Dragonbox: A New Floating-Point Binary-to-Decimal Conversion Algorithm

Junekey Jeon

The Department of Mathematics
University of California, San Diego
USA
j6jeon@ucsd.edu

Abstract

We present a new algorithm for efficiently converting a binary floating-point numbers into shortest and correctly rounded decimal representation. The algorithm is based on *Schubfach* algorithm [1] introduced in around 2017-2018, and is also inspired from Grisu [2] and Grisu-Exact [4]. In addition to the core idea of Schubfach, Dragonbox utilizes some Grisu-like ideas to minimize the number of expensive 128-bit $\times$ 64-bit multiplications, at the cost of having more branches and divisions-by-constants. According to our benchmarks, Dragonbox performs better than Ryū, Grisu-Exact, and Schubfach for both IEEE-754 binary32 and binary64 formats.

0. Disclaimer

This paper is not a completely formal writing, and is not intended for publications into peer-reviewed conferences or journals. The paper might contain some alleged claims and/or lack of references.

1. Introduction

Due to recent popularity of JavaScript and JSON, interest on fast and correct algorithm for converting between binary and decimal representations of floating-point numbers has been continuously increasing. As a consequence, many new algorithms have been proposed recently, in spite of the long history of the subject.

We will assume all floating-point numbers are in either IEEE-754 binary32 or binary64 formats, as these are the most common formats used today.[1][2] We will also focus on the binary-to-decimal conversion in this paper and will not discuss how to do decimal-to-binary conversion. Contrary to one might think, in fact decimal-to-binary conversion and binary-to-decimal converseion are largely asymmetric, because of the asymmetric nature of input and output. In general, for the input side, one needs to deal with wide variety of possible input data, but the form of output is usually definitive. On the other hand, for the output side, the input data has a strict format but one needs to choose between various possibilities of outputs. Floating-point I/O is not an exception. When it comes to decimal-to-binary conversion, which corresponds to the input side, the input data can be usually arbitrarily long so we have to somehow deal with that, but any input data can, if not malformed, usually represent a unique floating-point number. On the other hand, in binary-to-decimal conversion, which corresponds to the output side, the input is a single binary floating-point number but the output can be all decimal numbers which any correct parser will read as the original floating-point number. To resolve this ambiguity, Steele and White proposed the following criteria in [6]:[3]

1. **Information preservation**: a correct decimal-to-binary converter must return the original binary floating-point number,
2. **Minimum-length output**: the output decimal significand should be as short as possible, and
3. **Correct rounding**: among all possible shortest outputs, the one that is closest to the true value of the given floating-point number should be chosen.

[1] Details of these formats will be reviewed in Section 2

[2] It should be not so difficult to generalize Dragonbox to similar formats, such as IEEE-754 binary16 or binary128.

[3] To be precise, the criteria given by Steele and White were in terms of the character string generated from the decimal representation. However, we can write those criteria in terms of the decimal representation itself as well.

Notable examples of recently proposed binary-to-decimal conversion algorithms include but not limited to Grisu [2], Errol [3], Ryū [5], and Grisu-Exact [4]. Among these, Errol, Ryū, and Grisu-Exact satisfy all of the above criteria. Grisu does not satisfy all of the criteria, but Grisu3, which can detect its failure to satisfy the criteria, with the fallback into Dragon4 [6], proposed by Steele and White and satisfies all the criteria, is still popular.

Schubfach [1] is another example of those algorithms, developed in around 2017-2018, but it seems that, compared to Ryū, it did not get much attention from the public probably because at that time there was no document explaining details of the algorithm. Nevertheless, the underlying idea of Schubfach is theoretically very appealing and its implementation [7] also seems to outperform that of the other algorithms.

Although Schubfach is already a very tight algorithm, there can be ways to improve its performance further. One possible way might be to eliminate the necessity to perform three 128-bit $\times$ 64-bit multiplications all the time. The core idea of Dragonbox is to achieve this by applying some Grisu-like ideas to Schubfach.

2. IEEE-754 Specifications[4]

Before diving into the details of Dragonbox, let us review IEEE-754 and fix some related notations. For a real number w, by *(binary) floating-point representation* we mean the representation

$$w = (-1)^{\sigma_w} \cdot F_w \cdot 2^{E_w}$$

where $\sigma_w = 0, 1$, $0 \le F_w < 2$, and E_w is an integer. We say the above representation is *normal* if $1 \le F_w < 2$. Of course, there is no normal floating-point representation of 0, while any other real number has a unique normal floating-point representation. If the representation is not normal, we say it is *subnormal*.

IEEE-754 specifications consist of the following rules that define a mapping from the set of fixed-length bit patterns $b_{q-1}b_{n-2}\cdots b_0$ for some q into the real line augmented with some special values:

1. The most-significant bit b_{q-1} is the sign σ_w.
2. The least-significant p-bits $b_{p-1}\cdots b_0$ are for storing the significand F_w, while the remaining $(q-p-1)$-bits are for storing the exponent E_w. We call p the *precision* of the representation.[5]
3. If $q-p-1$ exponent bits are not all-zero nor all-one, the representation is normal. In this case, we compute F_w as
$$F_w = 1 + 2^{-p} \cdot \sum_{k=0}^{p-1} b_k \cdot 2^k$$
and E_w as
$$E_w = -(2^{q-p-2}-1) + \sum_{k=0}^{q-p-2} b_{p+k} \cdot 2^k.$$
The constant term $2^{q-p-2}-1$ is called the *bias*, and we denote this value as $E_{\max} \coloneqq 2^{q-p-2}-1$.
4. If $q-p-1$ exponent bits are all-zero, the representation is subnormal. In this case, we compute F_w as
$$F_w = 2^{-p} \cdot \sum_{k=0}^{p-1} b_k \cdot 2^k$$
and let $E_w = -(2^{q-p-2}-2)$. Let us denote this value of E_w as $E_{\min} \coloneqq -(2^{q-p-2}-2)$.
5. If $q-p-1$ exponent bits are all-one, the pattern represents either $\pm\infty$ when all of p significand bits are zero, or NaN's (Not-a-Number) otherwise.

