Towards a correctly-rounded and fast power function in binary64 arithmetic

Tom Hubrecht, Claude-Pierre Jeannerod, Paul Zimmermann, Laurence Rideau, Laurent Théry

Towards a correctly-rounded and fast power function in binary64 arithmetic

Tom Hubrecht
Département d'Informatique de l'ENS
École Normale Supérieure,
CNRS, PSL University
F-75005 Paris, France
tom.hubrecht@ens.fr

Claude-Pierre Jeannerod
Inria, Université de Lyon
CNRS, ENSL, UCBL, LIP
F-69342 Lyon, France
claude-pierre.jeannerod@inria.fr

Paul Zimmermann
Université de Lorraine
CNRS, Inria, LORIA
F-54000 Nancy, France
paul.zimmermann@inria.fr

Laurence Rideau
Université Côte d'Azur, Inria
F-06902 Sophia-Antipolis, France
laurence.rideau@inria.fr

Laurent Théry
Université Côte d'Azur, Inria
F-06902 Sophia-Antipolis, France
laurent.thery@inria.fr

***Abstract*—We design algorithms for the correct rounding of the power function x^y in the binary64 IEEE 754 format, for all rounding modes, modulo the knowledge of hardest-to-round cases. Our implementation of these algorithms largely outperforms previous correctly-rounded implementations and is not far from the efficiency of current mathematical libraries, which are not correctly-rounded. Still, we expect our algorithms can be further improved for speed. The proofs of correctness are fully detailed and have been formally verified. We hope this work will motivate the next IEEE 754 revision committee to require correct rounding for mathematical functions.**

***Index Terms*—IEEE 754, double precision, binary64 format, power function, correct rounding, efficiency.**

I. Introduction

The IEEE standard for floating-point arithmetic requires correct rounding for basic arithmetic operations since 1985, but in its latest revision (IEEE 754-2019) [11], it still does not require correct rounding for the most common mathematical functions such as $\log$, $\exp$, $\sin$, $\cos$, etc.

Among all these commonly used functions, one of the trickiest ones to implement is the power function $(x, y) \mapsto x^y$. One difficulty is that it is a bivariate function, thus it has many more possible inputs for a given IEEE format, and in particular its hardest-to-round cases are not known yet for the binary64 format (formerly called 'double precision' and where the precision is 53). Another difficulty of the power function is that it has many exact and midpoint cases (that is, cases where x^y is exactly representable in precision 54). Finally, as the relation $x^y = b^{y \log_b x}$ for $b > 1$ and $x > 0$ shows, the power function grows very fast in some regions. This requires that some intermediate steps be performed with a significant amount of extra precision which however must be kept small enough in order to achieve high efficiency in practice.

Current mathematical libraries do not provide a correctly rounded power function for the binary64 format. For this format, the maximal known error goes from 0.523 ulp for the GNU libc to 636 ulps for OpenLibm [12].

A. State of the Art

Detailed descriptions of how to implement the power function are given in Markstein's book [19, Chapter 12] and Beebe's book [1, Chapter 14]. Markstein suggests to use the formula $x^y = 2^{y \log_2 x}$, a choice already advocated in [5]. We tried this approach first as it looked quite promising for binary arithmetic, but in our context it turned out to be less efficient than using the relation $x^y = e^{y \log x}$, essentially since the Taylor approximations of $\log x$ and e^x have simpler coefficients. Beebe also suggests evaluating x^y by means of $2^{y \log_2 x}$ for binary arithmetic. His algorithms are more complex, in particular since the fused multiply-add (FMA) instruction is not used for producing exact products. Finally, none of these two chapters discusses ways to ensure correct rounding.

The MathLib library [25] includes a correctly rounded power function for the binary64 format. However, the algorithm used is not fully detailed, and MathLib is no longer maintained. Some MathLib routines (including the power function) were included in the GNU libc up to release 2.27, but the "slow path" was gradually removed, and the latest release of the GNU libc does no longer provide a correctly rounded power [12]. Furthermore, MathLib only provides correct rounding for rounding to nearest-even.

Sun Microsystems developed a library called LIBMCR that included in its latest version (release 0.9, April 2004) seven functions with claimed correct rounding (presumably for rounding to nearest-even):

exp, log, pow, atan, sin, cos, tan. LIBMCR still compiles with modern compilers but has both efficiency and correctness issues (see §VI).

CRLIBM [6] also provides a correctly rounded power function. Like for MathLib, this function is only implemented for rounding to nearest-even. Also, this function has remained tagged as "experimental" and has indeed some correctness issues (see §VI). Compared to MathLib, though, CRLIBM has a better treatment of exact cases and midpoint cases, based on works by Lauter and Lefèvre [14], [15].

More recently, the RLIBM project [17], the LLVM C Library [18], and the CORE-MATH project [20] have proposed implementations for various correctly-rounded functions. However, at the time of writing, RLIBM only provides binary32 functions, LLVM libc only provides the `hypot` and `log10` functions in binary64, and CORE-MATH provides about 20 binary64 functions, but not x^y.

B. Our Contributions

In this paper we present a new algorithm for the binary64 power function with correct rounding (modulo the knowledge of hardest-to-round cases) together with a very efficient implementation. Compared to previous work (MathLib, LIBMCR, and CRLIBM), this algorithm is not restricted to rounding to nearest-even, but also works for the directed rounding modes from IEEE 754.

Furthermore (and as far as we can tell), this is the first time an algorithm for binary64 powering with correct rounding is fully detailed and presented together with its rounding error analysis. The algorithm, which makes heavy use of the FMA instruction available on most modern processors, can be readily implemented.

Finally, our implementation of the algorithm yields a two-fold to five-fold speedup with respect to MathLib, CRLIBM, and LIBMCR.

C. Outline

The paper is organized as follows. Section II gives a high-level description of our three-phase algorithm for computing correctly-rounded powers for the IEEE binary64 format, together with some notation to be used later on. Section III then provides a detailed description of all the building blocks and algorithms used in the first phase of the powering algorithm, together with explicit error bounds (each of which having a detailed proof given in appendix). Section IV only provides a coarse description of the second and third phases, since those follow essentially the same approach as the first phase (up to higher precisions, larger approximation polynomials, and more sophisticated range reduction). Section V discusses the correctness of our implementation on known worst cases as well as on random inputs. Section VI compares our implementation to the MathLib, LIBMCR and CRLIBM libraries, and to the incorrectly-rounded GNU libc power function. We conclude and discuss further work in Section VIII.

II. High-level Algorithm

Before approximating x^y using $\exp(y \log x)$, the case $x \leq 0$ gets a special treatment according to the IEEE 754-2019 specification [11, §9]. The same is done for the cases where x or y is $\pm\infty$ or NaN. (See also [13, §F.10.4.5].) Thus, we assume from now on that both x and $|y|$ are in the range $[\alpha, \Omega]$, where $\alpha := 2^{-1074}$ denotes the smallest positive subnormal binary64 number, and $\Omega := (1 - 2^{-53}) \cdot 2^{1024}$ denotes the largest finite binary64 number. Then, clearly, $\log x$ is either zero or in a subrange of the range $[-\Omega, -2^{-1022}] \cup [2^{-1022}, \Omega]$ of normal binary64 numbers.

Our powering algorithm has three phases. A first phase approximates x^y using double-double arithmetic (§III) in a very efficient way; if the rounding test of the first phase fails, then a second phase approximates x^y using 128-bit arithmetic (§IV); if the rounding test of the second phase fails again, then a third phase approximates x^y using 256-bit arithmetic (§IV). Since worst cases for the binary64 power function are still unknown, the third phase contains a final rounding test. In the very unlikely event where this last test fails, an error message is printed, which guarantees that no incorrect rounding is produced.

The three phases use the same algorithm:

- first we compute an approximation of $\log x$, by first rewriting $x = 2^e \cdot t$ with $\sqrt{2}/2 < t < \sqrt{2}$. Then we use Tang's algorithm [22]: a table value r approximates $1/t$, and yields a small value $z \approx r \cdot t - 1$. Finally we get $\log x \approx e \log 2 - \log r + \log(1 + z)$, where $\log r$ is read from a table, and $\log(1 + z)$ is approximated using a small-degree polynomial;
- then we multiply the approximation of $\log x$ by y, yielding a value called r;
- finally we approximate $\exp(r)$ by writing $r = (k + i/2^n) \log 2 + z$, where k is an integer, $i/2^n$ is a small rational, and z is small. It yields $\exp(r) = 2^k \cdot 2^{i/2^n} \cdot \exp(z)$, where $2^{i/2^n}$ is read from a table, and $\exp(z)$ is approximated using a small-degree polynomial.

A. Notation and basic properties

We write $\mathbb{Z}$, $\mathbb{N}$, $\mathbb{F}$ to denote the set of integers, nonnegative integers, and finite binary64 numbers, respectively. We write $\mathrm{ulp}(x)$ to denote the unit in the last place of $x \in \mathbb{F}$, which for $x \neq 0$ gives the weight of the last bit of the significand of x. More generally, for any real number x such that $|x| \leq \Omega$, we define its ulp as $\mathrm{ulp}(x) := 2^{\max(-1022, \lfloor \log_2 |x| \rfloor) - 52}$, that is, $\max(\alpha, 2^{\lfloor \log_2 |x| \rfloor - 52})$, with the convention that $\mathrm{ulp}(0) = \alpha$.

We denote by $\circ$ the current rounding mode, which can be to nearest with ties to even, toward zero, toward $+\infty$, or toward $-\infty$. Our algorithm is division free and makes heavy use of operations of the form

$$t \leftarrow \circ(xy + z),$$

with a single rounding; in practice, such operations are performed efficiently by fused multiply-add (FMA) instructions. Unless stated explicitly, all variables such as x, y, z, t above represent finite binary64 numbers.

Our error bounds will be deduced in particular from the following accuracy properties of $\circ$: when $|xy + z| \leq \Omega$, the absolute error of t is bounded as

$$|t - (xy + z)| \leq \mathrm{ulp}(xy + z) \leq \mathrm{ulp}(t);$$

if in addition $|xy+z| \geq 2^{-1022}$ (normal range), then the relative error satisfies $|t/(xy+z)-1| \leq \mathrm{ulp}(xy+z)/|xy+z|$ and is thus always at most 2^{-52}. Finally, in some proofs we shall also exploit the fact that if $|xy + z| \leq 2^e$ for some integer $e \geq -1021$, then $|t - (xy + z)| \leq 0.5 \cdot \mathrm{ulp}(2^e)$ instead of $\leq \mathrm{ulp}(2^e)$.

III. First Phase

The first phase uses double-double arithmetic for efficiency, and delivers an approximation to x^y with about 63 correct bits in the general case (see Algorithm 9). The only special cases are those discussed earlier: the cases $y \in \{0, 1/2, 1, 2\}$ are only checked after the first phase, in order not to slow down the critical path.

In the first phase we make heavy use of the following ExactMul and FastTwoSum algorithms. In the absence of underflow and overflow, ExactMul rewrites the exact product of two finite binary64 numbers as a double-double value, that is, as the unevaluated sum of two finite binary64 numbers. The last instruction $\ell \leftarrow \circ(ab - h)$ of ExactMul is

Algorithm 1 (ExactMul)

Input: $a, b \in \mathbb{F}$
Output: h, ℓ such that $h + \ell = ab$
1: $h \leftarrow \circ(ab)$
2: $\ell \leftarrow \circ(ab - h)$

efficiently implemented using an FMA.

Lemma 0: Assuming no overflow occurs, the output h, ℓ of ExactMul is exact if $ab \in \alpha\mathbb{Z}$, and otherwise satisfies:

$$|h + \ell - ab| < \alpha.$$

Proof: If $ab = 0$, this is clear. We now assume $ab \neq 0$, and we give a general proof for a precision-p binary system, with subnormal numbers and smallest positive value α. We distinguish two cases: $|ab| \geq 2^{2p-1}\alpha$, and $|ab| < 2^{2p-1}\alpha$.

Case $|ab| \geq 2^{2p-1}\alpha$. Assume $ab > 0$ (the case $ab < 0$ is similar). Since the significands of a, b have at most p bits, their product has at most $2p$ bits: we can write $ab = c \cdot 2^e$ with $2^{2p-1} \leq c < 2^{2p}$. Moreover since $ab \geq 2^{2p-1}\alpha$ we have $c \cdot 2^e \geq 2^{2p-1}\alpha$ thus $2^e/\alpha \geq 2^{2p-1}/c > 1/2$; since $2^e/\alpha$ is a power of two it is at least 1 thus $2^e \geq \alpha$ and ab is an integer multiple of α. Now let $c = c_h 2^p + c_\ell$ with $2^{p-1} \leq c_h < 2^p$ and $0 \leq c_\ell < 2^p$. The rounding of c is either $c_h 2^p$ or $(c_h + 1)2^p$. Thus $h = c_h 2^{e+p}$ or $h = (c_h + 1)2^{e+p}$, and $ab - h = c_\ell 2^e$ or $ab - h = (c_\ell - 2^p)2^e$. Since $0 \leq c_\ell < 2^p$, both c_ℓ and $c_\ell - 2^p$ are exactly representable on p bits, thus since ab is an integer multiple of α, so is $ab - h$, and $\ell = \circ(ab - h) = ab - h$.

If $|ab| < 2^{2p-1}\alpha$, then $\mathrm{ulp}(h) \leq \mathrm{ulp}(2^{2p-1}\alpha) = 2^p\alpha$, thus if we define $\ell' = ab - h$, then $|\ell'| < \mathrm{ulp}(h) \leq 2^p\alpha$. If $ab \in \alpha\mathbb{Z}$, then $\ell' = ab - h \in \alpha\mathbb{Z}$, thus together with $|\ell'| < 2^p\alpha$, ℓ' is exactly representable on p bits, and the output of ExactMul is exact. Otherwise, the rounding error on $\ell = \circ(\ell')$ is less than $\mathrm{ulp}(\ell') = \alpha$ (since $|\ell'| < 2^p\alpha$). ∎

FastTwoSum rewrites the exact sum of two finite binary64 numbers as a double-double value whose first component equals the rounded sum and whose second component is (an approximation to) the associated rounding error.

Algorithm 2 (FastTwoSum)

Input: $a, b \in \mathbb{F}$ with $a = 0$ or $|a| \geq |b|$
Output: h, ℓ such that $h + \ell$ approximates $a + b$
1: $h \leftarrow \circ(a + b)$
2: $t \leftarrow \circ(h - a)$
3: $\ell \leftarrow \circ(b - t)$

In the absence of underflow and overflow, it is well known that for rounding to nearest FastTwoSum is an error-free transform, that is, $h+\ell = a+b$ exactly, while for directed roundings we have in general only an approximation: $h + \ell \approx a + b$. It is shown in [24, Theorem 1] that the corresponding approximation error satisfies $|h+\ell-(a+b)| \leq 2^{-105}|h|$ and that it is zero when the exponent difference between a and b does not exceed 53. (See also [10] for a weaker bound.) Also, it is shown in [8, Theorem 3] that the middle computation $t \leftarrow \circ(h - a)$ is always exact, whatever the rounding mode.

Besides FastTwoSum, we shall also use the following FastSum algorithm, which allows us to add an extra error term to the low order term produced by FastTwoSum.

Algorithm 3 (FastSum)

Input: $a, b_h, b_\ell \in \mathbb{F}$ with $a = 0$ or $|a| \geq |b_h|$
Output: h, ℓ such that $h+\ell$ approximates $a+b_h+b_\ell$
1: $h, t \leftarrow \mathrm{FastTwoSum}(a, b_h)$
2: $\ell \leftarrow \circ(t + b_\ell)$

$$
\begin{aligned}
P &= z - z^2/2 + \texttt{0x1.5555555555558p-2}\, z^3 \\
&- \texttt{0x1.0000000000003p-2}\, z^4 \\
&+ \texttt{0x1.999999981f535p-3}\, z^5 \\
&- \texttt{0x1.55555553d1eb4p-3}\, z^6 \\
&+ \texttt{0x1.2494526fd4a06p-3}\, z^7 \\
&- \texttt{0x1.0001f0c80e8cep-3}\, z^8
\end{aligned}
$$

Fig. 1. Degree-8 polynomial $P(z)$ generated by Sollya [4] for approximating $\log(1+z)$ when $|z| \le 33 \cdot 2^{-13}$, with absolute error at most $2^{-81.63}$ and relative error at most $2^{-72.423}$.

