \infty\mathtt{i}.\end{aligned}$$

Moreover, the 4 additional representations for large complex numbers in one of the 4 possible halfplanes do not cover all the cases. Take for instance the expression

$$(\mathrm{NaN} + \infty\mathtt{i}) \otimes (\mathrm{NaN} - \infty\mathtt{i})$$

which should be interpreted as

$$(|\infty| \exp^{[0,\pi]\mathtt{i}}) \times (|\infty| \exp^{[-\pi,0]\mathtt{i}}) = |\infty| \exp^{[-\pi,\pi]\mathtt{i}}. \tag{2.4}$$

Its result is a complex number that can lie in any of the four quadrants. For this result Annex G does not provide a representation. After recomputation, as recommended in the cited guideline for the multiplication, the product returned by Annex G is erroneously $-\infty + \infty\mathtt{i}$.

3. A Proposal for Reliable Complex Arithmetic

Let us take another look at example (2.1). The result of the correct interpretation (2.2) could also be stored as (we detail our implementation further down)

$$|\infty| \exp^{[-\pi,0]\mathtt{i}} = (\mp\infty) + (-\infty)\mathtt{i}$$

instead of using NaN$-\infty\mathtt{i}$ as representation for the lower halfplane. By introducing three possible "signs," namely positive, negative and insecure, denoted respectively by +, $-$, and $\mp$, one obtains 9 representations for complex infinity,

$$(*\infty) + (\star\infty)\mathtt{i} \qquad *, \star \in \{+, -, \mp\} \tag{3.1}$$

and 9 representations for complex zero,

$$(*0) + (\star 0)\mathtt{i} \qquad *, \star \in \{+, -, \mp\}. \tag{3.2}$$

A simple verification shows that in this way also the identity $1 / (1 / x) = x$ can be restored.

Moreover, complex numbers $x + y\mathtt{i}$ of which real and/or imaginary part equal NaN, can now be reserved for truely invalid results. In this way, the ambiguity introduced in Annex G about NaN parts in complex numbers is eliminated.

When we return to expression (2.4), it is clear that with the introduction of (3.1), we can now store

$$(|\infty| \exp^{[0,\pi]\mathtt{i}}) \times (|\infty| \exp^{[-\pi,0]\mathtt{i}}) = |\infty| \exp^{[-\pi,\pi]\mathtt{i}} = (\mp\infty) + (\mp\infty)\mathtt{i}.$$

Tables 1 and 2 contain some more examples of expressions that cannot be dealt with correctly by Annex G and necessitate the new approach. For operands representing a halfplane, we have used the Annex G notation.

For the correct implementation of the basic operations on complex operands, making use of the indispensable additional special values, the sign (+, $-$, or $\mp$) accompanying intermediate NaN results in the real and/or imaginary part, can be useful to determine the quadrant(s) containing the result. For instance:

$$\begin{aligned}
(2.0 + 0.0\mathtt{i}) \otimes (3.0 + \infty\mathtt{i}) &= (6.0 \ominus +\mathrm{NaN}) + (+\infty \oplus 0.0)\mathtt{i} \\
&= \mp\mathrm{NaN} + \infty\mathtt{i} \to \mp\infty + \infty\mathtt{i}, \\
(0.0 + 3.0\mathtt{i}) \otimes (2.0 + \infty\mathtt{i}) &= (0.0 \ominus +\infty) + (6.0 \oplus +\mathrm{NaN})\mathtt{i} \\
&= -\infty + \mathrm{NaN}\mathtt{i} \to -\infty + \infty\mathtt{i}.
\end{aligned}$$

Moreover, while Annex G says that for the implementation of $\exp(-\infty + \infty\mathtt{i})$, the signs returned for the real and imaginary zero parts of the result are at the implementor's choice, we can correctly return

$$\exp(-\infty + \infty\mathtt{i}) = (\mp 0) + (\mp 0)\mathtt{i}.$$

In the same way

$$\log(-0 + 0\mathtt{i}) = -\infty + (\mp\infty)\mathtt{i}$$

while Annex G suggests $\log(-0 + 0\mathtt{i}) = -\infty + \pi\mathtt{i}$.