When $(q,p) = (32, 23)$, the resulting encoding format is called *binary32*, and when $(q,p) = (64, 52)$, the resulting encoding format is called *binary64*.

For simplicity, let us only consider bit patterns corresponding to positive real numbers from now on. Zeros, infinities, and NaN's should be treated specially, and for negative numbers, we can simply ignore the sign until the final output string is generated. Hence, for example, we do not think of all-zero nor all-one patterns, and especially exponent bits are never all-one. Also, we always assume that the sign bit is 0. With these assumptions, the mapping defined above is one-to-one: each bit pattern corresponds to a unique real number, and no different bit patterns correspond to a same real number.

From now on, by saying $w = F_w \cdot 2^{E_w}$ a *floating-point number* we implicitly assumes that

(1) w is a positive number representable within an IEEE-754 binary format with some q and p, and

(2) F_w and E_w are those obtained from the rules above.

In particular, the representation is normal ($1 \le F_w < 2$) if $E_w \ne E_{\min}$ and is can be subnormal ($0 \le F_w < 1$) only if $E_w = E_{\min}$. If the representation is normal, we call w a *normal number*, and for otherwise, we call w a *subnormal number*.

For a floating-point number $w = F_w \cdot 2^{E_w}$, we define w^- as the greatest floating-point number smaller than w. When w is the minimum possible positive floating-number representable within the specified encoding format, that is, $w = 2^{-p} \cdot 2^{E_{\min}}$, then we define $w^- = 0$. Similarly, we define w^+ as the smallest floating-point number greater than w. Again, if w is the largest possible finite number representable within the format, that is, $w = (2-2^{-p})2^{E_{\max}}$, then we define $w^+ \coloneqq 2^{E_{\max}+1}$.

[4] This section is mostly copied from [4].

[5] Usually, it is actually $p+1$ that is called the precision of the format in other literatures. However, we call p the precision in this paper for simplicity.

In general, it can be shown that

$$w^- = \begin{cases} (F_w - 2^{-p-1})2^{E_w} & \text{if } F_w = 1 \text{ and } E_w \neq E_{\min} \\ (F_w - 2^{-p})2^{E_w} & \text{otherwise} \end{cases}$$

and

$$w^+ = (F_w + 2^{-p})2^{E_w}.$$

We will also use the notations

$$m_w^- \coloneqq \frac{w^- + w}{2} = \begin{cases} (F_w - 2^{-p-2})2^{E_w} & \text{if } F_w = 1 \text{ and} \\ & E_w \neq E_{\min} \\ (F_w - 2^{-p-1})2^{E_w} & \text{otherwise} \end{cases},$$

$$m_w^+ \coloneqq \frac{w + w^+}{2} = (F_w + 2^{-p-1})2^{E_w}$$

to denote the midpoints of the intervals $[w^-, w]$, $[w, w^+]$, respectively.

2.1 Rounding Modes

Floating-point calculations are inherently imprecise as the available precision is limited. Hence, it is necessary to round calculational results to make them fit into the precision limit. Specifying how any rounding should be performed means to define for each real number a corresponding floating-point number in a consistent way. IEEE-754 currently defines five rounding modes. We can describe those rounding modes by specifying the inverse image in the real line of each floating-point number w:

1. *Round to nearest, ties to even*: If the LSB (Least Significant Bit) of the significand bits of w is 0, then the inverse image is the closed interval $[m_w^-, m_w^+]$. Otherwise, it is the open interval (m_w^-, m_w^+). This is the default rounding mode in most of the platforms. In fact, it is required to be the default mode for binary encodings.
2. *Round to nearest, ties away from zero*: The inverse image of w is the half-open interval $[m_w^-, m_w^+)$. This mode is introduced in the 2008 revision of the IEEE-754 standard. Some platforms and languages, such as the recent standards of the C and C++ languages, do not have the corresponding way of representing this rounding mode.
3. *Round toward* 0: The inverse image of w is the half-open interval $[w, w^+)$.
4. *Round toward* $+\infty$: The inverse image of w is the half-open intervals $(w^-, w]$ if w is positive, and $[w, w^+)$ if w is negative.[6]
5. *Round toward* $-\infty$: The inverse image of w is the half-open intervals $[w, w^+)$ if w is positive, and $(w^-, w]$ is w is negative.

[6] We supposed to deal only with positive numbers, so w here is actually a positive number. The phrases "if w is positive" or "if w is negative" simply mean that the original input is positive or negative, respectively.

Though not included in the IEEE-754 standard, we can think of the following additional rounding modes with their obvious meanings:

- *Round to nearest, ties to odd*
- *Round to nearest, ties toward zero*
- *Round to nearest, ties toward* $+\infty$
- *Round to nearest, ties toward* $-\infty$
- *Round away from* 0

Note that if I is the interval given as the inverse image of w according to a given rounding mode, then a correct decimal-to-binary converter must output w from any numbers in I. Therefore, in order to produce a shortest possible decimal representation of w, we need to search for a number inside I that has the least number of decimal significand digits.