Lemma 1: In the absence of underflow and overflow, the pair (h, ℓ) computed by Algorithm FastSum satisfies

$$\big|h + \ell - (a + b_h + b_\ell)\big| \le 2^{-105}|h| + \mathrm{ulp}(\ell).$$

Proof: The error of FastSum is bounded by the sum of the error of the FastTwoSum call, plus the rounding error in $t + b_\ell$. The first one is bounded by $2^{-105}|h|$ from [24], and the second one by $\mathrm{ulp}(\ell)$. ■

Section III-A computes a double-double approximation $h+\ell$ of $\log x$, which is multiplied by y in Section III-B to obtain a double-double approximation r_h+r_ℓ of $y \log x$, and a double-double approximation of $\exp(r_h + r_\ell)$ is obtained in Section III-C. Finally Section III-D details how these three steps are used in the first phase.

A. Approximation of $\log x$

To approximate $\log x$ for $x \in [\alpha, \Omega]$ a binary64 number, after some argument reduction described below, we have to approximate $\log(1+z)$ for some binary64 number z such that $|z| \le 33 \cdot 2^{-13}$. For this task, we use Algorithm p_1 which computes a double-double approximation p_h+p_ℓ of $\log(1+z)-z$, using a degree-8 polynomial (Fig. 1).

Algorithm 4 Algorithm p_1

Input: $z \in \mathbb{F}$
Output: $p_h + p_\ell$ approximating $\log(1+z) - z$

$w_h, w_\ell \leftarrow \mathrm{ExactMul}(z, z)$
$t \leftarrow \circ(P_8 z + P_7)$
$u \leftarrow \circ(P_6 z + P_5)$
$v \leftarrow \circ(P_4 z + P_3)$
$u \leftarrow \circ(t w_h + u)$
$v \leftarrow \circ(u w_h + v)$
$u \leftarrow \circ(v w_h)$
$p_h \leftarrow -0.5 \cdot w_h$
$p_\ell \leftarrow \circ(uz - 0.5 \cdot w_\ell)$

Lemma 2: Given $|z| \le 33 \cdot 2^{-13}$ with z an integer multiple of 2^{-61}, the double-double approximation $p_h + p_\ell$ to $\log(1+z) - z$ returned by Algorithm p_1 satisfies

$$|p_h + p_\ell - (\log(1+z) - z)| < 2^{-75.492},$$

with $|p_h| < 2^{-16.9}$ and $|p_\ell| < 2^{-25.446}$. If $z \ne 0$, and assuming further $|z| < 32 \cdot 2^{-13}$, the relative error satisfies

$$\left|\frac{z + p_h + p_\ell}{\log(1+z)} - 1\right| < 2^{-67.441}.$$

Proof: See Appendix A-A, and refinement via $|\delta_0| + |\delta_6| \le 3.505\mathbf{u}$ for the relative error in the case $|z| < 32 \cdot 2^{-13}$. ■

The approximation of $\log x$ is done using Algorithm 5. First, $x \in \mathbb{F} \cap [\alpha, \Omega]$ is written $2^e \cdot t$ with $e \in \mathbb{Z}$ and $t \in \mathbb{F} \cap (1/\sqrt{2}, \sqrt{2})$. Then, following [16], [22], we reduce the range of t even further by computing $z = \circ(rt - 1)$, where r is a precomputed 9-bit approximation to $1/t$ obtained from $i = \lfloor 2^8 t \rfloor$, and denoted $\mathrm{INVERSE}_i$ below. (The value of r is explicitly defined in Appendix A-B; when $i \in \{255, 256\}$, we shall simply take $r = 1$ in order to avoid cancellation when approximating $\log z - \log r$.) Note that z can be produced efficiently using an FMA.

Lemma 3: For any $t \in \mathbb{F} \cap (1/\sqrt{2}, \sqrt{2})$, let $i = \lfloor 2^8 t \rfloor$ and $r = \mathrm{INVERSE}_i$. Then $z = \circ(rt - 1)$ is exact, $|z| \le 33 \cdot 2^{-13}$, and $z \in 2^{-61}\mathbb{Z}$.
Proof: See Appendix A-B. ■

In Lemma 3, it can be checked that the upper bound $33 \cdot 2^{-13}$ is optimal if we want $z = \circ(rt - 1)$ to be exact, when using a table indexed by $\lfloor 2^8 t \rfloor$.

In Algorithm log_1, the table value LOGINV_i is a double-double approximation $\ell_1 + \ell_2$ to $-\log r$, such that $\ell_1 \in \mathbb{F}$ is an integer multiple of 2^{-42} nearest $-\log r$, and $\ell_2 \in \mathbb{F}$ is nearest $(-\log r) - \ell_1$. Similarly, LOG2H + LOG2L is a double-double approximation to $\log 2$, to nearest and with LOG2H an integer multiple of 2^{-42}.

Algorithm 5 Algorithm log_1

Input: a binary64 value $x \in [\alpha, \Omega]$
Output: $h + \ell$ approximating $\log x$

write $x = t \cdot 2^e$ with $t \in (1/\sqrt{2}, \sqrt{2})$ and $e \in \mathbb{Z}$
$i \leftarrow \lfloor 2^8 t \rfloor$ ▷ i integer
$r \leftarrow \mathrm{INVERSE}_i, \quad \ell_1, \ell_2 \leftarrow \mathrm{LOGINV}_i$
$z \leftarrow \circ(rt - 1)$
$t_h \leftarrow \circ(e\,\mathrm{LOG2H} + \ell_1)$
$t_\ell \leftarrow \circ(e\,\mathrm{LOG2L} + \ell_2)$
$h, \ell \leftarrow \mathrm{FastSum}(t_h, z, t_\ell)$
$p_h, p_\ell \leftarrow \mathrm{p\_1}(z)$
$h, \ell \leftarrow \mathrm{FastSum}(h, p_h, \circ(\ell + p_\ell))$
if $e = 0$ **then** $h, \ell \leftarrow \mathrm{FastTwoSum}(h, \ell)$

Lemma 4: Given $x \in [\alpha, \Omega]$, Algorithm log_1 computes (h, ℓ) such that $|\ell| \le 2^{-23.89}|h|$, and

$$|h + \ell - \log x| \le \varepsilon_{\log} \cdot |\log x| \qquad (1)$$

with $\varepsilon_{\log} = 2^{-73.527}$ if $x \notin (1/\sqrt{2}, \sqrt{2})$, and $\varepsilon_{\log} = 2^{-67.0544}$ otherwise.
Proof: If $x = 1$, then it is easy to see that $e = 0$, $t = 1$, $i = 256$ thus $r = 1$ (remember for $i \in \{255, 256\}$

we use $r = 1$ to avoid cancellation), $z = 0$, so that Algorithm p_1 returns $p_h = p_\ell = 0$ and Algorithm log_1 returns $h = \ell = 0$. The proof is thus finished in this case.

Let us now assume $x \neq 1$. Firstly, from the proof of Lemma 3, the value z computed at line 4 fulfills the conditions of Lemma 2. The main analysis of Algorithm log_1 is performed in Appendix A-C, where we distinguish three cases:

1) case $e \neq 0$, that is, $x < \sqrt{2}/2$ or $\sqrt{2} < x$. This case is detailed in Appendix A-D;
2) case $e = 0$ and $i \neq \{255, 256\}$, that is, $\sqrt{2}/2 < x < 255/256$ or $257/256 \leq x < \sqrt{2}$. This case is detailed in Appendix A-E;
3) case $e = 0$ and $i \in \{255, 256\}$, that is, $255/256 \leq x < 257/256$. This case is detailed in Appendix A-F.

■

B. Multiplication by y

Once we have computed a double-double approximation $h + \ell$ to $\log x$, we multiply it by y in order to obtain an approximation to $y \log x$, as in Algorithm 6. As pointed out in [19, Chapter 12], what is important is to bound the *absolute* error in the approximation to $y \log x$.

Algorithm 6 Algorithm mul_1

Input: a double-double value $h + \ell$, and a double y
Output: $r_h + r_\ell$ approximating $y(h + \ell)$
1: $r_h, s \leftarrow \text{ExactMul}(y, h)$
2: $r_\ell \leftarrow \circ(y\ell + s)$

Lemma 5: If x, h, ℓ are as in Lemma 4 and if the exact product yh satisfies $2^{-969} \leq |yh| \leq 709.7827$, then Algorithm 6 computes (r_h, r_ℓ) such that $|r_h| \in [2^{-970}, 709.79]$, $|r_\ell| \leq 2^{-14.4187}$, $|r_\ell/r_h| \leq 2^{-23.8899}$, $|r_h + r_\ell| \leq 709.79$, and

$$|r_h + r_\ell - y \log x| \leq \varepsilon_{\text{mul}}$$

with $\varepsilon_{\text{mul}} = 2^{-63.799}$ if $x \notin (1/\sqrt{2}, \sqrt{2})$, and $\varepsilon_{\text{mul}} = 2^{-57.580}$ otherwise.

Proof: See Appendix A-G. ■

C. Final exponentiation

Finally we approximate $\exp(r_h + r_\ell)$. After some argument reduction described in Algorithm 8, we have to approximate $\exp(z)$ for z a binary64 value such that $|z| \leq 2^{-12.905}$. (This bound comes from the paragraph "About the value z" in Appendix A-I.) We use a degree-4 polynomial (Fig. 2), that is evaluated using Algorithm q_1.

Lemma 6: Given $z \in \mathbb{F}$ such that $|z| \leq 2^{-12.905}$, Algorithm q_1 returns (q_h, q_ℓ) such that

$$\left| \frac{q_h + q_\ell}{\exp(z)} - 1 \right| < 2^{-64.902632}$$

and $|q_\ell| \leq 2^{-51.999}$.

$$\begin{aligned} Q &= 1 + z + z^2/2 \\ &+ \texttt{0x1.5555555997996p-3}\, z^3 \\ &+ \texttt{0x1.5555555849d8dp-5}\, z^4 \end{aligned}$$

Fig. 2. Degree-4 polynomial $Q(z)$ generated by Sollya for approximating $\exp(z)$ when $|z| \leq 2^{-12.905}$, with absolute error at most $2^{-74.34}$.

Algorithm 7 Algorithm q_1

Input: $z \in \mathbb{F}$
Output: $q_h + q_\ell$ approximating $\exp(z)$
1: $q \leftarrow \circ(Q_4 z + Q_3)$
2: $q \leftarrow \circ(qz + Q_2)$
3: $h_0 \leftarrow \circ(qz + Q_1)$
4: $h_1, \ell_1 \leftarrow \text{ExactMul}(z, h_0)$
5: $q_h, q_\ell \leftarrow \text{FastSum}(Q_0, h_1, \ell_1)$

Algorithm 8 Algorithm exp_1

Input: a double-double value $r_h + r_\ell$
Output: $e_h + e_\ell$ approximating $\exp(r_h + r_\ell)$
1: $\rho_0 = \texttt{-0x1.74910ee4e8a27p+9} \approx -745.133$
2: $\rho_1 = \texttt{-0x1.577453f1799a6p+9} \approx -686.909$
3: $\rho_2 = \texttt{0x1.62e42e709a95bp+9} \approx 709.78267$
4: $\rho_3 = \texttt{0x1.62e4316ea5df9p+9} \approx 709.78276$
5: **if** $\rho_3 < r_h$ **then** return $e_h = e_\ell = \Omega$
6: **if** $r_h < \rho_0$ **then** return $e_h = \alpha$, $e_\ell = -\alpha$
7: **if** $r_h < \rho_1$ or $\rho_2 < r_h$ **then** return $e_h = e_\ell = \text{NaN}$
8: $\text{INVLN2} \leftarrow \texttt{0x1.71547652b82fep+12}$
9: $k \leftarrow \lfloor \circ(r_h \cdot \text{INVLN2}) \rceil$ ▷ nearest integer
10: $\text{LN2H} \leftarrow \texttt{0x1.62e42fefa39efp-13}$
11: $\text{LN2L} \leftarrow \texttt{0x1.abc9e3b39803fp-68}$
12: $z_h \leftarrow \circ(r_h - k\text{LN2H})$
13: $z_\ell \leftarrow \circ(r_\ell - k\text{LN2L})$
14: $z \leftarrow \circ(z_h + z_\ell)$
15: write $k = e \cdot 2^{12} + i_2 \cdot 2^6 + i_1$ with $0 \leq i_2, i_1 < 2^6$
16: let $h_2 + \ell_2$ approximate $2^{i_2/64}$
17: let $h_1 + \ell_1$ approximate $2^{i_1/2^{12}}$
18: $p_h, s \leftarrow \text{ExactMul}(h_1, h_2)$
19: $t \leftarrow \circ(\ell_1 h_2 + s)$
20: $p_\ell \leftarrow \circ(h_1 \ell_2 + t)$
21: $q_h, q_\ell \leftarrow \text{q\_1}(z)$
22: $h, s \leftarrow \text{ExactMul}(p_h, q_h)$
23: $t \leftarrow \circ(p_\ell q_h + s)$
24: $\ell \leftarrow \circ(p_h q_\ell + t)$
25: $e_h, e_\ell \leftarrow \circ(2^e \cdot h), \circ(2^e \cdot \ell)$

Proof: See Appendix A-H. ∎

Lemma 7: With the assumptions from Lemma 5, let r_h, r_ℓ be the values computed by Algorithm 6. In the case $\rho_1 \le r_h \le \rho_2$, then the value $e_h + e_\ell$ returned by Algorithm exp_1 satisfies

$$\left|\frac{e_h + e_\ell}{\exp(r_h + r_\ell)} - 1\right| < 2^{-63.78597}.$$

Moreover, $e_h \ge 2^{-991}$ and $|e_\ell/e_h| \le 2^{-49.2999}$.

Proof: Since the assumptions from Lemma 5 hold, we deduce from Lemma 5 that $|r_h| \in [2^{-970}, 709.79]$, $|r_\ell/r_h| \le 2^{-23.8899}$, and $|r_\ell| \le 2^{-14.4187}$.

Appendix A-J shows that for $\rho_1 \le r_h \le \rho_2$, both e_h, $2^e \cdot h$ and $2^e \cdot (h + \ell)$ lie in $[2^{-991}, \Omega]$.

Appendix A-I analyzes Algorithm exp_1 and proves the bound $|e_\ell/e_h| \le 2^{-49.2999}$, while Appendix A-K establishes the relative error bound $2^{-63.78597}$ for $e_h + e_\ell$. ∎

D. Main algorithm

The main algorithm for the first phase is Algorithm 9 (phase_1), which calls in turn Algorithms log_1, mul_1 and exp_1 seen above. The notation $\mathrm{RU}(2^{-62.792})$ means the rounding upwards of the error bound $2^{-62.792}$; this value is precomputed or obtained at compile-time, and any value in $\mathbb{F}$ larger than $2^{-62.792}$ works. For the rounding test, we cannot use the algorithms from [7] which assume rounding to nearest-even, whereas our goal is to design algorithms working for all rounding modes. Instead, we compute a left bound u and a right bound v with the current rounding mode, and if both u and v round to the same value, x^y rounds to that value.