Table 1. Some exceptional cases for the basic operations. $\lightning$ is used to indicate a mathematically incorrect result, because of its interpretation by Annex G.

operation	C++ Annex G	New Style
$(+\infty+\infty\,\mathtt{i})\otimes(-\infty-\infty\,\mathtt{i})$	$\mathrm{NaN}-\infty\,\mathtt{i}$	$(\mp\infty)-\infty\,\mathtt{i}$
$(-\infty+\infty\,\mathtt{i})\otimes(3.0+2.0\,\mathtt{i})$	$-\infty+\mathrm{NaN}\,\mathtt{i}$	$-\infty+(\mp\infty)\,\mathtt{i}$
$(-\infty+\infty\,\mathtt{i})\oslash(3.0+2.0\,\mathtt{i})$	$\mathrm{NaN}+\infty\,\mathtt{i}$	$(\mp\infty)+\infty\,\mathtt{i}$
$(+\infty+\mathrm{NaN}\,\mathtt{i})\otimes(-\infty+\infty\,\mathtt{i})$	$-\infty+\infty\,\mathtt{i}\ \lightning$	$(\mp\infty)+(\mp\infty)\,\mathtt{i}$
$(\mathrm{NaN}-\infty\,\mathtt{i})\oslash(3.0+2.0\,\mathtt{i})$	$-\infty-\infty\,\mathtt{i}\ \lightning$	$(\mp\infty)+(\mp\infty)\,\mathtt{i}$
$(+\infty+\infty\,\mathtt{i})\oplus(-\infty+\infty\,\mathtt{i})$	$\mathrm{NaN}+\infty\,\mathtt{i}$	$(\mp\infty)+\infty\,\mathtt{i}$
$(\mathrm{NaN}+\infty\,\mathtt{i})\oplus(-\infty+\infty\,\mathtt{i})$	$\mathrm{NaN}+\infty\,\mathtt{i}\ \lightning$	$\mathrm{NaN}+\mathrm{NaN}\,\mathtt{i}$
$(+\infty-\infty\,\mathtt{i})\otimes(\mathrm{NaN}+2.0\,\mathtt{i})$	$+\infty+\infty\,\mathtt{i}\ \lightning$	$\mathrm{NaN}+\mathrm{NaN}\,\mathtt{i}$
$(\mathrm{NaN}+2.0\,\mathtt{i})\otimes(x+1.0\,\mathtt{i})$[1]	$-\infty+\infty\,\mathtt{i}\ \lightning$	$\mathrm{NaN}+\mathrm{NaN}\,\mathtt{i}$
$\mathrm{NaN}\oplus(5.0+\infty\,\mathtt{i})$	$\mathrm{NaN}+\infty\,\mathtt{i}\ \lightning$	$\mathrm{NaN}+\infty\,\mathtt{i}$
$1.0\oslash(+\infty-\infty\,\mathtt{i})$	$+0.0+0.0\,\mathtt{i}$	$+0.0+0.0\,\mathtt{i}$
$1.0\oslash(\mathrm{NaN}-\infty\,\mathtt{i})$	$+0.0+0.0\,\mathtt{i}\ \lightning$	$(\mp0.0)+0.0\,\mathtt{i}$
$1.0\oslash(+0.0+0.0\,\mathtt{i})$	$+\infty+\mathrm{NaN}\,\mathtt{i}\ \lightning$	$+\infty-\infty\,\mathtt{i}$
$(+0.0+0.0\,\mathtt{i})\oplus(-0.0+0.0\,\mathtt{i})$	$+0.0+0.0\,\mathtt{i}\ \lightning$	$(\mp0.0)+0.0\,\mathtt{i}$
$(+0.0+0.0\,\mathtt{i})\oplus(-0.0-0.0\,\mathtt{i})$	$+0.0+0.0\,\mathtt{i}\ \lightning$	$(\mp0.0)+(\mp0.0)\,\mathtt{i}$
$((\mp0.0)+0.0\,\mathtt{i})\oplus(-0.0+0.0\,\mathtt{i})$	–	$(\mp0.0)+0.0\,\mathtt{i}$
$(2.0+\infty\,\mathtt{i})\otimes(0.0+3.0\,\mathtt{i})$	$-\infty+\mathrm{NaN}\,\mathtt{i}\ \lightning$	$-\infty+\infty\,\mathtt{i}$
$(7.0+0.0\,\mathtt{i})\otimes(4.0-0.0\,\mathtt{i})$	$+28.0+0.0\,\mathtt{i}\ \lightning$	$28.0+(\mp0.0)\,\mathtt{i}$
$(-6.0+0.0\,\mathtt{i})\oplus(-9.0-0.0\,\mathtt{i})$	$-15.0+0.0\,\mathtt{i}\ \lightning$	$-15.0+(\mp0.0)\,\mathtt{i}$