2.2 Notations

From now on, we will assume that a floating-point number w and a specific rounding mode is given so the interval I is defined accordingly. Note that for all cases I is an interval contained in the positive real axis and it avoids 0. We will denote the left and the right endpoints of I as w_L and w_R, respectively. For example, when one of the round-to-nearest rounding mode is specified, $w_L = m_w^-$ and $w_R = m_w^+$. We will also denote the length of I as $\Delta \coloneqq w_R - w_L$. Note that there are only three possible values of Δ:

1. $\Delta = 2^{E_w - p - 1}$, if $w_L = w^-$, $w_R = w$, $F_w = 1$, and $E_w \neq E_{\min}$,
2. $\Delta = 3 \cdot 2^{E_w - p - 2}$ if $w_L = m_w^-$, $w_R = m_w^+$, $F_w = 1$, and $E_w \neq E_{\min}$, and
3. $\Delta = 2^{E_w - p}$ for all other cases.

We also denote

$$e \coloneqq E_w - p, \quad f_c \coloneqq F_w 2^p$$

so that f_c is an integer and

$$w = f_c \cdot 2^e,$$

$$w^- = \begin{cases} \left(f_c - \frac{1}{2}\right) \cdot 2^e & \text{if } F_w = 1 \text{ and } E_w \neq E_{\min} \\ (f_c - 1) \cdot 2^e & \text{otherwise} \end{cases},$$

$$w^+ = (f_c + 1) \cdot 2^e,$$

$$m_w^- = \begin{cases} \left(f_c - \frac{1}{4}\right) \cdot 2^e & \text{if } F_w = 1 \text{ and } E_w \neq E_{\min} \\ \left(f_c - \frac{1}{2}\right) \cdot 2^e & \text{otherwise} \end{cases},$$

$$m_w^+ = \left(f_c + \frac{1}{2}\right) \cdot 2^e.$$

With this notation, Δ is one of 2^{e-1}, $3 \cdot 2^{e-2}$, or 2^e.

3. Review of Schubfach

In this section, we will briefly review how Schubfach works. Most of the results are from [1], but we changed the nota-

tions and formulations, and also rewrote the proofs to help understanding the rest of our paper.

The beauty of Schubfach is that, not like Ryū or Grisu-Exact, it does not perform an iterative search to find the shortest decimal representation. Rather, Schubfach finds it with just one trial using the following simple fact:[7]

Proposition 3.1.
Let $k_0 \coloneqq -\lfloor \log_{10} \Delta \rfloor$. *Then*

1. $\left|I \cap 10^{-k_0+1}\mathbb{Z}\right| \le 1$, *and*
2. $\left|I \cap 10^{-k_0}\mathbb{Z}\right| \ge 1$.[8]

where $|\cdot|$ *denotes the cardinality the set and for any* $a \in \mathbb{R}$ *and* $A \subseteq \mathbb{R}$, aA *denotes the set* $\{av \colon v \in A\}$.

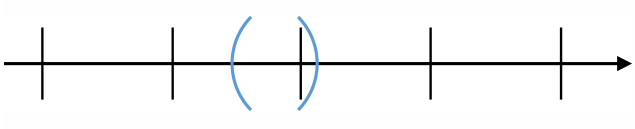

Figure 1. If I is shorter than the unit, then it contains at most one lattice point

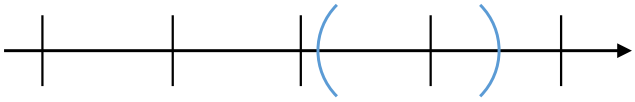

Figure 2. If I is longer than the unit, then it contains at least one lattice point

Proof. By definition of k_0, we have

$$-k_0 \le \log_{10} \Delta < -k_0 + 1,$$

or equivalently,

$$10^{-k_0} \le \Delta < 10^{-k_0+1}.$$

If $\left|I \cap 10^{-k_0+1}\mathbb{Z}\right| > 1$, then it means there are at least two distinct points in I which are apart from each other by distance 10^{-k_0+1}. Hence, the length of I should be at least 10^{-k_0+1}, or equivalently,

$$\Delta \ge 10^{-k_0+1},$$

which is a contradiction. This shows the first claim.

On the other hand, pick any point $v \in I$, then we know

$$\left\lfloor 10^{k_0} v \right\rfloor \le 10^{k_0} v < \left\lfloor 10^{k_0} v \right\rfloor + 1.$$

We claim that at least one of $\left\lfloor 10^{k_0} v \right\rfloor$ and $\left\lfloor 10^{k_0} v \right\rfloor + 1$ is in $10^{k_0} I$. Suppose not, then the left endpoint of $10^{k_0} I$ should lie inside $\left[\left\lfloor 10^{k_0} v \right\rfloor, 10^{k_0} v\right]$ and the right endpoint of $10^{k_0} I$ should lie inside $\left[10^{k_0} v, \left\lfloor 10^{k_0} v \right\rfloor + 1\right]$. This implies that the length of $10^{k_0} I$ is at most 1, but since $10^{-k_0} \le \Delta$, it follows that $\Delta = 10^{-k_0}$ and $10^{k_0} I = \left(\left\lfloor 10^{k_0} v \right\rfloor, \left\lfloor 10^{k_0} v \right\rfloor + 1\right)$.

Note that $\Delta = 10^{-k_0}$ is only possible for very rare cases; indeed, since 5 does not appear as a prime factor of Δ (as a rational number), the equality $\Delta = 10^{-k_0}$ can hold only when $k_0 = 0$. Hence, we have $\Delta = 1$, which can hold only when $e = 1$ or $e = 0$ because Δ is one of 2^{e-1}, $3 \cdot 2^{e-2}$, or 2^e, depending on how I is given.[9] However, this implies that $w = f_c \cdot 2^e$ is an integer, but since $w \in I$, we get that $I \cap \mathbb{Z} \ne \emptyset$. This is absurd, because I is an open interval between two consecutive integers. □

It should be noted that the shortest decimal numbers in I are the elements of the intersection $I \cap 10^{-k}\mathbb{Z}$ where k is the smallest integer making the intersection nonempty. Although this sounds obvious, let us formally prove it. First, we define the number of decimal significand digits of a nonzero real number v as $\left\lfloor \log_{10}(v \cdot 10^k) \right\rfloor + 1$ where k is the smallest integer such that $v \cdot 10^k \in \mathbb{Z}$. For example,

- If $v = 1.23$, then $k = 2$ and $\left\lfloor \log_{10}(v \cdot 10^k) \right\rfloor + 1 = 3$,
- If $v = 0.01234$, then $k = 5$ and $\left\lfloor \log_{10}(v \cdot 10^k) \right\rfloor + 1 = 5$, and
- If $v = 1200$, then $k = -2$ and $\left\lfloor \log_{10}(v \cdot 10^k) \right\rfloor + 1 = 2$.