Algorithm 9 Algorithm phase_1

Input: two binary64 numbers x, y with $x > 0$
Output: the correct rounding of x^y, or FAIL

1: $h, \ell \leftarrow \log\_1(x)$
2: $r_h, r_\ell \leftarrow \mathrm{mul}\_1(h, \ell, y)$
3: $e_h, e_\ell \leftarrow \exp\_1(r_h, r_\ell)$
4: **if** $\sqrt{2}/2 < x < \sqrt{2}$ **then** $\varepsilon \leftarrow \mathrm{RU}(2^{-57.560})$
5: **else** $\varepsilon \leftarrow \mathrm{RU}(2^{-62.792})$
6: $u \leftarrow \circ(e_h + \circ(e_\ell - \varepsilon e_h))$
7: $v \leftarrow \circ(e_h + \circ(e_\ell + \varepsilon e_h))$
8: **if** $u = v$ **then** return u
9: **else** return FAIL

Theorem 1: The value returned by Algorithm phase_1 (if not FAIL) is correctly rounded.

Proof: If $r_h < \rho_1$, see Appendix A-L. If $\rho_2 < r_h$, see Appendix A-M. If $\rho_1 \le r_h \le \rho_2$, see Appendix A-N. ∎

For the first phase, the following four tables are used, which occupy a total of 6416 bytes:

- a table INVERSE used in Algorithm log_1, with $r = \mathrm{INVERSE}_i$ a 9-bit approximation of $1/t$ for $i \cdot 2^{-8} \le t < (i+1) \cdot 2^{-8}$, $181 \le i \le 362$, such that $rt - 1$ is exactly representable in binary64. When the interval $[i \cdot 2^{-8}, (i+1) \cdot 2^{-8}]$ contains 1, that is, for $i \in \{255, 256\}$, we use $r = 1$ to avoid cancellation in the computation of $\log z - \log r$. This table has 182 binary64 entries, thus occupies 1456 bytes;
- a table LOGINV used in Algorithm log_1 such that for $181 \le i \le 362$, LOGINV_i is a double-double approximation to nearest of $-\log r$, where r is defined above. This table has 182 entries with two binary64 values, thus occupies 2912 bytes;
- a table T_2 such that for $0 \le i < 64$, $T_2[i]$ is a double-double approximation of $2^{i/2^6}$, used in line 16 of Algorithm exp_1. It has 64 entries with two binary64 values, thus occupies 1024 bytes;
- a table T_1 such that for $0 \le i < 64$, $T_1[i]$ is a double-double approximation of $2^{i/2^{12}}$, used in line 17 of Algorithm exp_1. It has 64 entries with two binary64 values, thus occupies 1024 bytes.

IV. Second and Third Phases

The second phase uses 128-bit arithmetic (using two 64-bit integer words for the significands), and delivers an approximation with about 113 correct bits. It is very similar to the first phase, except for the computation of $\log x$, where we use a two-step (instead of single-step) argument reduction $z = r_1 r_2 t - 1$ so that $|z| \le 2^{-13}$; then we use a degree-9 polynomial for $\log(1 + z)$, another two-step argument reduction and a degree-7 polynomial for the computation of $\exp(r)$. When the rounding test of the second phase fails, exact and midpoint cases are checked using the algorithm from [15], where it is shown that all exact and midpoint cases belong to the following set:

$$\begin{aligned}\mathbb{S} &= \{(x, y) \in \mathbb{F}^2 \mid y \in \mathbb{N}, 2 \le y \le 35\} \\ &\cup \{(m, 2^F n) \in \mathbb{F}^2 \mid F \in \mathbb{Z}, -5 \le F < 0, \\ &\quad n \in 2\mathbb{N} + 1, 3 \le n \le 35, m \in 2\mathbb{N} + 1\}.\end{aligned}$$

We have computed hard-to-round cases for all pairs (x, y) from $\mathbb{S}$. If the rounding test of the second phase fails, we use Algorithm 1 from [15] to detect exact and midpoint cases.[1] Our relative error bound for the second phase is slightly worse than the bound 2^{-117} from [15, Algorithm 1], but no remaining worst case at that point defeats that algorithm.

The third phase uses 256-bit arithmetic (using four 64-bit integer words for the significands), and delivers an approximation with about 240 correct bits. For the computation of $\log x$, we use the same two-step argument reduction as in the second phase,

[1] For the new IEEE mode 'to nearest with ties to away', our algorithms should be easy to adapt, since the only difference with 'to nearest-even' is how to round midpoint cases, thus it suffices to adapt the routine handling them.

phase	error bound	throughput
1	$2^{-62.792}/2^{-57.560}$	54
2	$2^{-113.17}$	543
3	$2^{-240.44}$	1857

Fig. 3. Relative error bound and reciprocal throughput (in i7-8700 cycles) of the different phases. For the first phase, the error bound differs depending on whether $x \in (1/\sqrt{2}, \sqrt{2})$ or not. The second phase includes the exact/midpoint detection.

GNU libc 2.37	MathLib	LIBMCR	CRLIBM	this work
43	123	256	211	54
77	166	285	275	89

Fig. 4. Comparison of the reciprocal throughput (top) and latency (bottom) of some implementations of the binary64 power function, in terms of cpu cycles, on an Intel(R) Core(TM) i7-8700 with gcc 13.2.0, for rounding to nearest-even.

with a degree-18 polynomial for $\log(1+z)$, and another two-step argument reduction with a degree-14 polynomial for the computation of $\exp(r)$. Figure 3 summarizes the parameters of the three phases. In particular, the error bound of $2^{-62.792}$ of the first phase when $x \notin (1/\sqrt{2}, \sqrt{2})$ corresponds to a probability of about $1/1000$ of calling the second phase, thus to an average overhead of less than one cycle with respect to the first phase and corresponding rounding test.

Lauter estimates to 2^{112} the number of pairs (x, y) such that x^y lies in the binary64 range [14]. Thus a full search for hardest-to-round cases is not possible, even with clever algorithms like SLZ [21]. As a consequence, it is possible, albeit extremely unlikely, that the rounding test of the third phase fails. In such a case, an error message is printed, with the corresponding inputs x and y, which will enrich the knowledge of hard-to-round cases. This guarantees that whenever the function does not print this error message, the returned value is correctly rounded. If worst-case information was available, one could avoid the rounding test of the third phase, by adding some exceptional cases if needed. This would ensure the function never yields an error message and always returns a correctly-rounded result, but this would have little impact on its efficiency.

V. Correctness

We have tested the correctness of our implementation, which is publicly available at https://gitlab.inria.fr/zimmerma/core-math-power-b64, for all rounding modes, on a set of 917231 input pairs (x, y). This set includes: (a) worst cases for exponents y that might yield exact or midpoint cases, including cases where x^y lies in the subnormal range; (b) special values specified by IEEE 754-2019, for example $(x, y) = (1, -0)$ or $(\pm\infty, -0)$; (c) pairs with $x < 0$ and y an integer; (d) pairs with x near 1 and x^y near underflow/overflow; (e) exact and midpoint cases; (f) inputs exhibiting bugs in other libraries; (g) non-regression tests; (h) additional worst cases found in the literature, in particular from Section 14.14 in [1], and worst cases computed by Vincent Lefèvre for x^n and $x^{1/n}$, n integer. We also tested our implementation on a set of 10^8 random inputs pairs, again for all rounding modes. For all these tests, we used as reference the value given by GNU MPFR [9], with exponent range matching that of binary64, and emulating subnormals using `mpfr_subnormalize`.

VI. Efficiency

In this section, we measure the efficiency of our implementation in the C language of the algorithms described in this paper (CORE-MATH commit 44c0399). We measure both the reciprocal throughput and the latency, using the CORE-MATH `perf.sh` tool [20]. For the power function, CORE-MATH generates random x and y uniformly in $[0, 20]$, so that x^y lies in $[0, 10^{26}]$ approximately. We compare our implementation to the GNU libc (which is not correctly rounded), to MathLib, CRLIBM, and CR-LIBM, for rounding to nearest-even.

For MathLib, we used the non-official copy from https://github.com/dreal-deps/mathlib, which we compiled with the default compile flags (using `-O3 -march=native` yields plenty of incorrect roundings). Our tests confirm that the MathLib `upow` routine does not provide correct rounding for directed rounding modes.

For LIBMCR, we used the non-official copy from https://github.com/simonbyrne/libmcr/, which matches our copy of release 0.9 from 2004. For rounding to nearest-even, on our set of about one million tests, the pow function from LIBMCR does not terminate for 15 inputs pairs, for example $x =$ `0x1.470574d68e0afp+1` and $y =$ `0x1.02e0706205c0ep+1`, and we get about 96% of failures on the remaining tests, for example for $x =$ `0x1.f80b060553772p-1` and $y =$ `0x1.99cp+13`, LIBMCR yields `0x1.00001p+0`.

The power function from CRLIBM is explicitly said as experimental, and only works for rounding to nearest-even. We also noticed some issues, for example for $x =$ `0x1.524ebae943097p+1` and $y =$ `0x1.ep-2`, it returns -5 instead of `0x1.93bd0cd47eb5fp+0`. Looking at the code, it appears that the strange return value of -5 corresponds to a failure in the last rounding test, which is not treated (here, x^y has 68 identical bits after the round bit).

Figure 4 shows that our implementation is only about 26% slower than the GNU libc, which is not correctly rounded (only about 16% for the latency). (On a i7-1260P, we get a reciprocal throughput of 27 cycles and a latency of 64 cycles, against 22 and 53 respectively for the GNU libc.) While Figure 4 only

considers rounding to nearest-even, timings of our implementation are very similar for other rounding modes. For the reciprocal throughput, our implementation outperforms that of MathLib by a factor 2.3, that of CRLIBM by a factor 3.9, and that of LIBMCR by a factor 4.7.

VII. Formal Proofs

All the proofs presented in this paper and in [24] have been formally verified using the Coq proof assistant [23]. A system like Coq lets the user interactively write formal proofs. In order to apply it to our algorithms, we use a specific library, Flocq [2], that enables us to talk about floating-point numbers within the proof assistant. To get an idea of how it looks, let us consider the FastTwoSum algorithm (Algorithm 2). Its definition in the formal world is the following

```
Definition fastTwoSum (a b : ℝ) :=
 let h := RND (a + b) in
 let t := RND (h - a) in DWR s (RND (b - t)).
```

The FastTwoSum algorithm returns a pair of reals, so we have a dedicated constructor *DWR*, i.e., *DWR* x y represents the pair (x, y).

Floating-point numbers in Coq are a subset of the real numbers: a real number x (x of type $\mathbb{R}$) is a floating-point number if it verifies the *format* predicate (*format* x). The format we have been using is called *FLT*. It is parametrized by a radix β (here $\beta = 2$), a precision p (here $p = 53$) and a minimal exponent $emin$ (here $emin = -1074$) but no maximal exponent. This means that our formalisation faithfully captures underflows but not overflows. Nevertheless all the arguments in the paper that justify that no overflow occurs have also been formalized.

If we come back to our example, the formal statement of its correctness is the following lemma

```
Lemma fastTwoSum_correct a b :
 format a ⇒ format b ⇒ (a ≠ 0 ⇒ |b| ≤ |a|) ⇒
 let: DWR h l := fastTwoSum a b in
 |h + l - (a + b)| ≤ pow (1 - 2 × p) × |h|.
```

where *pow* z represents the real function 2^z. This theorem is proved in the general setting following the paper proof in [24, Theorem 1], where p is an arbitrary precision and applied in the formalisation with $p = 53$.

All the theorem statements presented in this paper have a Coq equivalent. A list that links the theorems of the paper with their associated formal lemma is given in the `README` file of the repository:

`https://github.com/thery/ExpFloat`

The formalisation has been done a posteriori, the paper proofs were already there. As Coq is an interactive theorem prover, we have tried to follow closely the paper proofs. So, we not only have the correctness of these statements but this let us spot several minor errors in the initial version of the paper proofs. These errors have obviously been corrected. At two occasions, we had to rework some part of the original proofs and provide a more direct and nicer argument.

Finally, the tables and the Sollya polynomials used in the paper have also been formally checked. This has been made possible thanks to the presence of the very powerful Coq tactic *interval* [3]. This tactic uses interval arithmetic and Taylor models to prove automatically inequalities over a restricted set of operators (exponential, logarithm, sin, cos, power function, ...). For example, the relative error given in the caption of Figure 1 is formally checked using a formally verified Taylor model of the logarithm of degree 50 using a precision on the coefficients of 200 bits.

VIII. Conclusion

We have proposed algorithms for computing the power function x^y with correct rounding, for the binary64 floating-point format and the four main rounding modes (to nearest-even as well as the three directed roundings of IEEE 754), and up to the knowledge of hardest-to-round cases. Our approach makes heavy use of the FMA instruction (either in hardware or emulated in software), and leads to a very efficient implementation, which is 2x to 5x faster than the power functions from MathLib, CRLIBM, and LIBMCR, claiming correct rounding.

When designing these algorithms, our aim has been to ensure both correctness and high efficiency. Efficiency follows from a three-phase approach whose first phase handles most binary64 inputs (x, y) very fast thanks to carefully designed FMA-based double-double computations. Correctness is obtained by returning either the correctly-rounded value of x^y (for which detailed error analysis has been done for the first phase), or switch to the next (more accurate) phase. Our implementation handles all the special input cases (such as $x \leq 0$, x or y NaN or infinite, etc.) according to the current IEEE 754 specification [11, §9.2.1].

As can be seen from the material in Appendix, we have systematically based our algorithms and implementations on very detailed proofs, which allows us to carefully control the accuracy of each of the routines involved as well as to predict and handle the (im)possibility of underflow and overflow. Furthermore, in the unlikely case where we cannot conclude that the correct value is returned for x^y, returning instead an error message along with the corresponding input (x, y) makes it possible to usefully augment the list of known hardest-to-round cases for that function.

Although our current implementation is already very fast, we do not claim that it cannot be improved further for speed. We hope that our results will contribute to turn the current IEEE recommendation of

correct rounding for the mathematical functions listed in [11, Table 9.1] into a requirement. Our approach also easily extends to other rounding modes (such as round to nearest with ties to away) and to other formats (such as double extended and binary128, or decimal formats). Finally, the high level of detail of our proofs enabled the last two authors to design a formal proof of the algorithms of the first phase, using the Coq proof assistant.

ACKNOWLEDGEMENTS

The authors are grateful to the three anonymous referees for their useful feedback, and to Vincenzo Innocente for his feedback at early stages of this research. The first author is partially funded by CERN, and the search for worst cases was performed using computer resources from CERN and Grid 5000.

REFERENCES

[1] BEEBE, N. H. F. *The Mathematical-Function Computation Handbook - Programming Using the MathCW Portable Software Library*. Springer, 2017.

[2] BOLDO, S., AND MELQUIOND, G. Flocq: A Unified Library for Proving Floating-Point Algorithms in Coq. *Proceedings - Symposium on Computer Arithmetic* (07 2011).

[3] BOLDO, S., AND MELQUIOND, G. Some formal tools for computer arithmetic: Flocq and Gappa. In *Proceedings of the 28th IEEE International Symposium on Computer Arithmetic* (June 2021), M. Joldes and F. Lamberti, Eds.

[4] CHEVILLARD, S., JOLDES, M. M., AND LAUTER, C. Sollya: an environment for the development of numerical codes. In *Third International Congress on Mathematical Software - ICMS 2010* (Kobe, Japan, 2010), K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama, Eds., vol. 6327 of *Lecture Notes in Computer Science*, Springer, pp. 28–31.

[5] CLARK, N. W., AND CODY, W. J. Self-contained exponentiation. In *AFIPS Conference Proceedings* (1969), vol. 35, ACM, pp. 701–706.

[6] DARAMY-LOIRAT, C., DEFOUR, D., DE DINECHIN, F., GALLET, M., GAST, N., LAUTER, C., AND MULLER, J.-M. CR-LIBM: A library of correctly rounded elementary functions in double-precision. Research report, LIP, 2006. https://hal-ens-lyon.archives-ouvertes.fr/ensl-01529804.

