Review of "The Schubfach way to render doubles"

Jean-Michel Muller and Paul Zimmermann

October 14, 2021

Abstract

This is a review of the article "The Schubfach way to render doubles" written by Raffaello Giulietti, dated from 2020-03-16 (file `Schubfach2.pdf` downloaded October 14, 2021 from `https://drive.google.com/file/d/1luHhyQF9zKlM8yJ1nebU0OgVYhfC6CBN/view`, md5sum `6c212c90ebbea4fcfe2bb6d9ded59104`).

This article, while quite technical, is extremely well written, and the underlying algorithm is very clever. As said by Guy Steele in `https://github.com/openjdk/jdk/pull/3402#issuecomment-942063994`, it would be a pity to not publish an academic paper. Moreover your algorithm would be alive: once published, people will improve it, generalize it (for example to larger formats than double precision), and maybe correct it (if we missed some error). We both agree to advise you for a potential submission (journals such as ACM Transactions on Mathematical Software do accept articles of 27 pages). In such a case, maybe more bibliography would be welcome: in particular the article of Matula [3] and that of Boldo and Melquiond on rounding to odd [1] are both relevant.

References

[1] Boldo, S., and Melquiond, G. When double rounding is odd. In *17th IMACS World Congress* (Paris, France, July 2005), p. 11.

[2] Hardy, G. H., and Wright, E. M. *An Introduction to the Theory of Numbers*, 4th ed. Clarendon Press, Oxford, 1960.

[3] Matula, D. W. In-and-out conversions. *Commun. ACM 11*, 1 (1968), 47–50.

[4] Muller, J.-M., Brunie, N., de Dinechin, F., Jeannerod, C.-P., Joldes, M., Lefèvre, V., Melquiond, G., Revol, N., and Torres, S. *Handbook of Floating-point Arithmetic (2nd edition)*. Birkhäuser Basel, July 2018.

1 Major Comments

The examples with large decimal expansions for exact conversion can be anticipated: if the extremal exponents $e_{\min}$ and $e_{\max}$ of the binary format satisfy $e_{\min} \approx -e_{\max}$ (which holds for all usual formats) and if p is its precision, then the largest width of a decimal significand we can obtain by exact conversion is (in digits):

$$-e_{\min} + p + \lfloor (e_{\min} + 1)\log_{10}(2) - \log_{10}(1 - 2^{-p}) \rfloor.$$

This is taken literally from Section 4.9 of [4] (page 143).

Instead of producing the *shortest* string that rounds back to the original number v, one could produce a string with $H = 17$ decimal digits for double precision numbers (cf Figure 3), which always ensures we recover v. Can Schubfach be modified for this case?

In Section 7, a figure would be welcome to show R_v, D_k and D_{k+1}.

In Result 13, since you don't use the right-hand side of (5), you can check (for example using interval arithmetic) for which largest interval $e_{\min} \le e \le e_{\max}$ the statement holds. We find $-3\,606\,689 \le e \le 3\,150\,619$ instead of $-2\,956\,395 \le e \le 2\,500\,325$. Same for Result 14, where we find $-6\,432\,162 \le e \le 6\,432\,162$. For Result 15, this does not improve the bound. Also, why did you choose those particular values $Q = 41$ and $Q = 38$? For double precision, $Q = 22$ would suffice.

Result 20. There is no need to resort to a mechanical proof for this, if you use the following classical result about continued fractions (see for example Theorem 182 page 151 of [2]):

Theorem 1 *Let x be a positive real number, and (p_i/q_i) be the list of convergents from the continued fraction of x. Then for any positive integers c, d, with $c < q_i$, we have $|q_i x - p_i| < |cx - d|$.*

In other terms, the $|q_i x - p_i|$ form successive records (in decreasing value). In your case, x is either $2^q D^{-k}$ or $2^{q+1} D^{-k}$, where k depends from q (in fact, at most two values of k are possible for any value of q, $Q_{\min} \le q \le Q_{\max}$). Then for each pair (q, k), it suffices to compute the (finite) continued fraction expansion of x. For the bound on $2V$, let p_i/q_i be the last convergent of $x = 2^{q+1} D^{-k}$ such that $q_i < 2^P$ and $x \neq p_i/q_i$. Then for $c < 2^P$, the value of $|cx - d|$, if not zero, is lower bounded by $|q_i x - p_i|$, which gives a lower bound between $2V = cx$ and the nearest integer. A simple program (for example using the SageMath computer algebra system) yields an explicit value for ϵ, and the corresponding values of q, k and c for double precision: $\epsilon = 1323359619378521 \cdot 5^{-49} \approx 7.45 \cdot 10^{-20}$, attained for $q = 164$, $k = 49$, and $c = 5592117679628511$.