Proposition 3.2.
The set $I \cap 10^{-k}\mathbb{Z}$, *where* k *is the smallest integer making the intersection nonempty, is precisely the set of elements in* I *with the smallest number of decimal significand digits..*

Proof. By the assumption on k, we know that $I \cap 10^{-k}\mathbb{Z}$ is not empty while $I \cap 10^{-k+1}\mathbb{Z}$ is empty. Equivalently, $10^k I \cap \mathbb{Z}$ is not empty while $10^{k-1} I \cap \mathbb{Z}$ is empty. Since I is an interval, $10^k I \cap \mathbb{Z} = \{m, m+1, \cdots, M-1, M\}$ for some integers $m, M \in \mathbb{Z}$. Since $10^{k-1} I \cap \mathbb{Z}$ is empty, there is no multiple of 10 among $m, \cdots, M$. Hence, we get $\lfloor \log_{10} m \rfloor = \lfloor \log_{10} M \rfloor$; otherwise, we have

$$\begin{aligned} \log_{10} m &< \lfloor \log_{10} m \rfloor + 1 \\ &\le \lfloor \log_{10} M \rfloor \le \log_{10} M, \end{aligned}$$

thus

$$m < 10^{\lfloor \log_{10} m \rfloor + 1} \le M,$$

which contradicts to that there is no multiple of 10 among $m, \cdots, M$. Note that for any v in the set

$$I \cap 10^{-k}\mathbb{Z} = \left\{10^{-k} m, \cdots, 10^{-k} M\right\},$$

k is the smallest integer such that $v \cdot 10^k$ is an integer, thus all such v have $\lfloor \log_{10} m \rfloor + 1$ decimal significand digits.

[7] One might regard this propisition as a form of the pigeonhole principle. In fact, the name *Schubfach* is coming from the German name of the pigeonhole principle, *Schubfachprinzip*, meaning "drawer principle".

[8] In fact, we show in the proof that for any $v \in I$, at least one of $\left\lfloor 10^k v \right\rfloor$ and $\left\lfloor 10^k v \right\rfloor + 1$ should be in $10^k I$.

[9] In fact, since I is an open interval, the first case is impossible, so we have $e = 0$.

Now, let us show that $\lfloor \log_{10} m \rfloor + 1$ is the minimum possible number of decimal significand digits. To show that, we first claim that

$$\lfloor \log_{10}(m-1) \rfloor = \lfloor \log_{10} m \rfloor$$

if $m \neq 1$. Indeed, if not, then we have

$$\begin{aligned} \log_{10}(m-1) &< \lfloor \log_{10}(m-1) \rfloor + 1 \\ &\leq \lfloor \log_{10} m \rfloor \leq \log_{10} m, \end{aligned}$$

thus

$$m - 1 < 10^{\lfloor \log_{10}(m-1) \rfloor + 1} \leq m.$$

Since $10^{\lfloor \log_{10}(m-1) \rfloor} + 1$ is an integer, we must have $m = 10^{\lfloor \log_{10}(m-1) \rfloor + 1}$, which contradicts to that m is not a multiple of 10. This shows the claim.

Next, note that for any $v \in I$ such that there exists $l \in \mathbb{Z}$ with $v \cdot 10^l \in \mathbb{Z}$, we have $l \geq k$ because of how we chose k. If $l = k$, then $v \cdot 10^l$ is one of $m, \cdots, M$, so we may assume $l > k$. Note also that we may assume $m \neq 1$, because if $m = 1$ then the number of decimal significand digits of elements in $I \cap 10^{-k}\mathbb{Z}$ is 1, which is of course a lower bound on the number of decimal significand digits of v. Now, since we have

$$\begin{aligned} \left\lfloor \log_{10}(v \cdot 10^l) \right\rfloor &= \left\lfloor \log_{10}(v \cdot 10^k) \right\rfloor + (l-k) \\ &\geq \left\lfloor \log_{10}(v \cdot 10^k) \right\rfloor + 1, \end{aligned}$$

it suffices to show that $\left\lfloor \log_{10}(v \cdot 10^k) \right\rfloor \geq \lfloor \log_{10} m \rfloor$. This inequality actually follows directly from our previous claim $\lfloor \log_{10}(m-1) \rfloor = \lfloor \log_{10} m \rfloor$; indeed, as $10^{-k}(m-1)$ is not an element of I, we should have $v > 10^{-k}(m-1)$, or equivalently, $v \cdot 10^k > m - 1$, which implies

$$\left\lfloor \log_{10}(v \cdot 10^k) \right\rfloor \geq \lfloor \log_{10}(m-1) \rfloor = \lfloor \log_{10} m \rfloor .$$

□

Since we have the following *chain property*

$$I \cap 10^{-k+1}\mathbb{Z} \subseteq I \cap 10^{-k}\mathbb{Z}$$

for all $k \in \mathbb{Z}$, we get the following:

Corollary 3.3.
Let $k_0 \coloneqq -\lfloor \log_{10} \Delta \rfloor$. Then:

1. *If $I \cap 10^{-k_0+1}\mathbb{Z}$ is not empty, then the unique element in it has the smallest number of decimal significand digits in I.*
2. *Otherwise, elements in $I \cap 10^{-k_0}\mathbb{Z}$ have the smallest number of decimal significand digits.*