[7] DE DINECHIN, F., LAUTER, C., MULLER, J.-M., AND TORRES, S. On Ziv's rounding test. *ACM Trans. Math. Softw. 39*, 4 (2013), 25:1–25:19.

[8] DEMMEL, J., AND NGUYEN, H. D. Fast reproducible floating-point summation. In *21st IEEE Symposium on Computer Arithmetic* (2013), pp. 163–172.

[9] FOUSSE, L., HANROT, G., LEFÈVRE, V., PÉLISSIER, P., AND ZIMMERMANN, P. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw. 33*, 2 (2007), article 13.

[10] GRAILLAT, S., AND JÉZÉQUEL, F. Tight interval inclusions with compensated algorithms. *IEEE Transactions on Computers 69*, 12 (2020), 1774–1783.

[11] IEEE. *IEEE Standard for Floating-Point Arithmetic (IEEE Std 754-2019)*. 2019.

[12] INNOCENTE, V., AND ZIMMERMANN, P. Accuracy of mathematical functions in single, double, extended double and quadruple precision. https://members.loria.fr/PZimmermann/papers/accuracy.pdf, 2023. Version of February 14, 21 pages.

[13] ISO/IEC. C programming language – N3096, working draft of the standard, 2023. https://en.wikipedia.org/wiki/C2x.

[14] LAUTER, C. Q. *Arrondi correct de fonctions mathématiques. Fonctions univariées et bivariées, certification et automatisation*. PhD thesis, Université de Lyon - École Normale Supérieure de Lyon, 2008.

[15] LAUTER, C. Q., AND LEFÈVRE, V. An efficient rounding boundary test for $\mathrm{pow}(x, y)$ in double precision. *IEEE Trans. Comput. 58*, 2 (2009), 197–207.

[16] LE MAIRE, J., BRUNIE, N., DE DINECHIN, F., AND MULLER, J. Computing floating-point logarithms with fixed-point operations. In *23nd IEEE Symposium on Computer Arithmetic, ARITH 2016* (2016), P. Montuschi, M. J. Schulte, J. Hormigo, S. F. Oberman, and N. Revol, Eds., IEEE Computer Society, pp. 156–163.

[17] LIM, J. P., AND NAGARAKATTE, S. High performance correctly rounded math libraries for 32-bit floating point representations. In *PLDI'21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event* (2021), S. N. Freund and E. Yahav, Eds., ACM, pp. 359–374.

[18] The LLVM C Library. https://libc.llvm.org/.

[19] MARKSTEIN, P. *IA-64 and Elementary Functions: Speed and Precision*. Prentice Hall, 2000. Hewlett-Packard Professional Books.

[20] SIBIDANOV, A., ZIMMERMANN, P., AND GLONDU, S. The CORE-MATH Project. In *ARITH 2022 - 29th IEEE Symposium on Computer Arithmetic* (virtual, France, 2022). https://hal.inria.fr/hal-03721525.

[21] STEHLÉ, D., LEFÈVRE, V., AND ZIMMERMANN, P. Searching worst cases of a one-variable function using lattice reduction. *IEEE Transactions on Computers 54*, 3 (2005), 340–346.

[22] TANG, P. T. P. Table-driven implementation of the logarithm function in IEEE floating-point arithmetic. *ACM Trans. Math. Softw. 16*, 4 (1990), 378–400.

[23] THE COQ DEVELOPMENT TEAM. The Coq Proof Assistant, 2023. https://coq.inria.fr/.

[24] ZIMMERMANN, P. Note on FastTwoSum with Directed Roundings. Working paper or preprint, available at https://hal.inria.fr/hal-03798376, 2023.

[25] ZIV, A. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Trans. Math. Softw. 17*, 3 (1991), 410–423.

APPENDIX A
PROOFS

A. Analysis of Algorithm 4 and proof of Lemma 2

Our first goal is to bound $|p_h + p_\ell - (P(z) - z)|$. For convenience we write u', v', u'' to denote the variables computed at lines 5, 6, 7, respectively.

Exact operations. At line 1, the computation of $w_\ell = \circ(z^2 - w_h)$ is exact, since z^2 is in $2^{-122}\mathbb{Z}$ and at most $(33\cdot 2^{-13})^2 < 2^{-15.91}$. Hence $z^2 = w_h + w_\ell$. Furthermore, $|w_h| \le z^2 + \mathrm{ulp}(z^2) \le 2^{-15.91} + 2^{-68}$, $|w_\ell| \le \mathrm{ulp}(z^2) \le 2^{-68}$, and $w_h \in 2^{-122}\mathbb{Z}_{\ge 0}, w_\ell \in 2^{-122}\mathbb{Z}$. Note that the quantities $0.5\cdot w_h$ and $0.5\cdot w_\ell$ are computed exactly as well. In particular, $p_h = -w_h/2$. Finally, z^2, $z^2 - w_h$, $0.5\cdot w_h$, and $0.5\cdot w_\ell$ are in $2^{-123}\mathbb{Z}$ and of absolute value less than Ω, so underflow or overflow cannot occur when rounding them.

Local rounding errors. Let $\epsilon_1, \ldots, \epsilon_7$ denote the errors associated with the remaining operations, given by $t = P_8 z + P_7 + \epsilon_1$, $u = P_6 z + P_5 + \epsilon_2$, $v = P_4 z + P_3 + \epsilon_3$, $u' = t w_h + u + \epsilon_4$, $v' = u' w_h + v + \epsilon_5$, $u'' = v' w_h + \epsilon_6$, and $p_\ell = u'' z - w_\ell/2 + \epsilon_7$.

Since $|P_8| = -P_8 < P_7$ and $|z| < 1$, we have $0 < P_8 z + P_7 \le |P_8||z| + P_7$. Since $z \in 2^{-61}\mathbb{Z}$ and $P_7, P_8 \in 2^{-55}\mathbb{Z}$, we have $P_8 z + P_7 \in 2^{-116}\mathbb{Z}_{>0}$ and, using the bound $|z| \le 33\cdot 2^{-13}$ and the values of P_7 and P_8, we can check that $0 < P_8 z + P_7 < 2^{-2.8022}$. Hence there is no underflow/overflow when computing $t = \circ(P_8 z + P_7)$, and $|\epsilon_1| \le \mathrm{ulp}(P_8 z + P_7) \le 2^{-55}$ and $0 < t < 2^{-2.8022} + 2^{-55} < 2^{-2.802}$ and $t \in 2^{-116}\mathbb{Z}_{>0}$.

A similar analysis can be done for the computation of $u = \circ(P_6 z + P_5)$ and $v = \circ(P_4 z + P_3)$: there is no underflow/overflow when computing u and v, and $|\epsilon_2| \le 2^{-55}$, $|\epsilon_3| \le 2^{-54}$, $0 < u < 2^{-2.31709} + 2^{-55} < 2^{-2.317}$, $0 < v < 2^{-1.5806} + 2^{-54} < 2^{-1.580}$, $u \in 2^{-116}\mathbb{Z}_{>0}$, and $v \in 2^{-115}\mathbb{Z}_{>0}$.

It follows from $t \in 2^{-116}\mathbb{Z}_{>0}$, $w_h \in 2^{-122}\mathbb{Z}_{\ge 0}$, and $u \in 2^{-116}\mathbb{Z}_{>0}$ that $t w_h + u \in 2^{-238}\mathbb{Z}_{>0}$. On the other hand, the upper bounds on t, w_h, u seen before yield $t w_h + u < 2^{-2.802}\cdot(2^{-15.91} + 2^{-68}) + 2^{-2.31709} + 2^{-55} < 2^{-2.31707}$. Hence there is no underflow/overflow when computing $u' := \circ(t w_h + u)$, and $|\epsilon_4| \le \mathrm{ulp}(t w_h + u) \le 2^{-55}$, $0 < u' < 2^{-2.31707} + 2^{-55} < 2^{-2.31706}$, and $u' \in 2^{-238}\mathbb{Z}_{>0}$.

It follows from $u' \in 2^{-238}\mathbb{Z}_{>0}$, $w_h \in 2^{-122}\mathbb{Z}_{\ge 0}$, and $v \in 2^{-115}\mathbb{Z}_{>0}$ that $u' w_h + v \in 2^{-360}\mathbb{Z}_{>0}$. Furthermore, $u' w_h + v < 2^{-2.31706}\cdot(2^{-15.91} + 2^{-68}) + 2^{-1.5806} + 2^{-54} < 2^{-1.58058}$. Hence there is no underflow/overflow when computing $v' := \circ(u' w_h + v)$, and $|\epsilon_5| \le \mathrm{ulp}(u' w_h + v) \le 2^{-54}$, $0 < v' < 2^{-1.58058} + 2^{-54} < 2^{-1.5805}$, and $v' \in 2^{-360}\mathbb{Z}_{>0}$.

Since $v' \in 2^{-360}\mathbb{Z}_{>0}$ and $w_h \in 2^{-122}\mathbb{Z}_{\ge 0}$, we have $v' w_h \in 2^{-482}\mathbb{Z}_{\ge 0}$. Furthermore, $v' w_h < (2^{-1.58058} + 2^{-54})(2^{-15.91} + 2^{-68}) < 2^{-17.49057}$. Hence there is no underflow/overflow when computing $u'' := \circ(v' w_h)$, and $|\epsilon_6| \le \mathrm{ulp}(v' w_h) \le 2^{-70}$, $0 \le u'' < 2^{-17.49057} + 2^{-70} < 2^{-17.4905}$, and $u'' \in 2^{-482}\mathbb{Z}_{\ge 0}$.

Finally, since $u'' \in 2^{-482}\mathbb{Z}_{\ge 0}$, $z \in 2^{-61}\mathbb{Z}$, and $w_\ell/2 \in 2^{-123}\mathbb{Z}$, we have $u'' z - w_\ell/2 \in 2^{-543}\mathbb{Z}$. On the other hand, $|u'' z - w_\ell/2| \le u''|z| + |w_\ell|/2 \le (2^{-17.49057} + 2^{-70})\cdot 33\cdot 2^{-13} + 2^{-69} < 2^{-25.4461}$. Hence there is no underflow/overflow when computing $p_\ell := \circ(u'' z - w_\ell/2)$, and $|\epsilon_7| \le 2^{-78}$, $|p_\ell| < 2^{-25.4461} + 2^{-78} < 2^{-25.446}$, and $p_\ell \in 2^{-543}\mathbb{Z}$.

Global error. Let $E := |p_h + p_\ell - (P(z) - z)|$, where $P(z) - z = -\frac{1}{2}z^2 + z^3 Q(z)$ with $Q(z) = P_3 + P_4 z + R(z) z^2$ and $R(z) = P_5 + P_6 z + P_7 z^2 + P_8 z^3$. Since $-\frac{1}{2}z^2 = -\frac{1}{2}w_h - \frac{1}{2}w_\ell = p_h + p_\ell - u'' z - \epsilon_7$, we obtain $E = |u'' z + \epsilon_7 - z^3 Q(z)|$. Hence $E \le E_1|z| + \widetilde{E}_0$ with $E_1 := |u'' - z^2 Q(z)|$ and $\widetilde{E}_0 := |\epsilon_7|$.

Now, $u'' = v' z^2 - v' w_\ell + \epsilon_6$, so that $E_1 \le E_3 z^2 + \widetilde{E}_1$ with $E_3 := |v' - Q(z)|$ and $\widetilde{E}_1 := |- v' w_\ell + \epsilon_6|$.

Since $v' = u'(z^2 - w_\ell) + (P_4 z + P_3 + \epsilon_3) + \epsilon_5$, we have $v' - Q(z) = (u' - R(z)) z^2 - u' w_\ell + \epsilon_3 + \epsilon_5$. Hence $E_3 \le E_5 z^2 + \widetilde{E}_3$ with $E_5 := |u' - R(z)|$ and $\widetilde{E}_3 := |- u' w_\ell + \epsilon_3 + \epsilon_5|$.

Finally, $u' = t(z^2 - w_\ell) + (P_6 z + P_5 + \epsilon_2) + \epsilon_4$, so that $u' - R(z) = \epsilon_1 z^2 - t w_\ell + \epsilon_2 + \epsilon_4$. Hence $E_5 \le \widetilde{E}_7 z^2 + \widetilde{E}_5$ with $\widetilde{E}_7 := |\epsilon_1|$ and $\widetilde{E}_5 := |-t w_\ell + \epsilon_2 + \epsilon_4|$.

Thus $E \le \widetilde{E}(|z|) := \widetilde{E}_0 + \sum_{i=0}^{3} \widetilde{E}_{2i+1}|z|^{2i+1}$, where $|z| \le 33\cdot 2^{-13}$ and where each $\widetilde{E}_i$ can be bounded as follows: $\widetilde{E}_0 = |\epsilon_7| \le 2^{-78}$, $\widetilde{E}_1 \le v'|w_\ell| + |\epsilon_6| < (2^{-1.58058} + 2^{-54})\cdot 2^{-68} + 2^{-70} < 2^{-68.775}$, $\widetilde{E}_3 \le u'|w_\ell| + |\epsilon_3| + |\epsilon_5| < (2^{-2.31707} + 2^{-55})\cdot 2^{-68} + 2^{-54} + 2^{-54} < 2^{-52.999}$, $\widetilde{E}_5 \le t|w_\ell| + |\epsilon_2| + |\epsilon_4| < (2^{-2.8022} + 2^{-55})\cdot 2^{-68} + 2^{-55} + 2^{-55} < 2^{-53.999}$, and $\widetilde{E}_7 = |\epsilon_1| \le 2^{-55}$. All this then leads immediately to $E \le \widetilde{E}(|z|) < 2^{-75.513}$.

Conclusions. Since for $|z| \le 33\cdot 2^{-13}$ the approximation error $|P(z) - \log(1+z)| = |P(z) - z - (\log(1+z) - z)|$ is known to be at most $2^{-81.63}$, we conclude that Algorithm 4 produces p_h, p_ℓ such that $|p_h + p_\ell - (\log(1+z) - z)| < 2^{-75.513} + 2^{-81.63} < 2^{-75.492}$.

Furthermore, we have checked that underflows and overflows never occur during this polynomial evaluation.

Finally, we deduce from this analysis that $|p_h| = \frac{1}{2}w_h \le (33\cdot 2^{-13})^2/2 < 2^{-16.9}$ (the square of $33\cdot 2^{-13}$ is exactly representable in binary64), and that $|p_\ell| < 2^{-25.446}$.

At this stage, we have thus proven the bounds on $|p_h + p_\ell - (\log(1+z) - z)|$, $|p_h|$, and $|p_\ell|$ given in Lemma 2. What follows aims at establishing the relative error bound claimed in that lemma as well.

Relative error bounds. Assume now that $z \ne 0$ and let $\delta_0, \delta_1, \ldots, \delta_7$ be the relative errors associated with the computed values w_h, t, u, v, u', v', u'', p_ℓ, so that $w_h = z^2(1+\delta_0)$, $t = (P_8 z + P_7)(1+\delta_1)$, etc. Since, as already seen, there is no underflow and overflow during these computations, we have $w_\ell =$

$-z^2\delta_0$ and $|\delta_i| \le \lambda_i \mathbf{u}$ with $\lambda_i = 2$ and $\mathbf{u} = 2^{-53}$ for all i.