For the V_l and V_r bounds, it is slightly trickier to prove Result 20, and to find the best possible bound and the corresponding inputs. Indeed, for V_l and V_r, excluding the "irregular spacing" case (which can be checked by exhaustive search), we want a lower bound for the distance between $(2c \pm 1)2^q D^{-k}$ and the nearest integer, with $2c \pm 1 < 2^{P+1}$. This corresponds to convergents of $x = 2^q D^{-k}$ with odd denominator less than 2^{P+1}. A non-optimal bound could be obtained by applying Theorem 1 on convergents of x with denominator less than 2^{P+1} (not only those with odd denominator). Luckily, in the case of double precision, the convergents of $2^{q+1} D^{-k}$ with unconstrained denominator less than 2^P (i.e., what was used above to obtain the bound for $2V$) yield the same lower bound (when considering all possible values of q and k) than those of $2^q D^{-k}$ with unconstrained denominator less than 2^{P+1}, thus the above bound for $2V$ holds also for V_l and V_r (and we know it is attained for $2V$). The same idea applies to the other formats of Figure 3.

In the algorithm from Figure 5, some processors have an assembly instruction which computes both y_0 and y_1 at the same time.

Section 10: why split in $1 + 8 + 8$, and not in say $5 + 6 + 6$?

Section 14: the definition of g_1 and g_0 could be more detailed: g fits into a long, thus we guess you mean $g_0 = 0$ and not $g_0 > 0$. You have $\bar{c} < 2^{26}$ from (9), thus $\bar{c}g < 2^{89}$ fits into two longs.

2 Minor Comments

Maybe some more classical notations could be used, for example for the bitwise or, or for the integer quotient and remainder.

Although Section 3.2.1 mentions a generic rounding, the article only consider rounding to nearest (with ties to even). We only checked correctness in this setting. It is not clear that results remains true otherwise. In particular for the round-trip property to hold (Section 3.4), if the conversion from string to double is rounded towards zero, then one should use a rounding away from zero for the conversion from double to string. See also Figure 1 which considers a generic rounding.

In Figure 1, the term *significand* is not defined in Section 2.

Examples 1 and 2 are with numbers in the subnormal range. Could similar examples in the normal range be given (maybe with larger significands)? Otherwise the reader might think that such examples do only occur in the subnormal range.

In Figure 3, the last row (S=256) is not in IEEE 754-2019. Where did you find these parameters?

In the whole article, it would be better to use ℓ instead of l, in particular in v_l and c_l.

Definition 3: please add that x is a D-ary. As we understand, len$x$ is only defined for x in shortest form. However below in 6.3 you consider len$x$ for $x = dD^i$ not in shortest form.

Page 8, Section 6.3, line 10: $b \neq D^l$ should be $b \neq dD^l$.

Result 6: is y in D_j for some j?

Result 7: the first assumption is not well phrased. For example it could be $\text{len}(dD^i) = \text{len}(dD^j)$ for $\ell \leq i < j \leq r$, and the common length m satisfies $m > n$.

Definition 4: R_i should be $R_i(v)$ since it depends on v. At least explain that you write R_i because v is fixed.

The reasoning for Result 8 could be simpler: if both x and y are in R_i, then $|x - y| = kD^i$ and $|x - y| \leq ||R_v||$, which since $D^i > ||R_v||$ implies $k = 0$.

Page 9, middle: "is equivalent" should read "is equivalent to".

End of page 9: `MIN_VALUE` and `MAX_VALUE` are not defined.

Page 10, line 5: "that and" should read "that".

In the skeleton of Schubfach (8.1), please explain why $s < D^M$ implies that v is very small. "Its boundaries are outside R_v": they can be on the bounds of R_v, no? In the case $s \geq D^M$, second item: why len$y > M$? Can we have an example with $T \neq \{u'\}$? In this skeleton, you use a variable y: how y is defined?

Page 11, line 3, "from R6": please say to which values you apply R6, what are n, d, x and y from the statement of R6?

Page 11: "so $s < D$ means that ... is subnormal": why?

Page 14, line 7: the left inequality on the left can be replaced by $<$. Idem for the left inequality two lines below.

Page 16, "because $s', t', ...$ are not guaranteed to be even": but what appears in fact is $s'D$ and $t'D$ which are even for $D = 10$.

Page 17, line 5: "every vale" should read "every value".

Page 17, "fitting in a long": this assumes a "long" is 64-bit wide, which is not true on all processors in the C language (maybe it is true for Java?).

Page 18, first line of Section 9.7: "the estimate can replace" should be "the estimates can replace".

Definition 9: could $<$ be $\leq$ in the first branch?

Section 9.9.2: we have checked using a Sage script that the hexadecimal values for g_1 and g_0 are correct in the file `MathUtils.java`.

Page 20, last line: the condition $121 \leq -(\bar{q} + r + 1) \leq 124$ should read $123 \leq -(\bar{q} + r + 1) \leq 126$, which if we replace in $h = 128 + (\bar{q} + r + 1)$ page 21 gives $2 \leq h \leq 5$ as claimed.

Page 23, "Translating this pieces" should read "Translating this piece" or "Translating these pieces".