Proof. Suppose first that $I \cap 10^{-k_0+1}\mathbb{Z}$ is not empty. Let $l \in \mathbb{Z}$ be the smallest integer such that $I \cap 10^{-l}\mathbb{Z}$ is not empty, then by the chain property, we know

$$\emptyset \neq I \cap 10^{-l}\mathbb{Z} \subseteq I \cap 10^{-k_0+1}\mathbb{Z},$$

but since $I \cap 10^{-k_0+1}\mathbb{Z}$ can have at most 1 element by Proposition 3.1, it follows that the unique element of $I \cap 10^{-k_0+1}\mathbb{Z}$ is the unique element of $I \cap 10^{-l}\mathbb{Z}$. Hence, that unique element has the smallest number of decimal significand digits in I by Proposition 3.2.

Next, suppose that $I \cap 10^{-k_0+1}\mathbb{Z} = \emptyset$. Then again by the chain property, k_0 must be the smallest integer such that $I \cap 10^{-k_0}\mathbb{Z}$ is not empty, so the result follows from Proposition 3.2. □

Note that, since we always have $w \in I$, so if $I \cap 10^{-k}\mathbb{Z}$ is nonempty for some $k \in \mathbb{Z}$, then at least one of $\lfloor w \cdot 10^k \rfloor 10^{-k}$ and $\left(\lfloor w \cdot 10^k \rfloor + 1\right) 10^{-k}$ must be in $I \cap 10^{-k}\mathbb{Z}$. More precisely, pick any $v \in I \cap 10^{-k}\mathbb{Z}$, then if $v \leq w$, then $\lfloor w \cdot 10^k \rfloor 10^{-k}$ is in $I \cap 10^{-k}\mathbb{Z}$ since $\lfloor w \cdot 10^k \rfloor$ is the largest integer smaller than or equal to $w \cdot 10^k$, so it should lie in between $v \cdot 10^k$ and $w \cdot 10^k$. Similarly, if $v > w$, then $\left(\lfloor w \cdot 10^k \rfloor + 1\right) 10^{-k}$ is in $I \cap 10^{-k}\mathbb{Z}$ since $\lfloor w \cdot 10^k \rfloor + 1$ is the smallest integer strictly greater than $w \cdot 10^k$, so it should lie in between $w \cdot 10^k$ and $v \cdot 10^k$. This leads us to the following strategy of finding the shortest decimal representation of w, which is the basic skeleton of Schubfach:

Algorithm 3.4 (Skeleton of Schubfach)**.**

1. Compute $k_0 \coloneqq -\lfloor \log_{10} \Delta \rfloor$.
2. Compute $\lfloor w \cdot 10^{k_0-1} \rfloor$ and $\lfloor w \cdot 10^{k_0-1} \rfloor + 1$. If one of them (and only one of them) belongs to $I \cdot 10^{k_0-1}$, then call that number s. In this case, $s \cdot 10^{-k_0+1}$ is the unique number in I with the smallest number of decimal significand digits. However, s might contain trailing decimal zeros; that is, it might be a multiple of a power of 10 as $I \cdot 10^{-l}\mathbb{Z}$ might not be empty for some $l < k_0 - 1$. Thus, let d be the greatest integer such that 10^d divides s, then $\frac{s}{10^d} \times 10^{d-k_0+1}$ is the unique shortest decimal representation of w.
3. Otherwise, we compute $\lfloor w \cdot 10^{k_0} \rfloor$ and $\lfloor w \cdot 10^{k_0} \rfloor + 1$. Then at least one of them must be in $I \cdot 10^{k_0}$, and if only one of them is inside I, call that number s. In this case, $s \cdot 10^{-k_0}$ is the uniuqe number in I with the smallest number of decimal significand digits. Since we assumed that $I \cap 10^{-k_0+1}\mathbb{Z}$ is empty, s is never divisible by 10 so there is no trailing decimal zeros and $s \times 10^{-k_0}$ is the unique shortest decimal representation of w.
4. If both $\lfloor w \cdot 10^{k_0} \rfloor$ and $\lfloor w \cdot 10^{k_0} \rfloor + 1$ are inside $I \cdot 10^{k_0}$, choose the one that is closer to $w \cdot 10^{k_0}$. When the distances from $w \cdot 10^{k_0}$ to those numbers are the same, break the tie according to a given rule.[10] Call the chosen number s, then again s cannot have any trailing decimal zeros and $s \times 10^{-k_0}$ is the correctly rounded shortest decimal representation of w.

[10] The most common rule is to choose the even one, but we can consider other rules as well.

Based on the above strategy, the details of Schubfach include following:

- How to efficiently compute $\lfloor \log_{10} \Delta \rfloor$?
- How to efficiently compute $\lfloor w \cdot 10^{k_0-1} \rfloor$, $\lfloor w \cdot 10^{k_0-1} \rfloor + 1$, $\lfloor w \cdot 10^{k_0} \rfloor$ and $\lfloor w \cdot 10^{k_0} \rfloor + 1$?
- How to efficiently compare these numbers to the endpoints of $I \cdot 10^{k_0-1}$ or $I \cdot 10^{k_0}$?

Similar to Ryū and Grisu-Exact, Schubfach uses a table of precomputed binary digits of powers of 10 in order to accomplish the second item. In addition to that, it uses an ingenious rounding trick to make the third item trivial.[11] More precisely, after computing k_0, Schubfach computes approximations of $w_L \cdot 10^{k_0}$ and $w_R \cdot 10^{k_0}$ along with that of $w \cdot 10^{k_0}$, with the aforementioned rounding rule applied, and the construction of the rounding rule ensures that we can just compare our number to the computed approximations of $w_L \cdot 10^{k_0}$ and $w_R \cdot 10^{k_0}$ in order to deduce if our number is in the interval or not.