Smaller constants λ_i can sometimes be obtained, as follows. For $|z| \le 33 \cdot 2^{-13}$ and the values of $P_3, \dots, P_8$, we have $P_8 z + P_7 \in (2^{-2.8125}, 2^{-2.8022})$, $P_6 z + P_5 \in (2^{-2.3268}, 2^{-2.3170})$, and $P_4 z + P_3 \in (2^{-1.5894}, 2^{-1.5806})$, which implies $\lambda_1 < 2 \cdot 2^{-3}/2^{-2.8125} < 1.76$, $\lambda_2 < 2 \cdot 2^{-3}/2^{-2.3268} < 1.255$, and $\lambda_3 < 2 \cdot 2^{-2}/2^{-1.5894} < 1.505$. We already know that $tw_h + u < 2^{-2.31707}$. Furthermore, since $t > 0$ and $w_h \ge 0$, we have $tw_h + u \ge u = \circ(P_6 z + P_5) > 2^{-2.3268} - 2^{-55} > 2^{-2.3269}$, which implies $\lambda_4 \le 2 \cdot 2^{-3}/2^{-2.3269} < 1.255$. Similarly, we know that $u'w_h + v < 2^{-1.58058}$; since $u' > 0$ and $w_h \ge 0$, we have also $u'w_h + v \ge v = \circ(P_4 z + P_3) > 2^{-1.5894} - 2^{-54} > 2^{-1.5895}$, which gives $\lambda_5 < 2 \cdot 2^{-2}/2^{-1.5895} < 1.505$.

For readability, let us now define $A := P_8 z + P_7$, $B := P_6 z + P_5$, and $C = P_4 z + P_3$. Hence $t = A(1+\delta_1)$, $u = B(1+\delta_2)$, $v = C(1+\delta_3)$. Using $w_h = z^2(1+\delta_0)$, we deduce that $u' = (tw_h + u)(1+\delta_4)$ has the form $u' = Az^2(1+\delta_0)(1+\delta_1)(1+\delta_4) + B(1+\delta_2)(1+\delta_4)$, and that $v' = (u'w_h + v)(1+\delta_5)$ has the form $v' = Az^4(1+\delta_0)^2(1+\delta_1)(1+\delta_4)(1+\delta_5) + Bz^2(1+\delta_0)(1+\delta_2)(1+\delta_4)(1+\delta_5) + C(1+\delta_3)(1+\delta_5)$. Thus $u'' = v'w_h(1+\delta_6) = v'z^2(1+\delta_0)(1+\delta_6)$ can be written

$$u'' = Az^6(1+\theta_7) + Bz^4(1+\theta_6) + Cz^2(1+\theta_4)$$

with $1+\theta_7 := (1+\delta_0)^3(1+\delta_1)(1+\delta_4)(1+\delta_5)(1+\delta_6)$, $1+\theta_6 := (1+\delta_0)^2(1+\delta_2)(1+\delta_4)(1+\delta_5)(1+\delta_6)$, and $1+\theta_4 := (1+\delta_0)(1+\delta_3)(1+\delta_5)(1+\delta_6)$. Since $p_\ell = (u''z - \frac{1}{2}w_\ell)(1+\delta_7)$, $p_h = -\frac{1}{2}w_h$, $w_h + w_\ell = z^2$, and $w_\ell = -z^2\delta_0$, one can check that $p_h + p_\ell = -\frac{1}{2}z^2 + \frac{1}{2}\delta_0\delta_7 z^2 + u''z(1+\delta_7)$.

Defining $1+\theta_8 := (1+\theta_7)(1+\delta_7)$, $1+\theta'_7 := (1+\theta_6)(1+\delta_7)$, $1+\theta_5 := (1+\theta_4)(1+\delta_7)$ and recalling that $P(z) = z - \frac{1}{2}z^2 + z^3 Q(z)$, we obtain $u''z(1+\delta_7) = z^3 Q(z) + \theta_8 Az^7 + \theta'_7 Bz^5 + \theta_5 Cz^3$. Hence $p_h + p_\ell = P(z) - z + \frac{1}{2}\delta_0\delta_7 z^2 + \theta_5 Cz^3 + \theta'_7 Bz^5 + \theta_8 Az^7$. By dividing both this expression and $P(z) = z - \frac{1}{2}z^2 + z^3 Q(z)$ by $z \ne 0$ (and by noting that $P(z)/z > 0$ for all $|z| \in (0, 33 \cdot 2^{-13})$, we deduce that the relative error $|z + p_h + p_\ell - P(z)|/|P(z)|$ is exactly

$$\varphi(z) := \frac{|\frac{1}{2}\delta_0\delta_7 z + \theta_5 Cz^2 + \theta'_7 Bz^4 + \theta_8 Az^6|}{1 - \frac{1}{2}z + z^2 Q(z)}.$$

Let us now bound the numerator of $\varphi(z)$ by $B(|z|)$, where $B(X) := \sum_{i=1}^{7} B_i X^i$ and $B_i > 0$. First, $|\frac{1}{2}\delta_0\delta_7| \le 2^{-105} =: B_1$. Then, repeated use of inequalities such as $|\theta_5| \le (1+|\theta_4|)(1+|\delta_7|) - 1$ together with the bounds $|\delta_0| \le 2\mathbf{u}$, $|\delta_1| \le 1.76\mathbf{u}$, etc. allows us to choose the bounds $B_2 := 2^{-51.413}$, $B_3 := 2^{-51.828}$, $B_4 := 2^{-51.735}$, $B_5 := 2^{-51.998}$, $B_6 := 2^{-51.947}$, $B_7 := 2^{-52.139}$, for which we have

$$\varphi(z) \le \frac{B(|z|)}{1 - \frac{1}{2}z + z^2 Q(z)} =: \psi(z).$$

Now, $Q(z) = |P_3| - |P_4|z + |P_5|z^2 - |P_6|z^3 + |P_7|z^4 - |P_8|z^5 \ge Q(|z|)$ and $1 - \frac{1}{2}z \ge 1 - \frac{1}{2}|z|$, so that $\psi(z) \le \psi(|z|)$. Finally, it can the be checked that the function ψ is increasing over $(0, 33 \cdot 2^{-13})$, so $\varphi(z) \le \psi(z) \le \psi(|z|) \le \psi(33 \cdot 2^{-13}) < 2^{-67.31693}$. Together with the relative error bound $|P(z)/\log(1+z) - 1| \le 2^{-72.423}$ from Sollya, this gives $|(z + p_h + p_\ell)/\log(1+z) - 1| \le (1+2^{-67.31693})(1+2^{-72.423}) - 1 \le 2^{-67.2756}$.

A refinement via $|\delta_0| + |\delta_6| \le 3.505\mathbf{u}$. So far we have used $|\delta_0|, |\delta_6| \le 2\mathbf{u}$ and so $|\delta_0| + |\delta_6| \le 4\mathbf{u}$. We will see below that, in fact,

$$|\delta_0| + |\delta_6| \le 3.505\mathbf{u}, \qquad \mathbf{u} = 2^{-53}.$$

This allows to replace the bound $|\theta_5| \le 9.01\mathbf{u}$ used up so far by $|\theta_5| \le 8.515\mathbf{u}$ and to choose the smaller values $B_2 := 2^{-51.4949}$ and $B_3 := 2^{-51.9099}$ when defining the function $\psi(z)$. For $|z| \le 32 \cdot 2^{-13}$, we obtain $\varphi(z) \le \psi(32 \cdot 2^{-13}) < 2^{-67.4878}$ and $|(z + p_h + p_\ell)/\log(1+z) - 1| \le (1 + 2^{-67.4878})(1 + 2^{-72.423}) - 1 \le 2^{-67.441}$.

(One could also use $|\delta_0| + |\delta_6| \le 3.505\mathbf{u}$ to decrease slightly the bounds on $|\theta'_7|$ and $|\theta_8|$ and thus take smaller values for $B_4, \dots, B_7$, but this would have a negligible effect on the final relative error bound.)

Let us now check that $|\delta_0| + |\delta_6| \le 3.505\mathbf{u}$. We have $v := \circ(P_4 z + P_3) \ge -|P_4| \cdot 33 \cdot 2^{-13} + P_3 - 2^{-54} > 2^{-1.5894}$. Since $u'w_h \ge 0$, we have $v' := \circ(u'w_h + v) \ge 2^{-1.5894} \cdot (1 - 2^{-52}) =: v_0$. On the other hand, we have already seen $v' \le 2^{-1.5805} =: v_1$. Hence $v' \in [v_0, v_1]$ with $v_0 \approx v_1 \approx 0.33$ and, up to scaling and in order to bound the relative error of $u'' := \circ(v'w_h) = (v'w_h)(1+\delta_6)$ where $w_h = \circ(z^2) = z^2(1+\delta_0)$, we can assume that $|z| \in [1, 2)$. Then $z^2 \in [1, 4)$, $w_h \in [1, 4]$, and $v'w_h \in [v', 4v']$.

The case $w_h = 4$ is not possible, since whatever the rounding mode, for $z = 2 - 2\mathbf{u}$, which is the largest value of z smaller than 2, z^2 cannot round to 4.

If $w_h \in [1, 2)$, then $z^2 \in [1, 2)$ and $v'w_h \in [v', 2v') \subset [2^{-2}, 2^{-1}) \cup [2^{-1}, 1)$. The case $v'w_h < 2^{-1}$ gives $|\delta_6| \le 2\mathbf{u} \cdot 2^{-2}/(v'w_h) \le \mathbf{u}/(2v_0) < 1.505\mathbf{u}$, while the case $v'w_h \ge 2^{-1}$ gives $z^2 \ge 1/(2v'(1+\delta_0)) \ge 1/(2v'(1+2\mathbf{u}))$ and thus $|\delta_0| \le 2\mathbf{u}/z^2 \le 2\mathbf{u} \cdot 2v'(1+2\mathbf{u}) \le 4\mathbf{u}(1+2\mathbf{u})v_1 \le 1.34\mathbf{u} < 1.505\mathbf{u}$. Hence $|\delta_0| + |\delta_6| \le 3.505\mathbf{u}$.

If $w_h \in [2, 4)$, then the same analysis can be done, which yields $|\delta_0| + |\delta_6| \le 3.505\mathbf{u}$ as well.

B. Proof of Lemma 3

Here $t \in \mathbb{F} \cap (1/\sqrt{2}, \sqrt{2})$ and $t_i = i \cdot 2^{-8}$ for $181 \le i \le 363$. Since $\lfloor 1/\sqrt{2} \cdot 2^8 \rfloor = 181$ and $\lceil \sqrt{2} \cdot 2^8 \rceil = 363$, there is a unique $i \le 362$ such that $t \in [t_i, t_{i+1})$.

■ If $i \le 255$ then $t < 1$ and we approximate $t^{-1} > 1$ by $r \in \mathbb{F} \cap [1, 2)$ of the form $r = 1 + k \cdot 2^{-8}$, $0 \le k < 256$. More precisely, since $t^{-1} \in (t_{i+1}^{-1}, t_i^{-1}]$, we restrict ourselves to $r \in [\underline{r}_i, \overline{r}_i]$ with $\underline{r}_i = \lfloor t_{i+1}^{-1} \cdot 2^8 \rfloor \cdot$

2^{-8} and $\overline{r}_i = \lceil t_i^{-1} \cdot 2^8 \rceil \cdot 2^{-8}$. Since $t_i^{-1} > t_{i+1}^{-1} \geq 1$ for $i \leq 255$, we have $\overline{r}_i > \underline{r}_i \geq 1$.

In fact, r can take at most 4 distinct values: to see this, note that the nonnegative integers $\underline{k}_i, \overline{k}_i$ defined by $\underline{r}_i = 1 + \underline{k}_i \cdot 2^{-8}$ and $\overline{r}_i = 1 + \overline{k}_i \cdot 2^{-8}$ satisfy

$$\begin{aligned} 0 < \overline{k}_i - \underline{k}_i &= \lceil t_i^{-1} \cdot 2^8 \rceil - \lfloor t_{i+1}^{-1} \cdot 2^8 \rfloor \\ &< t_i^{-1} \cdot 2^8 - t_{i+1}^{-1} \cdot 2^8 + 2 \\ &= \frac{2^8 \cdot 2^8}{i(i+1)} + 2, \end{aligned}$$

which for $i \geq 181$ gives $\overline{k}_i - \underline{k}_i < 3.98\ldots$ and thus $1 \leq \overline{k}_i - \underline{k}_i \leq 3$.

Since $t \in \mathbb{F} \cap [t_i, t_{i+1})$ and $t < 1$, it has the form $t = t_i + j \cdot 2^{-53}$, $0 \leq j \leq 2^{45} - 1 =: j_{\max}$. The (signed) relative error of the approximation r to t^{-1} is thus $z_{i,k}(j) := rt - 1 = (1 + k \cdot 2^{-8})(t_i + j \cdot 2^{-53}) - 1$. Given i and k, it is an increasing function of j over $[0, j_{\max}]$, so that $|z_{i,k}(j)| \leq \max\{|z_{i,k}(0)|, |z_{i,k}(j_{\max})|\} =: B_{i,k}$ for all j. For each i and the (at most) 4 values of k, we can easily compute the bound $B_{i,k}$, and then deduce the minimum value $B_i := \min_k B_{i,k}$ together with $k_i := \arg\min_k B_{i,k}$. This gives $r_i := 1 + k_i \cdot 2^{-8}$ for $i \in \{181, \ldots, 255\}$.

If $i \notin \{187, 196, 199\}$, these computed values of r_i are such that $B_i \leq 2^{-8}$. Since $r_i \in 2^{-8}\mathbb{Z}$ and $t \in 2^{-53}\mathbb{Z}$, we obtain $2^{-8} \geq B_i \geq |r_i t - 1| \in 2^{-61}\mathbb{Z}$, so that $r_i t - 1$ is either zero or a normal binary64 number.

If $i \in \{187, 196\}$, the bounds B_i are slightly larger than 2^{-8} and satisfy $B_i = 2^{-7.9\ldots} \leq 2^{-7}$. Fortunately, in these two cases k_i is even ($k_i \in \{94, 78\}$), so $r_i \in 2^{-7}\mathbb{Z}$. It follows that $2^{-7} \geq |r_i t - 1| \in 2^{-60}\mathbb{Z}$, which is still enough to conclude that $r_i t - 1$ is either zero or a normal binary64 number.

If $i = 199$ then $B_i = 2^{-7.95\ldots}$ and $k_i = 73$ is odd. In this case, by adding the constraint that k be even, we find that $B_{i,k}$ is minimized for $k_i = 72$; this value then leads to $B_i = 33 \cdot 2^{-13}$, which is still $2^{-7.95\ldots} \leq 2^{-7}$ and we conclude as previously that $r_i t - 1 \in 2^{-60}\mathbb{Z}$ and is either zero or a normal binary64 number.

Note finally that the largest error $|rt - 1|$ is $33 \cdot 2^{-13} = 2^{-7.9556058\ldots} = 0.0040283203125$: this error is attained for $(i, k_i, j) = (199, 72, 0)$; furthermore, we have already said that $B_i \leq 2^{-8}$ for all $i \notin \{187, 196, 199\}$; if $i \in \{187, 196\}$, one can check that $B_i < 33 \cdot 2^{-13}$.

■ If $i \geq 256$ then $t \geq 1$ and we approximate $t^{-1} \leq 1$ by $r \in \mathbb{F} \cap [1/2, 1]$ of the form $r = 1 - k \cdot 2^{-9}$, $0 \leq k \leq 256$. (Note that k can be zero.) More precisely, $t^{-1} \in (t_{i+1}^{-1}, t_i^{-1}]$ with $t_{i+1}^{-1} < t_i^{-1} \leq 1$, and so we restrict ourselves to $r \in [\underline{r}_i, \overline{r}_i]$ with $\underline{r}_i = \lfloor t_{i+1}^{-1} \cdot 2^9 \rfloor \cdot 2^{-9}$ and $\overline{r}_i = \lceil t_i^{-1} \cdot 2^9 \rceil \cdot 2^{-9}$. Note that $\underline{r}_i < \overline{r}_i \leq 1$, and we can define the two nonnegative integers $\underline{k}_i, \overline{k}_i$ as $\underline{r}_i = 1 - \underline{k}_i \cdot 2^{-9}$ and $\overline{r}_i = 1 - \overline{k}_i \cdot 2^{-9}$. One can check that $0 < \underline{k}_i - \overline{k}_i < \frac{2^{17}}{256 \cdot 257} + 2 = 3.99\ldots$, so k takes at most 4 distinct values for each i.