[1] $2\otimes x\geq\beta^{U+1}$

Table 2. Some exceptional cases for the square root.

operand[2]	C++ Annex G	operand[3]	New Style
$\pm\,0.0+0.0\,\mathtt{i}$	$+0.0+0.0\,\mathtt{i}$	$(*0.0)+0.0\,\mathtt{i}$	$+0.0+0.0\,\mathtt{i}$
$a+\infty\,\mathtt{i}$	$+\infty+\infty\,\mathtt{i}$	$a+\infty\,\mathtt{i}$	$+\infty+\infty\,\mathtt{i}$
$\mathrm{NaN}+\infty\,\mathtt{i}$	$+\infty+\infty\,\mathtt{i}$	$(\mp\infty)+\infty\,\mathtt{i}$	$+\infty+\infty\,\mathtt{i}$
$-\infty+b\,\mathtt{i}$	$+0.0+\infty\,\mathtt{i}$	$-\infty+b\,\mathtt{i}$	$+0.0+\infty\,\mathtt{i}$
$+\infty+b\,\mathtt{i}$	$+\infty+0.0\,\mathtt{i}$	$+\infty+b\,\mathtt{i}$	$+\infty+0.0\,\mathtt{i}$
$-\infty+\mathrm{NaN}\,\mathtt{i}$	$\mathrm{NaN}\pm\infty\,\mathtt{i}\ \lightning$	$+\infty+(\mp\infty)\,\mathtt{i}$	$(\mp\infty)+(\mp\infty)\,\mathtt{i}$
$+\infty+\mathrm{NaN}\,\mathtt{i}$	$+\infty+\mathrm{NaN}\,\mathtt{i}$	$+\infty+(\mp\infty)\,\mathtt{i}$	$+\infty+(\mp\infty)\,\mathtt{i}$
–	–	$(\mp\infty)+(\mp\infty)\,\mathtt{i}$	$(\mp\infty)+(\mp\infty)\,\mathtt{i}$

[2] *using the Annex G representations*
[3] *using the new representations*

4. Concluding Remarks

The signed zeroes have been the subject of a lot of debate in the scientific community. In complex floating-point arithmetic the IEEE signed infinities raise a similar problem. While the introduction of a zero with uncertain sign, namely ∓ 0, is not new (see [2]), its importance grows when considering complex arithmetic.

In IEEE floating-point arithmetic, it is the correct result for

$$(+0) + (-0) = \mp 0.$$

In complex arithmetic, it can be used for underflowing quantities of the type

$$\lim_{n \to \infty} \big(-1 / (2 + \mathtt{i}) \big)^n = (\mp 0) + (\mp 0)\mathtt{i}.$$

In addition, one, two or all four halfplanes of the complex plane can be represented by supporting $\mp\infty$. Even the 3 quadrant result

$$\frac{+\infty + (\mp\infty)\mathtt{i}}{+1 - 2\mathtt{i}} = |\infty| \exp^{[-\pi/2, \pi]\mathtt{i}} \subseteq |\infty| \exp^{[-\pi, \pi]\mathtt{i}}$$

can be contained in $(\mp\infty) + (\mp\infty)\mathtt{i}$.

While the Annex G guidelines were probably the best that could be achieved when solely making use of the IEEE real floating-point representations, it is clear that a reliable implementation of complex floating-point arithmetic needs additional special values.

References

1. ANSI/ISO/IEC 9899:1999: *Annex G: IEC 60559-Compatible Complex Arithmetic (Informative)*, ANSI, 1999, pp. 465–478.
2. Juric, Z.: *TIGCC—Routines for Floating Point Arithmetic*, Texas Instruments, `http://tigcc.ticalc.org/doc/timath.html`.
3. Kahan, W,: Branch Cuts for Complex Elementary Functions, or Much Ado about Nothing's Sign Bit, in: Iserles, A. and Powell, M. J. D. (eds), *The State of the Art in Numerical Analysis*, Oxford University Press, Oxford, 1987, pp. 165–211.
4. *Language Independent Arithmetic—Part 3: Complex Integer and Floating Point Arithmetic and Complex Elementary Numerical Functions*, Technical Report, WD 10967–3, ISO/IEC, 2001, `http://anubis.dkuug.dk/JTC1/SC22/WG11/docs/n476.pdf`.
5. Ziv, A.: Sharp Ulp Rounding Error Bound for the Hypotenuse Function, *Math. Comp.* **68** (227) (1999), pp. 1143–1148.