However, even with the precomputed cache, computing the approximate multiplications $w_L \times 10^{k_0}$, $w_R \times 10^{k_0}$, and $w \times 10^{k_0}$, is not cheap, because it requires several 64-bit multiplications, which, for typical modern x86 machines, are a lot slower than many other instructions. (We will review how these approximate multiplications can be done in Section 4.2.) The core idea of Dragonbox is, thus, on how we can avoid these multiplications.

4. Dragonbox

For this section, we will assume a round-to-nearest rounding rule, which is the most relevant and at the same time the most difficult case. Algorithms for other rounding rules can be developed in similar ways, and they will be covered in Appendix A and Appendix B.

4.1 Overview

We will describe the flow of Dragonbox for the case when $F_w \neq 1$ or $E_w = E_{\min}$ (we call this *normal interval case*), so that $\Delta = 2^{E_w - p}$. The case $F_w = 1$ and $E_w \neq E_{\min}$ (we call this *shorter interval case*) will be covered in Section 5

Not like Schubfach, consider the following exponent instead of $k_0 := -\lfloor \log_{10} \Delta \rfloor$:

$$k := k_0 + \kappa = -\lfloor \log_{10} \Delta \rfloor + \kappa,$$

where κ is a positive integer constant in a certain range that we will discuss in Section 4.5. We will also discuss on how to compute k efficiently in that section.

[11] To be honest, I did not look at this rounding trick carefully, and do not fully understand how it works. Dragonbox does not rely on this trick, so it should be irrelevant for the rest of the paper. However, it might be that we can still possibly apply the trick also to Dragonbox so that we can make it even faster.

Similarly to [4], let us use the following notations:

$$\begin{aligned} x &:= w_L \cdot 10^k, \\ y &:= w \cdot 10^k, \\ z &:= w_R \cdot 10^k, \\ \delta &:= z - x = \Delta \cdot 10^k, \end{aligned}$$

and for $a \in \mathbb{R}$, we denote $a^{(i)} := \lfloor a \rfloor$, $a^{(f)} := a - \lfloor a \rfloor$.

Using a Grisu-like idea based on the following simple fact, we can mostly avoid computing x and y when doing the second step of Algorithm 3.4:

Proposition 4.1.
Let s, r be the unique integers satisfying

$$z = 10^{\kappa+1} s + r, \quad 0 \leq r < 10^{\kappa+1}.$$

Then, $I \cap 10^{-k_0+1}\mathbb{Z}$ is nonempty if and only if

$$s \in 10^{k_0-1} I,$$

if and only if:

1. $r + z^{(f)} \leq \delta$, *when* $I = [w_L, w_R]$,
2. $r + z^{(f)} < \delta$, *when* $I = (w_L, w_R]$.
3. $r + z^{(f)} \leq \delta$ *and* $r \neq 0$ *or* $z^{(f)} \neq 0$, *when* $I = [w_L, w_R)$, *and*
4. $r + z^{(f)} < \delta$ *and* $r \neq 0$ *or* $z^{(f)} \neq 0$, *when* $I = (w_L, w_R)$.

Proof. We first show that $I \cap 10^{-k_0+1}\mathbb{Z}$ is nonempty if and only if $s \in 10^{k_0-1} I$. Clearly, $s_\kappa \cdot 10^{-k_0+1}$ is always an element of $10^{-k_0+1}\mathbb{Z}$, so if it belongs to I, then $I \cap 10^{-k_0+1}\mathbb{Z}$ is nonempty.

Conversely, suppose $I \cap 10^{-k_0+1}\mathbb{Z}$ is nonempty. Let v be any element of it. Then, $v \leq w_R$, so

$$10^{k-\kappa-1} v \leq \frac{z}{10^{\kappa+1}},$$

but since $10^{k-\kappa-1} v = 10^{k_0-1} v \in \mathbb{Z}$, it follows that

$$10^{k-\kappa-1} v \leq \left\lfloor \frac{z}{10^{\kappa+1}} \right\rfloor = s.$$

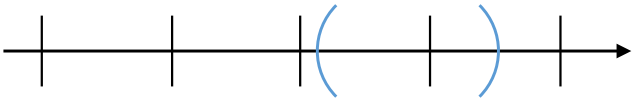

Figure 3. The unique lattice point in I should be the floor of the right endpoint, since I is longer than the unit

Now, since $10^{k_0-1} v$ and s are both integers, if we suppose

$$10^{k_0-1} v \neq s,$$

then

$$10^{k_0-1} v + 1 \leq s$$

follows, which implies

$$10^{-k_0+1}s \geq v + 10^{-k_0+1} > v + \Delta \geq w_L + \Delta = w_R$$

by definition of k_0. This is absurd, because

$$10^{-k_0+1}s = 10^{-k} \cdot 10^{\kappa+1}s \leq 10^{-k} \cdot z = w_R.$$

Hence, we deduce $s = 10^{k_0-1}v \in 10^{k_0-1}I$, concluding the first "if and only if".

To show the second "if and only if", let us recall that $10^{-k_0+1}s = 10^{-k} \cdot 10^{\kappa+1}s$ is at most w_R. Hence, when $w_R \in I$, $10^{-k_0+1}s$ is in I if and only if its distance from w_L is less than or equal to Δ, or strictly less than Δ, depending on whether or not if w_L is in I, which are precisely the claims *1* and *2*.

On the other hand, if $w_R \notin I$, then we need to rule out the case $w_R = 10^{-k_0+1}s_\kappa$ in addition, which is precisely the case when $r_\kappa = 0$ and $z^{(f)} = 0$, thus we have the last two claims as well. □

Note that $r + z^{(f)} \leq \delta$ if and only if

1. $r < \delta^{(i)}$, or
2. $r = \delta^{(i)}$ and $z^{(f)} \leq \delta^{(f)}$,

and we have a similar equivalence for $r + z^{(f)} < \delta$. As in [4], we can efficiently perform these comparisons.