Since $t \in \mathbb{F} \cap [t_i, t_{i+1})$ and $t \geq 1$, it has the form $t = t_i + j \cdot 2^{-52}$, $0 \leq j \leq 2^{44} - 1 =: j_{\max}$. The associated relative error function is now $z_{i,k}(j) := (1 - k \cdot 2^{-9})(t_i + j \cdot 2^{-52}) - 1$. As before, it increases with $j \in [0, j_{\max}]$ and so $|z_{i,k}(j)| \leq \max\{|z_{i,k}(0)|, |z_{i,k}(j_{\max})|\} =: B_{i,k} \leq \max_k B_{i,k} =: B_i$. We can compute both B_i and $k_i := \arg\min_k B_{i,k}$, which gives $r_i := 1 - k_i \cdot 2^{-9}$ for $i \in \{256, \ldots, 362\}$.

Now, in this case the computed values k_i always lead to error bounds $B_i \leq 2^{-8}$. Since $r_i \in 2^{-9}\mathbb{Z}$ and $t \in 2^{-52}\mathbb{Z}$, we deduce that the relative error satisfies $2^{-8} \geq B_i \geq |r_i t - 1| \in 2^{-61}\mathbb{Z}$. Hence $r_i t - 1$ is either zero or a normal binary64 number. □

C. Analysis of Algorithm 5

Range of exponent e in Algorithm log_1. Recall that $\alpha = 2^{-1074}$ and $\Omega < 2^{1024}$. For $x \in [\alpha, \Omega]$ and $t \in (1/\sqrt{2}, \sqrt{2})$, we have $2^e = xt^{-1} \in (\alpha/\sqrt{2}, \Omega \cdot \sqrt{2}) \subset (2^{-1074.5}, 2^{1024.5})$ and so, using the fact that e is an integer, $e \in [-1074, 1024]$ and $|e| < 2^{11}$.

About the constants LOG2H and LOG2L. Let

$$\begin{aligned} \text{LOG2H} &:= \lfloor \log 2 \cdot 2^{42} \rfloor \cdot 2^{-42} \\ &= 3048493539143 \cdot 2^{-42}, \\ \text{LOG2L} &:= \lfloor (\log 2 - \text{LOG2H}) \cdot 2^{97} \rfloor \cdot 2^{-97} \\ &= 544487923021427 \cdot 2^{-93}. \end{aligned}$$

Clearly, LOG2H $\in 2^{-42}\mathbb{Z}$. Furthermore, the bitstring below indicates that LOG2H is in fact the 43-bit floating-number nearest $\log 2$, which implies $|\text{LOG2H} - \log 2| < 2^{-44}$ (and not just $< 2^{-43}$).

$$\log 2 = \underbrace{0.\overbrace{1011\cdots0111}^{42}}_{\text{LOG2H}}00\underbrace{\overbrace{111101\cdots10000}^{53}}_{\text{LOG2L}}0000011\cdots$$

We see that LOG2H $\in \mathbb{F} \cap [1/2, 1)$ and LOG2L $\in \mathbb{F} \cap [2^{-45}, 2^{-44})$. From its definition, LOG2L $\in 2^{-93}\mathbb{Z}$, and the 7 bits after its 53rd bit imply $|\text{LOG2H} + \text{LOG2L} - \log 2| < 2^{-102}$ (and not just $< 2^{-97}$). More precisely, one can check that

$$-2^{-102.018} < \text{LOG2H} + \text{LOG2L} - \log 2 < 0.$$

Since $e \in [-1074, 1024]$, we obtain

$$-2^{-92.018} < e\,\text{LOG2H} + e\,\text{LOG2L} - e\log 2 < 2^{-91.949}$$

and so $\text{err8} := |e\,\text{LOG2H} + e\,\text{LOG2L} - e\log 2|$ is such that

$$\text{err8} = 0 \text{ if } e = 0, \quad \text{err8} \leq 2^{-91.949} \text{ if } e \neq 0.$$

About the values ℓ_1 and ℓ_2. One can check by inspection of table INVERSE that all its entries satisfy $r \in [\frac{181}{256}, \frac{509}{512}] \cup \{1\} \cup [\frac{129}{128}, \frac{361}{256}]$. Hence $\log r$ is either

zero or such that $|\log r| \geq |\log(\frac{509}{512})| > 0.00587$ and $|\log r| \leq |\log(\frac{181}{256})| < 0.347$.

Now, ℓ_1 is defined as the (unique) integral multiple of 2^{-42} nearest $-\log r$, that is,

$$\ell_1 := \lfloor (-\log r) \cdot 2^{42} \rceil \cdot 2^{-42}.$$

This implies $|\log r| - 2^{-43} \leq |\ell_1| \leq |\log r| + 2^{-43}$ and, using the range of $|\log r|$, we deduce that $|\ell_1| \in \{0\} \cup (0.00587, 0.347) \subset \{0\} \cup [2^{-8}, 1/2)$. Together with $\ell_1 \in 2^{-42}\mathbb{Z}$, this implies that ℓ_1 is either zero or a normal binary64 number. Furthermore, by inspecting each of the 182 entries of table INVERSE, one can check that

$$|\ell_1 - (-\log r)| \in \{0\} \cup [2^{-52}, 2^{-43}).$$

Let now ℓ_2 be defined as the binary64 number nearest $(-\log r) - \ell_1$. The range analysis done for $|\ell_1 - (-\log r)|$ implies that $|\ell_2| \in \{0\} \cup [2^{-52}, 2^{-43}]$, so $\ell_2 \in 2^{-104}\mathbb{Z}$ is either zero or a normal binary64 number, and that $\mathrm{ulp}(\ell_1 - (-\log r)) \leq 2^{-96}$, so

$$\text{err7} := |\ell_1 + \ell_2 - (-\log r)| \leq 2^{-97}.$$

About the values t_h and t_ℓ. Since e is an integer and LOG2H and ℓ_1 are integer multiples of 2^{-42}, we have $e\,\text{LOG2H} + \ell_1 \in 2^{-42}\mathbb{Z}$, and so is their rounding t_h. On the other hand, $|e\,\text{LOG2H} + \ell_1| \leq 1074 \cdot \text{LOG2H} + 0.347 < 744.8 \leq 2^{11}$. Hence $e\,\text{LOG2H} + \ell_1$ is either zero or a normal binary64 number, so the computation of t_h in Algorithm 5 is exact (and free of underflow and overflow):

$$t_h = e\,\text{LOG2H} + \ell_1.$$

If $e = 0$, then $|t_h| = |\ell_1|$; if $e \neq 0$, then $|t_h| \geq \text{LOG2H} - |\ell_1| > 0.693 - 0.347 = 0.346$. Therefore, $t_h \in 2^{-42}\mathbb{Z}$ and

$$|t_h| \in \begin{cases} \{0\} \cup [0.00587, 0.347] & \text{if } e = 0, \\ [0.346, 744.8] & \text{if } e \neq 0. \end{cases}$$

We deduce from $e \in \mathbb{Z}$, $\text{LOG2L} \in 2^{-93}\mathbb{Z}$, and $\ell_2 \in 2^{-104}\mathbb{Z}$ that $e\,\text{LOG2L} + \ell_2 \in 2^{-104}\mathbb{Z}$. Since $|\text{LOG2L}| < 2^{-44.048}$, we have $|e\,\text{LOG2L} + \ell_2| < 1074 \cdot 2^{-44.048} + 2^{-43} < 2^{-33.9}$, so $|e\,\text{LOG2L} + \ell_2|$ is either zero or in $[2^{-104}, 2^{-33.9})$, and its ulp is at most 2^{-86}. Thus the computation of $t_\ell := \circ(e\,\text{LOG2L} + \ell_2)$ is free of underflow and overflow, $t_\ell \in 2^{-104}\mathbb{Z}$ and is either zero or a normal binary64 number. More precisely, since $t_\ell = \ell_2$ when $e = 0$, we have

$$|t_\ell| \in \{0\} \cup \begin{cases} [2^{-52}, 2^{-43}] & \text{if } e = 0, \\ [2^{-104}, 2^{-33.8}] & \text{if } e \neq 0. \end{cases}$$

Also, defining $\text{err1} := |t_\ell - (e\,\text{LOG2L} + \ell_2)|$, we have

$$\text{err1} \leq 2^{-86} \qquad [\text{err1} = 0 \text{ if } e = 0].$$

First call to FastSum. If $t_h \neq 0$ then $|t_h| \geq 0.0058 \geq 33 \cdot 2^{-13} \geq |z|$, so the precondition of FastSum is satisfied. With $(h, t) = \text{FastTwoSum}(t_h, z)$ and $\ell = \circ(t + t_\ell)$, let

$$\text{err2} := |h + t - (t_h + z)|,$$
$$\text{err3} := |\ell - (t + t_\ell)|.$$

By definition of FastTwoSum, $h = \circ(t_h + z)$ and $t = \circ(z - \circ(h - t_h))$. When $t_h = 0$, we thus have $h = z$, $t = 0$, $\ell = t_\ell$, which implies $\text{err2} = \text{err3} = 0$ as well as the absence of underflow and overflow.

Assume now that $t_h \neq 0$. Since $t_h \in 2^{-42}\mathbb{Z}$ and $z \in 2^{-61}\mathbb{Z}$ (see proof of Lemma 3), we have $t_h + z \in 2^{-61}\mathbb{Z}$. On the other hand, $|t_h + z| \leq 744.8 + 33 \cdot 2^{-13} < 744.9$ (and $\leq 0.347 + 3 \cdot 2^{-33} < 0.35103$ if $e = 0$). Hence there is no underflow/overflow when computing h, and $h \in 2^{-61}\mathbb{Z}$, $|h| \leq |t_h + z| + \mathrm{ulp}(t_h + z) \leq 745$ (and ≤ 0.352 if $e = 0$). It follows that $h - t_h \in 2^{-61}\mathbb{Z}$ and $|h - t_h| \leq 2 \cdot 745 = 1490$ (and $\leq 0.352 + 0.347 = 0.699$ when $e = 0$). Hence there is no underflow/overflow when computing $\circ(h - t_h)$ and, according to [8, Theorem 3], this computation is exact: $\circ(h - t_h) = h - t_h$. Consequently, $t = \circ(t_h + z - h) \in 2^{-61}\mathbb{Z}$ and, recalling that $|t_h + z| < 2^{10}$ (and $< 2^{-1}$ if $e = 0$), we deduce that $|t| \leq \mathrm{ulp}(t_h + z) \leq 2^{-43}$ (and $< 2^{-54}$ if $e = 0$). Thus, no underflow/overflow can occur during this call to FastTwoSum and, using [24], we arrive at $\text{err2} \leq 2^{-105}|h| \leq 2^{-105} \cdot 745 \leq 2^{-94.458}$ (and $\leq 2^{-105} \cdot 0.352 \leq 2^{-106.5}$ if $e = 0$).

Since $t \in 2^{-61}\mathbb{Z}$ and $t_\ell \in 2^{-104}\mathbb{Z}$, we have $t + t_\ell \in 2^{-104}\mathbb{Z}$. Furthermore, $|t + t_\ell| \leq 2^{-43} + 2^{-33.8} \leq 2^{-33.79}$ (and $\leq 2^{-54} + 2^{-43} \leq 2^{-42.9}$ if $e = 0$). Hence there is no underflow/overflow when computing $\ell = \circ(t + t_\ell)$, $\ell \in 2^{-104}\mathbb{Z}$, $\text{err3} \leq \mathrm{ulp}(t + t_\ell) \leq 2^{-86}$ (and $\leq 2^{-95}$ if $e = 0$), and $|\ell| \leq |t + t_\ell| + \mathrm{ulp}(t + t_\ell) \leq 2^{-33.78}$ (and $\leq 2^{-42.89}$ if $e = 0$).

All this gives in particular

$$\text{err2} + \text{err3} \leq \begin{cases} 2^{-85.995} & \text{if } e \neq 0, \\ 2^{-94.999} & \text{if } e = 0. \end{cases}$$

Second call to FastSum. Since $\ell \in 2^{-104}\mathbb{Z}$ and $p_\ell \in 2^{-543}\mathbb{Z}$, we have $\ell + p_\ell \in 2^{-543}\mathbb{Z}$. Furthermore, $|\ell + p_\ell| \leq 2^{-33.78} + 2^{-25.446} \leq 2^{-25.441}$ (and $\leq 2^{-42.89} + 2^{-25.446} \leq 2^{-25.445}$ if $e = 0$). Hence, whatever the value of e, there is no underflow/overflow when computing $\circ(\ell + p_\ell)$, and we have $\mathrm{ulp}(\ell + p_\ell) \leq 2^{-78}$, $\circ(\ell + p_\ell) \in 2^{-543}\mathbb{Z}$, and $|\circ(\ell + p_\ell)| \leq 2^{-25.441} + 2^{-78} \leq 2^{-25.4409}$ (and $\leq 2^{-25.445} + 2^{-78} \leq 2^{-25.4449}$ if $e = 0$). It follows in particular that

$$\text{err5} := |\circ(\ell + p_\ell) - (\ell + p_\ell)| \leq 2^{-78}.$$

For convenience we write (h', ℓ') to denote the output of $\text{FastSum}(h, p_h, \circ(\ell + p_\ell))$ at line 10 of

Algorithm 5, and let $(h', t') = \text{FastTwoSum}(h, p_h)$ and $\ell' = \circ(t' + \circ(\ell + p_\ell))$. Let also

$$\text{err4} := |h' + t' - (h + p_h)|,$$
$$\text{err6} := |\ell' - (t' + \circ(\ell + p_\ell))|.$$

If $h = 0$, then $h' = p_h$, $t' = 0$, and $\ell' = \circ(\ell + p_\ell)$, which implies err4 = err6 = 0 as well as the absence of underflow and overflow.

Assume now that $h \neq 0$. If $t_h = 0$, then $|h| = |z|$. Since $1 \geq |z| \in \mathbb{F}$, $|h| = |z| \geq \circ(z^2) = 2|p_h| \geq |p_h|$. If $t_h \neq 0$, then $|h| \geq |t_h + z|(1 - 2^{-52}) \geq (|t_h| - |z|)(1 - 2^{-52}) \geq (0.0058 - 33 \cdot 2^{-13})(1 - 2^{-52}) \geq 2^{-16.9} \geq |p_h|$. Hence the condition on the input to FastTwoSum (and FastSum) is satisfied.

Since $h \in 2^{-61}\mathbb{Z}$ and $p_h \in 2^{-123}\mathbb{Z}$, we have $h + p_h \in 2^{-123}\mathbb{Z}$. Furthermore, $|h + p_h| \leq 745 + 9 \cdot 10^{-6}$ (and $\leq 0.352 + 9 \cdot 10^{-6}$ if $e = 0$), so there is no underflow/overflow when computing $h' = \circ(h + p_h)$, and we have $h' \in 2^{-123}\mathbb{Z}$, $|h'| \leq |h + p_h| + \text{ulp}(h + p_h) \leq 746$ (and ≤ 0.353 if $e = 0$). It follows that $h' - h \in 2^{-123}\mathbb{Z}$ and $|h' - h| \leq |h'| + |h| \leq 2 \cdot 746$ (and $\leq 0.352 + 0353$ if $e = 0$). Hence there is no underflow/overflow when computing $\circ(h' - h)$, this value is exactly $h' - h$ [8, Theorem 3], and so $t' = \circ(h + p_h - h')$. Hence $t' \in 2^{-123}\mathbb{Z}$ and, using $|h + p_h| < 2^{10}$ (and $< 2^{-1}$ if $e = 0$), $|t'| \leq \text{ulp}(h + p_h) \leq 2^{-43}$ (and $\leq 2^{-54}$ if $e = 0$). Hence there is no underflow/overflow during this call to FastTwoSum and, using [24], we obtain err4 $\leq 2^{-105}|h'| \leq 2^{-95.4589}$ (and $\leq 2^{-105} \cdot 0.353 \leq 2^{-106.502}$ if $e = 0$).