More precisely,

1. We first compute $z^{(i)}$ by computing the high q-bits of a Q-bits × q-bits multiplication, where Q is the precision of powers of 10 cache entries which is usually chosen to be $Q = 2q$. Details will be explained in Section 4.2.
2. Then, given that κ is a compile-time constant, computation of s and r can be done efficiently without actually issuing a notoriously slow integer division instruction, as described in [8]. Details will be explained in Section 4.7.
3. Computing $\delta^{(i)}$ can be done in a similar manner as for $z^{(i)}$, but it can be also done in a simpler way, without actually performing a single multiplication. When $\Delta = 3 \cdot 2^{e-2}$, this requires some care, but we already have assumed that it is not the case, so to compute $\Delta = 2^e$ times 10^k, we just need to shift a cache entry by a specific amount.[12] Details will be explained in Section 4.4.
4. As explained in [4], inspecting $z^{(f)} < \delta^{(f)}$ can be done by computing the parity of $x^{(i)}$, which can be computed with a similar way as for $z^{(i)}$. However, since we are only interested in the parity rather than the complete value of $x^{(i)}$, this computation can be done a little bit more efficiently. Details will be explained in Section 4.3.
5. As explained in [4], inspecting $z^{(f)} = \delta^{(f)}$ can be done by inspecting if x is an integer, which can be done efficiently. Details will be explaiend in Section 4.6.
6. When $r = 0$, we also need to inspect if $z^{(f)} = 0$, or equivalently, if z is an integer. Again this can be done efficiently, and details will be explained in Section 4.6.

Note that in order to compare $z^{(f)}$ and $\delta^{(f)}$, we need to compute (the parity of) $x^{(i)}$ which is what we want to avoid. Hence, we want to minimize the chance of having $r = \delta^{(i)}$. Thus, we want to choose κ as large as possible. However, choosing too big κ will prevent us from computing $z^{(i)}$ and $\delta^{(i)}$ efficiently, so there are in fact not so many choices for κ we have. Details will be explained in Section 4.5.

So far, we have seen how we can inspect if $I \cap 10^{-k_0+1}\mathbb{Z}$ is empty or not. If it is not empty, then the unique element in it is

$$s \cdot 10^{-k_0+1} = s \cdot 10^{-k+\kappa+1}.$$

Then we find the greatest integer d such that 10^d divides s, and conclude that

$$\frac{s}{10^d} \times 10^{-k+\kappa+1+d}$$

is the unique shortest decimal representation of w.

Next, let us discuss what should we do if $I \cap 10^{-k_0+1}\mathbb{Z}$ turns out to be empty. What we will do in this case is a bit different from the Schubfach's way. Recall that Corollary 3.3 tells us that in this case,

$$I \cap 10^{-k_0}\mathbb{Z} = 10^{-k}\left(10^k I \cap 10^\kappa \mathbb{Z}\right)$$

is not empty and its elements have the smallest number of significand digits.

We will now compute

$$y^{(ru)} := \left\lfloor \frac{y}{10^\kappa} + \frac{1}{2} \right\rfloor 10^\kappa \quad \text{and}$$
$$y^{(rd)} := \left\lceil \frac{y}{10^\kappa} - \frac{1}{2} \right\rceil 10^\kappa,$$

which are the elements in $10^\kappa\mathbb{Z}$ that are closest to $y \in 10^k I$, using a method similar to that described in [4]. As shown in [4], both of $y^{(ru)}$ and $y^{(rd)}$ should be in $10^k I$ because we assumed that $F_w \neq 1$ or $E_w = E_{\min}$; we will revisit this and explain it in detail in Section 4.9.

Note that $y^{(ru)} = y^{(rd)} + 1$ if

$$\frac{y}{10^\kappa} - \left\lfloor \frac{y}{10^\kappa} \right\rfloor = \frac{1}{2},$$

and $y^{(ru)} = y^{(rd)}$ otherwise. In other words, $y^{(ru)}$ and $y^{(rd)}$ are same except when there is a tie, so we just need to focus on computing $y^{(ru)}$, detect the tie, and decrease the computed value of $y^{(ru)}$ by one if we prefer to choose $y^{(rd)}$ according to the given rule to break the tie. Let $y^{(r)}$ be the chosen one when we had a tie, or otherwise the common value of $y^{(ru)} = y^{(rd)}$, then the correctly rounded decimal representation of w with the shortest number of digits is thus

$$y^{(r)} \times 10^{-k+\kappa}.$$

[12] In fact, even for the case $\Delta = 3 \cdot 2^{e-2}$, we can compute $\delta^{(i)}$ in a similar way without performing a multiplication, as explained in [4].

To actually compute $y^{(ru)}$, note that

$$\begin{aligned} y^{(ru)} &= \left\lfloor \frac{y + (10^\kappa/2)}{10^\kappa} \right\rfloor \\ &= \left\lfloor \frac{z + (10^\kappa/2) - (z - y)}{10^\kappa} \right\rfloor \\ &= 10s + \left\lfloor \frac{r + (10^\kappa/2) - \epsilon^{(i)} + (z^{(f)} - \epsilon^{(f)})}{10^\kappa} \right\rfloor \end{aligned}$$

where we define

$$\epsilon \coloneqq z - y.$$

Since we have assumed $F_w \neq 1$ or $E_w = E_{\min}$, w should lie at the exact center of I. Hence in particular, $\epsilon = \frac{\delta}{2}$, so $\epsilon^{(i)} = \left\lfloor \frac{\delta^{(i)}}{2} \right\rfloor$. Also, since κ is a positive integer, $10^\kappa/2$ is an integer. Thus, let us write

$$D \coloneqq r + (10^\kappa/2) - \epsilon^{(i)} = 10^\kappa t + \rho$$

for some integers t and $0 \le \rho < 10^\kappa$, then

$$y^{(ru)} = (10s + t) + \left\lfloor \frac{\rho + (z^{(f)} - \epsilon^{(f)})}{10^\kappa} \right\rfloor.$$