Since $t' \in 2^{-123}\mathbb{Z}$ and $\circ(\ell + p_\ell) \in 2^{-543}\mathbb{Z}$, we have $t' + \circ(\ell + p_\ell) \in 2^{-543}\mathbb{Z}$. Furthermore, $|t' + \circ(\ell + p_\ell)| \leq 2^{-43} + 2^{-25.4409} \leq 2^{-25.4408}$ (and $\leq 2^{-54} + 2^{-25.4449} \leq 2^{-25.44489}$ if $e = 0$). It follows that there is no underflow/overflow when computing $\ell' = \circ(t' + \circ(\ell + p_\ell))$, and we have $\ell' \in 2^{-543}\mathbb{Z}$, $|\ell'| \leq 2^{-25.4408}(1 + 2^{-52}) \leq 2^{-25.4407}$ (and $\leq 2^{-25.44489}(1 + 2^{-52}) \leq 2^{-25.44488}$ if $e = 0$), and err6 $\leq \text{ulp}(t' + \circ(\ell + p_\ell)) \leq 2^{-78}$ for all e. Hence, in particular,

$$\text{err4 + err6} \leq \begin{cases} 2^{-95.4589} + 2^{-78} & \text{if } e \neq 0, \\ 2^{-106.52} + 2^{-78} & \text{if } e = 0. \end{cases}$$

and so err4 + err6 $\leq 2^{-77.9999}$ for all e.

Final call to FastTwoSum (only when $e = 0$). Assume $e = 0$ and let (h'', ℓ'') denote the output of FastTwoSum(h',ℓ'), so that $h'' = \circ(h' + \ell')$ and $\ell'' = \circ(\ell' - \circ(h'' - h')) = \circ(\ell' + h' - h'')$ due to [8, Theorem 3]. Note first that with $h' \in 2^{-123}\mathbb{Z}$, $|h'| \leq 746$, $\ell' \in 2^{-543}\mathbb{Z}$, and $|\ell'| \leq 2^{-25}$, there cannot be underflow/overflow during these three operations. Let further

$$\text{err9} := |h'' + \ell'' - (h' + \ell')|.$$

If $h' = 0$, then $h'' = \ell'$ and $\ell'' = 0$, so there is no underflow/overflow and err9 = 0.

Assume now $h' \neq 0$. Since $e = 0$, $|t_h| = |\ell_1| \in \{0\} \cup (0.00587, 0.347)$. We consider separately the cases $i \notin \{255, 256\}$ and $i \in \{255, 256\}$.

If $i \notin \{255, 256\}$, then we see by inspecting table INVERSE that $r \neq 1$, which implies $\ell_1 \neq 0$ and thus $|t_h| = |\ell_1| \geq 0.00587$. Hence $h := \circ(t_h + z)$ satisfies $|h| \geq (|t_h| - |z|)(1 - 2^{-52}) \geq (0.00587 - 33 \cdot 2^{-13})(1 - 2^{-52}) \geq 2^{-9.09}$. Since $|p_h| \leq (33 \cdot 2^{-13})^2/2$, we deduce that $h' := \circ(h + p_h)$ satisfies $|h'| \geq (|h| - |p_h|)(1 - 2^{-52}) \geq 2^{-9.1}$. On the other hand, since $e = 0$, our previous analyses gave $|\ell'| \leq 2^{-25.44488}$. Consequently, $|h'| \geq |\ell'|$. Furthermore, we have seen that $h' \in 2^{-123}\mathbb{Z}$ with $|h'| \leq 0.353$, and that $\ell' \in 2^{-543}\mathbb{Z}$. Hence there is no underflow/overflow when calling FastTwoSum(h', ℓ'), and using [24] gives err9 $\leq 2^{-105}|h''| \leq 2^{-105}(|h'| + |\ell'|)(1 + 2^{-52}) \leq 2^{-105}(0.353 + 2^{-25.44488})(1 + 2^{-52}) \leq 2^{-106.502}$. To summarize,

$$\text{err9} \leq 2^{-106.502} \quad \text{if } i \notin \{255, 256\}.$$

If $i \in \{255, 256\}$, then the same bound holds (and is detailed later in Appendix A-F).

D. Proof of Lemma 4 when $e \neq 0$

In this case, necessarily $x \neq 1$, and the output of Algorithm 5 is what we have written (h', ℓ'). Let us first bound the ratio $|\ell'|/|h'|$. We have seen that for $e \neq 0$, $|\ell'| \leq 2^{-25.4408}$. On the other hand, $e \neq 0$ implies $|t_h| \geq \text{LOG2H} - 0.347 \geq 0.346147$, which for $|z| \leq 33 \cdot 2^{-13}$ leads to $|h| = |\circ(t_h + z)| \geq (0.346147 - 33 \cdot 2^{-13})(1 - 2^{-52}) \geq 0.342118$. Since $z^2 \leq (33 \cdot 2^{-13})^2 = 1089 \cdot 2^{-26} \in \mathbb{F}$, we have $|p_h| = \frac{1}{2}\circ(z^2) \leq 1089 \cdot 2^{-27}$ and thus $|h'| = |\circ(h + p_h)| \geq (0.342118 - 1089 \cdot 2^{-27})(1 - 2^{-52}) \geq 0.342$. All this leads to $|\ell'|/|h'| \leq 2^{-23.89}$.

Let us now determine $\varepsilon_{\log}$ such that for $x \neq 1$, $|(h' + \ell')/\log x - 1| \leq \varepsilon_{\log}$. Since $e \neq 0$, then $|\log x| = |e \log 2 + t| \geq \log 2 - |\log t|$ and, using $2^{-1/2} < t < 2^{1/2}$, we deduce that $|\log x| > \log 2 - \frac{1}{2}\log 2 = \frac{1}{2}\log 2 > 0.34657$. In what follows, we will find an upper bound on the absolute error $|h' + \ell' - \log x|$ and simply divide it by 0.34657 to deduce $\varepsilon_{\log}$.

Since $\log x = \log(1+z) - \log r + e \log 2$ and using the bounds on err1, ..., err8 seen before together with the bound err0 $:= |z + p_h + p_\ell - \log(1 + z)| \leq 2^{-75.492}$ from Lemma 2, one can check that $|h' + \ell' - \log x| \leq \sum_{i=0}^{8} \text{err}i \leq 2^{-75.492} + 2^{-86} + 2^{-85.995} + 2^{-95.4589} + 2^{-78} + 2^{-78} + 2^{-97} + 2^{-91.949} \leq 2^{-75.0558}$. Using $|\log x| \geq 0.34657$, we conclude that one can take

$$\varepsilon_{\log} = 2^{-73.527} \qquad \text{if } e \neq 0.$$

E. Proof of Lemma 4 for $e = 0$ and $i \notin \{255, 256\}$

In this case, we have seen that we can take err0 $\leq 2^{-75.492}$, err1 = err8 = 0, err2 + err3 $\leq 2^{-94.999}$, err4 + err6 $\leq 2^{-106.52} + 2^{-78}$, err5 $\leq 2^{-78}$, err7 $\leq 2^{-97}$, and err9 $\leq 2^{-106.502}$. Adding up all these

bounds yields $|h'' + \ell'' - \log x| \le 2^{-75.0573}$. Furthermore, we have $x = t \notin [t_{255}, t_{256}) \cup [t_{256}, t_{257})$. Since $t_{255} = 1 - 2^{-8}$ and $t_{257} = 1 + 2^{-8}$, this means $x \notin [1 - 2^{-8}, 1 + 2^{-8})$ and, therefore, $|\log x| \ge 2^{-8.0029}$. Dividing the absolute error bound $2^{-75.0573}$ by $2^{-8.0029}$ gives the relative error bound

$$\varepsilon_{\log} = 2^{-67.0544} \qquad \text{if } e = 0 \text{ and } i \notin \{255, 256\}.$$

Furthermore, $h'' := \circ(h' + \ell')$ satisfies $|h''| \ge (|h'| - |\ell'|)(1 - 2^{-52}) \ge (2^{-9.1} - 2^{-25.44488})(1 - 2^{-52}) \ge 2^{-9.10002}$. Since the second operation of FastTwoSum is exact, we have $\ell'' = \circ(h' + \ell' - h'')$ and $|\ell''| \le \mathrm{ulp}(h' + \ell') \le \mathrm{ulp}(0.353 + 2^{-25.44488}) \le 2^{-54}$. Consequently, $|\ell''|/|h''| \le 2^{-44.89998}$.

F. Proof of Lemma 4 when $e = 0$ and $i \in \{255, 256\}$

In this case, $x = t \in [1-2^{-8}, 1+2^{-8})$, $r = 1$, $\ell_1 = \ell_2 = 0$, $z = x - 1 \in [-2^{-8}, 2^{-8})$, $t_h = t_\ell = 0$, $h = z$, and $\ell = 0$. Consequently, Algorithm 5 has only three steps: the computation of (p_h, p_ℓ) followed by $(h', \ell') := \mathrm{FastSum}(z, p_h, p_\ell)$ and then $(h'', \ell'') := \mathrm{FastTwoSum}(h', \ell')$. Recalling that $x \ne 1$, we see in particular that $0 < |z| \le 2^{-8}$.

By Lemma 2, we know that for $0 < |z| \le 2^{-8}$, $z + p_h + p_\ell = \log x \cdot (1 + \delta)$ with $|\delta| \le 2^{-67.441}$. Denoting by δ' and δ'' the relative errors of $h' + \ell'$ and $h'' + \ell''$, respectively, we thus have $h'' + \ell'' = (h' + \ell')(1 + \delta'') = (z + p_h + p_\ell)(1 + \delta')(1 + \delta'') = \log x \cdot (1+\delta)(1+\delta')(1+\delta'')$, and it remains to bound $|\delta'|$ and $|\delta''|$.

Relative error of $(h', \ell') := \mathbf{FastSum}(z, p_h, p_\ell)$. Here $h' = \circ(z + p_h)$ and we have already seen that no underflow/overflow occurs, that $|z| \ge |p_h|$, and that $\circ(h' - z)$ is exact, so $\ell' = \circ\big(\circ(z + p_h - h') + p_\ell\big)$. Denote by δ'_1, δ'_2, δ'_3 the associated relative errors: $h' = (z + p_h)(1 + \delta'_1)$ and $\ell' = \big((z + p_h - h')(1 + \delta'_2) + p_\ell\big)(1 + \delta'_3)$. We have $|\delta'_i| \le 2^{-52}$ for all i and one can check that $h' + \ell' - (z + p_h + p_\ell) = -\delta'_1(\delta'_2 + \delta'_3 + \delta'_2\delta'_3)(z + p_h) + \delta'_3 p_\ell$. Hence $|(h'+\ell') - (z + p_h + p_\ell)| \le (2^{-103} + 2^{-156})|z + p_h| + 2^{-52}|p_\ell|$. Let us now determine both an interval containing $|z + p_h|/|z|$ and an upper bound on $|p_\ell|/|z|$.

We have $|z + p_h| = |z - \frac{1}{2} \circ (z^2)| = |z - \frac{1}{2}z^2(1 + \delta_0)|$, $|\delta_0| \le 2^{-52}$. Using $0 < |z| \le 2^{-8}$ then gives $|z + p_h|/|z| = |1 - \frac{1}{2}z(1 + \delta)| \in [A, B]$ with $A := 1 - 2^{-9}(1 + 2^{-52})$ and $B := 1 + 2^{-9}(1 + 2^{-52})$.

For p_ℓ, recall that $p_\ell = \circ(u''z - \frac{1}{2}w_\ell)$ with $u'' = \circ(v' \circ (z^2))$, $0 < v' \le 2^{-1.5805}$ and $w_\ell = z^2 - \circ(z^2) = -\delta_0 z^2$. Hence $0 \le u'' \le 2^{-1.5805}(1 + 2^{-52})^2 z^2 \le 2^{-1.580499} z^2$, $|w_\ell| \le 2^{-52} z^2$, and thus $|p_\ell| \le (2^{-1.580499}|z|^3 + 2^{-53} z^2)(1 + 2^{-52})$. Using $0 < |z| \le 2^{-8}$ yields $|p_\ell|/|z| \le (2^{-1.580499} \cdot 2^{-16} + 2^{-53} \cdot 2^{-8})(1 + 2^{-52}) \le 2^{-17.58049} =: C$. Note that $A - C > 0$, and so $|z + p_h + p_\ell| \ge (A - C)|z| > 0$. The relative error $\delta' := (h' + \ell')/(z + p_h + p_\ell) - 1$ thus satisfies

$$|\delta'| \le \frac{(2^{-103} + 2^{-156})B|z| + 2^{-52}C|z|}{(A - C)|z|} \le 2^{-69.5776}.$$

Relative error of $(h'', \ell'') := \mathbf{FastTwoSum}(h', \ell')$. We have just seen that $z + p_h = \lambda z$ and $p_\ell = \mu z$ for some λ, μ such that $|\lambda| \in [A, B]$ and $|\mu| \le C$. Hence $h' = \lambda z(1 + \delta'_1)$ and $\ell' = (-\lambda z \delta'_1(1 + \delta'_2) + \mu z)(1 + \delta'_3)$. Since $|z| > 0$, this implies $|h'|/|z| \ge A(1 - 2^{-52}) \ge 0.998$ and $|\ell'|/|z| \le \big(B \cdot 2^{-52}(1 + 2^{-52}) + C\big)(1 + 2^{-52}) \le 2^{-17.5804}$. Consequently, the input (h', ℓ') to FastTwoSum satisfies $|h'| \ge |\ell'|$. We deduce using the relative error bound from [24]:

$$|\delta''| := \frac{|h'' + \ell'' - (h' + \ell')|}{|h' + \ell'|} \le 2^{-105}.$$

Finally, we also deduce from [8, Theorem 3] that the second operation in this FastTwoSum call is exact, so that $\ell'' = \circ(h' + \ell' - h'')$. Hence there exist δ''_1, δ''_2 such that $|\delta''_i| \le 2^{-52}$ and $h' + \ell' = h''(1 + \delta''_1)$ and $\ell'' = (h' + \ell' - h'')(1 + \delta''_2) = h''\delta''_1(1 + \delta''_2)$, which leads to $|\ell''|/|h''| \le 2^{-52}(1 + 2^{-52}) \le 2^{-51.99}$.

Conclusion. Combining the bounds just obtained for $|\delta'|$ and $|\delta''|$ with the bound $|\delta| \le 2^{-67.441}$, we deduce that the relative error of $h'' + \ell''$ with respect to $\log x$ is at most $(1 + 2^{-67.441})(1 + 2^{-69.5776})(1 + 2^{-105}) - 1 \le 2^{-67.145}$, so that

$$\varepsilon_{\log} = 2^{-67.145} \qquad \text{if } e = 0 \text{ and } i \in \{255, 256\}.$$

Furthermore, $|\ell''|/|h''| \le 2^{-51.99}$ in this case. Since the bound for $\varepsilon_{\log}$ is smaller than in the case $e = 0$ and $i \notin \{255, 256\}$, when $e = 0$ we can take $\varepsilon_{\log} = 2^{-67.0544}$, and we always have $|\ell''|/|h''| \le 2^{-44.89998}$.

G. Analysis of Algorithm 6 and proof of Lemma 5

The input values h and ℓ come from the output of Algorithm 5 and are finite binary64 numbers. Furthermore, Lemma 4 implies $h \ne 0$, $\ell = \lambda h$ with $|\lambda| \le 2^{-23.89}$, and $\log x = (h + \ell)(1 + \varepsilon_1) = h(1 + \lambda)(1 + \varepsilon_1)$ with $|\varepsilon_1| \le (1 - \varepsilon_{\log})^{-1} - 1$. On the other hand, y is a finite binary64 number, and we assume that $|yh| \in [A, B]$ with $A = 2^{-969}$ and $B = 709.7827$. The condition $|yh| \ge A$ implies $\lfloor \log_2 |yh| \rfloor =: e \ge -969$, and since $|yh|/2^e \in [1, 2) \cap 2^{-105}\mathbb{Z}$, we have $|yh| \in 2^{-105+e}\mathbb{Z} \subseteq 2^{-1074}\mathbb{Z} = \alpha\mathbb{Z}$.