Note that the residue term

$$\left\lfloor \frac{\rho + (z^{(f)} - \epsilon^{(f)})}{10^\kappa} \right\rfloor$$

is always 0 except when $\rho = 0$ and $z^{(f)} < \epsilon^{(f)}$, and for that case the above residue term is equal to -1. Hence, we can just ignore the fractional parts and just conclude $y^{(ru)} = 10s + t$ when D is not divisible by 10^κ, which is usually the case especially when κ is large. Of course when D is divisible by 10^κ, we need to compare $z^{(f)}$ and $\epsilon^{(f)}$ but this can be done by computing the parity of $y^{(i)}$ just like the comparison of $z^{(f)}$ and $\delta^{(f)}$. Then we can determine if we should decrease $10s+t$ by 1 or not. Details will be explained in Section 4.3.

Note that tie happens exactly when $\rho = z^{(f)} - \epsilon^{(f)} = 0$; indeed, tie happens when the fractional part of $\frac{y}{10^\kappa}$ is exactly $1/2$, or equivalently,

$$\frac{y}{10^\kappa} + \frac{1}{2} = (10s + t) + \frac{\rho + (z^{(f)} - \epsilon^{(f)})}{10^\kappa}$$

is an integer. Since

$$-1 < \rho + (z^{(f)} - \epsilon^{(f)}) < 10^\kappa,$$

it follows that $\frac{y}{10^\kappa} + \frac{1}{2}$ is an integer if and only if

$$\rho + (z^{(f)} - \epsilon^{(f)}) = 0,$$

if and only if $\rho = z^{(f)} - \epsilon^{(f)} = 0$. Or equivalently, tie happens if and only if D is divisible by 10^κ and $y = z - \epsilon$ is an integer. Details will be explained in Section 4.6.

In summary, when $I \cap 10^{-k_0+1}\mathbb{Z}$ turns out to be empty, then:

1. Compute $D = r + (10^\kappa/2) - \left\lfloor \delta^{(i)}/2 \right\rfloor$.
2. Compute t, ρ by dividing D by 10^κ. Again, given that κ is a compile-time constant, this can be done efficiently using the method described in [8]. In fact, since we do not care about the actual value of ρ and we only need to know if ρ is zero or not, we can do even better. Details will be explained in Section 4.8.
3. If $\rho \neq 0$, then $(10s + t) \times 10^{-k+\kappa}$ is the answer we are looking for.
4. Otherwise, compute the parity of $y^{(i)}$ and from that, inspect if $z^{(f)} < \epsilon^{(f)}$ holds. If the inequality holds, then $(10s + t - 1) \times 10^{-k+\kappa}$ is the answer we are looking for.
5. Otherwise, check if y is an integer. If that is the case, then we have a tie; break it according to a given rule, so that we choose one of $(10s + t - 1) \times 10^{-k+\kappa}$ and $(10s + t) \times 10^{-k+\kappa}$ as the answer.
6. Otherwise, $(10s + t) \times 10^{-k+\kappa}$ is the answer we are looking for.

Again, we want to avoid computing (the parity of) $y^{(i)}$, so we prefer to choose κ as big as possible.

References

[1] R. Giulietti. The Schubfach Way to Render Doubles. *2020* https://drive.google.com/file/d/1KLtG_LaIbK9ETXI290zqCxvBW94dj058/view (Sep. 2020)

[2] F. Loitsch. Printing Floating-Point Numbers Quickly and Accurately with Integers. In *Proceedings of the ACM SIGPLAN 2010 Conference on Programming Language Design and Implementation, PLDI 2010*. ACM, New York, NY, USA, 233–243. https://doi.org/10.1145/1806596.1806623

[3] M. Andrysco, R. Jhala, and S. Lerner. Printing Floating-Point Numbers: a Faster, Always Correct Method In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016*. ACM, New York, NY, USA, 555–567. https://doi.org/10.1145/2837614.2837654

[4] J. Jeon Grisu-Exact: A Fast and Exact Floating-Point Printing Algorithm *2020* https://github.com/jk-jeon/Grisu-Exact/blob/master/other_files/Grisu-Exact.pdf. (Sep. 2020)

[5] U. Adams. Ryū: Fast Float-to-String Conversion In *Proceedings of the ACM SIGPLAN 2018 Conference on Programming Language Design and Implementation, PLDI 2018*. ACM, New York, NY, USA, 270–282. https://doi.org/10.1145/3296979.3192369

[6] G. L. Steel Jr. and J. L. White. How to Print Floating-Point Numbers Accurately. In *Proceedings of the ACM SIGPLAN 1990 Conference on Programming Language Design and Implementation, PLDI 1990*. ACM, New York, NY, USA, 112–126. https://doi.org/10.1145/93542.93559

[7] https://github.com/abolz/Drachennest. (Sep. 2020)

[8] T. Granlund and P. L. Montgomery. Division by Invariant Integers using Multiplication. In *ACM SIGPLAN Notices, Vol 29, Issue 6, Jun. 1994*. ACM, New York, NY, USA, 61–72. https://doi.org/10.1145/773473.178249

[9] https://github.com/jk-jeon/Grisu-Exact. (Sep. 2020)

[10] https://stackoverflow.com/questions/25095741/how-can-i-multiply-64-bit-operands-and-get-128-bit-result-portably. (Jun. 2020)

[11] https://github.com/ulfjack/ryu. (Jun. 2020)