These range conditions on yh imply that there is no underflow/overflow when computing $r_h := \circ(yh)$, and that $s := \circ(yh - r_h)$ is computed exactly: $s = yh - r_h$. (However, s might be zero or (sub)normal.) Hence $s = \delta_1 yh$ with $|\delta_1| \le 2^{-52}$, which gives $r_h = yh(1 - \delta_1)$ and so $|r_h| \in [A(1 - 2^{-52}), B(1 + 2^{-52})] \subset [2^{-970}, 709.79]$. Furthermore, $|y\ell + s| = |yh(\lambda + \delta_1)| \le B(|\lambda| + |\delta_1|) \le 2^{10}(2^{-23.89} + 2^{-52}) < 2^{-13}$. It follows that there is no overflow when computing $r_\ell := \circ(y\ell + s)$, but underflow may occur and so $r_\ell = (y\ell + s)(1 + \delta_2) + \varepsilon_2$ with $|\delta_2| \le 2^{-52}$, $|\varepsilon_2| \le \alpha$, and $\delta_2\varepsilon_2 = 0$. Since $\ell = \lambda h$ and $s = \delta_1 yh$, we deduce that $r_\ell = yh(\lambda + \delta_1)(1 + \delta_2) + \varepsilon_2$.

Hence $|r_\ell| \le B(2^{-23.89} + 2^{-52})(1 + 2^{-52}) + 2^{-1074} \le 2^{-14.4187}$. Furthermore, $r_\ell/r_h = (\lambda +$

$\delta_1)(1+\delta_2)(1-\delta_1)^{-1}+\varepsilon_2/(yh)\cdot(1-\delta_1)^{-1}$, so that $|r_\ell/r_h| \le (2^{-23.89}+2^{-52})(1+2^{-52})(1-2^{-52})^{-1}+2^{-1074+969}\cdot(1-2^{-52})^{-1} \le 2^{-23.8899}$.

It also follows from the above expressions of r_h and r_ℓ that $r_h+r_\ell = yh\big(1+\lambda(1+\delta_2)+\delta_1\delta_2\big)+\varepsilon_2$. This implies first that $|r_h+r_\ell| \le 709.7827\cdot(1+2^{-23.89}(1+2^{-52})+2^{-104})+2^{-1074} \le 709.79$. Second, recalling that $y\log x = yh(1+\lambda)(1+\varepsilon_1)$, we obtain $r_h+r_\ell - y\log x = yh\big(-(1+\lambda)\varepsilon_1+(\lambda+\delta_1)\delta_2\big)+\varepsilon_2$. It follows that the absolute error $|r_h+r_\ell-y\log x|$ is upper bounded by $C := B\cdot\big((1+2^{-23.89})|\varepsilon_1|+(2^{-23.89}+2^{-52})\cdot 2^{-52}\big)+2^{-1074}$. With $B=709.7827$ and $|\varepsilon_1| \le (1-\varepsilon_{\log})^{-1}-1$, the conclusion follows from Lemma 4: for $x \notin (1/\sqrt{2},\sqrt{2})$, $|\varepsilon_1| \le 2^{-73.5269}$ and $C \le 2^{-63.799}$; otherwise, $|\varepsilon_1| \le 2^{-67.05439}$ and $C \le 2^{-57.580}$. □

H. Analysis of Algorithm 7 and proof of Lemma 6

Let us first deal with underflow/overflow issues. Since all the coefficients of Q are in $[0,1]$, and $|z|$ is also bounded by 1, it is easy to see that all quantities are bounded by, say, 12, thus no overflow occurs.

Algorithm q_1 first computes $q = \circ(Q_4 z + Q_3)$. Since $|Q_4| < 2^{-4.58}$, $|Q_3| < 2^{-2.58}$, and $|z| \le 2^{-12.905}$, we get $|q| \le (2^{-4.58}\cdot 2^{-12.905}+2^{-2.58})(1+2^{-52}) \le 2^{-2.579}$. With $Q := Q_4 z + Q_3$, we have

$$|q-Q| \le \mathrm{ulp}(q) \le 2^{-55}.$$

Let $q' := \circ(qz+Q_2)$ be the value computed at line 2. Since $|q| \le 2^{-2.579}$, $|z| \le 2^{-12.905}$, and $Q_2 = 1/2$, we get $|q'| \le (2^{-2.579}\cdot 2^{-12.905}+1/2)(1+2^{-52}) \le 0.50003$. Defining $Q' := Q_4 z^2 + Q_3 z + Q_2$, we deduce that

$$\begin{aligned} |q'-Q'| &\le \mathrm{ulp}(q') + |q-Q||z| \\ &\le 2^{-53} + 2^{-55}\cdot 2^{-12.905} \\ &\le 2^{-52.999952}. \end{aligned}$$

Now let $h_0 := \circ(q'z+Q_1)$ be the value computed at line 3. Since $|q'| \le 0.50003$, $|z| \le 2^{-12.905}$, and $Q_1 = 1$, we have $|h_0| \le (0.50003\cdot 2^{-12.905}+1)(1+2^{-52}) \le 1.0001$. Writing $H_0 := Q_1 + Q'Z$, we deduce that

$$\begin{aligned} |h_0 - H_0| &\le \mathrm{ulp}(h_0) + |q'-Q'||z| \\ &\le 2^{-52} + 2^{-52.999952}\cdot 2^{-12.905} \\ &\le 2^{-51.999905}. \end{aligned}$$

Now we compute $h_1, \ell_1 = \mathrm{ExactMul}(z, h_0)$. Lemma 1 gives $|h_1 + \ell_1 - zh_0| < \alpha$. Since $|z| \le 2^{-12.905}$ and $|h_0| \le 1.0001$, we have $|h_1| = |\circ(zh_0)| \le 2^{-12.904}$ and $|\ell_1| \le \mathrm{ulp}(h_1) \le 2^{-65}$, and at the end:

$$\begin{aligned} |h_1+\ell_1 - zH_0| &= |h_0-H_0||z| + \alpha \\ &\le 2^{-51.999905}\cdot 2^{-12.905} + \alpha \\ &\le 2^{-64.9049049} =: \epsilon_1. \end{aligned}$$

The last instruction $q_h, q_\ell = \mathrm{FastSum}(Q_0, h_1, \ell_1)$ decomposes into $q_h, v = \mathrm{FastTwoSum}(Q_0, h_1)$ with $Q_0 = 1 \ge |h_1|$, followed by $q_\ell = \circ(v+\ell_1)$. Hence

$$\begin{aligned} &|q_h + q_\ell - (Q_0 + zH_0)| \\ &\quad\le |q_h + q_\ell - (Q_0 + h_1 + \ell_1)| + |h_1 + \ell_1 - zH_0| \\ &\quad\le |q_h + v - (Q_0 + h_1)| + |q_\ell - (v + \ell_1)| + \epsilon_1 \\ &\quad\le 2^{-105}|q_h| + \mathrm{ulp}(q_\ell) + \epsilon_1. \end{aligned}$$

Since $Q_0 = 1$ and $|h_1| \le 2^{-12.904}$, we deduce that $|q_h| \le (1+2^{-12.904})(1+2^{-52}) \le 1.0002$, thus $2^{-105}|q_h| \le 2^{-104.99}$. Now $|v| \le \mathrm{ulp}(q_h) \le 2^{-52}$, and since $|\ell_1| \le 2^{-65}$, we get $|q_\ell| \le 2^{-52} + 2^{-65} \le 2^{-51.999}$ since $2^{-52}+2^{-65}$ is exactly representable (which proves the bound on $|q_\ell|$ claimed in Lemma 6), thus $\mathrm{ulp}(q_\ell) \le 2^{-104}$. Recalling that $\epsilon_1 = 2^{-64.9049049}$, this yields finally $|q_h + q_\ell - (Q_0 + zH_0)| \le 2^{-104.99} + 2^{-104} + 2^{-64.9049049} \le 2^{-64.904904}$. Since $Q_0 + zH_0$ is the exact value of the polynomial Q at z and since we know by Sollya that $|Q(z) - \exp(z)| \le 2^{-74.34}$ for $|z| \le 2^{-12.905}$, we arrive at the following absolute error bound:

$$\begin{aligned} |q_h + q_\ell - \exp(z)| &\le |q_h + q_\ell - (Q_0 + zH_0)| \\ &\quad + |Q_0 + zH_0 - \exp(z)| \\ &\le 2^{-64.904904} + 2^{-74.34} \le 2^{-64.902821}. \end{aligned}$$

Now we divide by $\exp(z)$ to bound the relative error:

$$\left|\frac{q_h+q_\ell}{\exp(z)} - 1\right| < \frac{2^{-64.902821}}{\exp(z)},$$

and since $|z| \le 2^{-12.905}$, the right hand side is bounded by $2^{-64.902821}\cdot\exp(2^{-12.905}) \le 2^{-64.902632}$. This concludes the proof of Lemma 6.

I. Analysis of Algorithm 8 for $\rho_1 \le r_h \le \rho_2$.

About the constants INVLN2, LN2H, and LN2L. Let

$$\begin{aligned} \mathrm{INVLN2} &:= \lfloor 1/\log 2 \cdot 2^{52} \rceil \cdot 2^{-40} \\ &= 6497320848556798 \cdot 2^{-40}, \\ \mathrm{LN2H} &:= \lfloor \log 2 \cdot 2^{53} \rceil \cdot 2^{-65} \\ &= 6243314768165359 \cdot 2^{-65}, \\ \mathrm{LN2L} &:= \lfloor (\log 2 \cdot 2^{-12} - \mathrm{LN2H}) \cdot 2^{120} \rceil \cdot 2^{-120} \\ &= 7525737178955839 \cdot 2^{-120}. \end{aligned}$$

By definition, INVLN2 is the binary64 number nearest $2^{12}/\log 2$, LN2H is the binary64 number nearest $\log 2 \cdot 2^{-12}$, and finally LN2L is the binary64 number nearest $\log 2 \cdot 2^{-12} - \mathrm{LN2H}$. One can check that $|\mathrm{INVLN2} - 2^{12}/\log 2| < 2^{-43.447}$ and that $|\mathrm{LN2H} + \mathrm{LN2L} - \log 2 \cdot 2^{-12}| < 2^{-122.43}$. Furthermore, it is clear that INVLN2 $\in 2^{-40}\mathbb{Z}$, LN2H $\in 2^{-65}\mathbb{Z}$, and LN2L $\in 2^{-120}\mathbb{Z}$.

About the value k. From the range of r_h given in Lemma 5 and the range of INVLN2 seen just before, we deduce that there is no underflow/overflow when computing $\circ(r_h \cdot \mathrm{INVLN2})$. In particular,

$|\circ(r_h \cdot \text{INVLN2})| \le 709.79 \cdot (2^{12}/\log 2 + 2^{-43.447}) \cdot (1 + 2^{-52}) < 4194347.07$, so the integer k nearest $\circ(r_h \cdot \text{INVLN2})$ satisfies $|k| \le 4194347$.

Furthermore, $|k - (r_h + r_\ell) \cdot 2^{12}/\log 2| \le D_1 + D_2 + D_3 + D_4$ with $D_1 := |k - \circ(r_h\text{INVLN2})|$, $D_2 := |\circ(r_h\text{INVLN2}) - r_h\text{INVLN2}|$, $D_3 := |(r_h + r_\ell)(\text{INVLN2} - 2^{12}/\log 2)|$, and $D_4 := |r_\ell\text{INVLN2}|$. Clearly, $D_1 \le 1/2$. In addition, recalling from Lemma 5 that $|r_h| \le 709.79$, $|r_\ell| \le 2^{-14.4187}$, and $|r_h + r_\ell| \le 709.79$, we deduce that $|r_h\text{INVLN2}| < 2^{22.1}$, $D_2 \le \text{ulp}(2^{22.1}) \le 2^{-30}$, $D_3 \le 709.79 \cdot 2^{-43.447} \le 2^{-33.975}$, and $D_4 \le 2^{-14.4187} \cdot \text{INVLN2} \le 0.2698195$. All this gives $|k - (r_h + r_\ell) \cdot 2^{12}/\log 2| \le 0.7698196$ and, therefore,

$$|k \log 2 \cdot 2^{-12} - (r_h + r_\ell)| \le 2^{-12.906174}.$$

The subtraction $\circ(r_h - k\text{LN2H})$ is exact. Recalling from Lemma 5 that $|r_h| \in [2^{-970}, 709.79] \cap \mathbb{F}$ and since $\text{LN2H} \in (0,1) \cap 2^{-65}\mathbb{Z}$ and $|k| \le 4194347$, we have $\Omega \ge |r_h - k\text{LN2H}| \in 2^{-1022}\mathbb{Z} \subset \alpha\mathbb{Z}$. Hence there is no underflow/overflow when computing $z_h = \circ(r_h - k\text{LN2H})$.

Let us now show that this operation is exact. We have $|k - r_h/\text{LN2H}| \le D_1 + D_2 + D_3'$ with D_1, D_2 as before, and $D_3' := |r_h(\text{INVLN2} - 1/\text{LN2H})|$. As before, $D_1 \le 1/2$ and $D_2 \le 2^{-30}$. In addition, one can check that $|\text{INVLN2} - 1/\text{LN2H}| < 2^{-41.694}$, which leads to $D_3' \le 709.79 \cdot 2^{-41.694} \le 2^{-32.222}$. Hence $|r_h - k\text{LN2H}| \le (D_1 + D_2 + D_3')\text{LN2H} \le (1/2 + 2^{-30} + 2^{-32.222}) \cdot \text{LN2H} \le 2^{-13.528766}$.

Assume first that $|r_h| \ge 2^{-13}$. Then $r_h \in 2^{-65}\mathbb{Z}$ and so $r_h - k\text{LN2H} \in 2^{-65}\mathbb{Z}$. Hence $r_h - k\text{LN2H} = m \cdot 2^{-65}$ for some integer m such that $|m| \le 2^{65} \cdot 2^{-13.528766} \le 2^{53}$, and thus $r_h - k\text{LN2H}$ is in $\mathbb{F}$.

Assume now that $|r_h| < 2^{-13}$. Then $|r_h \cdot \text{INVLN2}| < 0.73$ and so $k := \lfloor \circ(r_h \cdot \text{INVLN2}) \rceil \in \{-1, 0, 1\}$. If $k = 0$, then clearly $r_h - k\text{LN2H} \in \mathbb{F}$. If $k = 1$, then $0 \le r_h < 2^{-13} < \text{LN2H} = 2^{-12.52...}$ and, on the other hand, $1/2 \le \circ(r_h \cdot \text{INVLN2})$. Hence $r_h \cdot \text{INVLN2} > 1/2 - 2^{-54}$ (the predecessor of $1/2$ in $\mathbb{F}$) and thus $r_h \ge \text{RU}\big((1/2 - 2^{-54})/\text{INVLN2}\big) = \text{LN2H}/2$. It follows from Sterbenz' theorem that $r_h - \text{LN2H}$ is in $\mathbb{F}$. Similarly, if $k = -1$ then $-2^{-13} < r_h \le 0$ and $\circ(r_h \cdot \text{INVLN2}) \le -1/2$, which requires $r_h \cdot \text{INVLN2} < -1/2 + 2^{-54}$ and so $r_h \le -\text{LN2H}/2$ and $r_h - k\text{LN2H} = -|r_h| + \text{LN2H} \in \mathbb{F}$ by Sterbenz' theorem.

We have thus shown that the value $z_h \in \mathbb{F}$ satisfies

$$z_h = r_h - k\text{LN2H}.$$

About the value z_ℓ. Since $|r_\ell| < 2^{-14.4187}$ (from Lemma 5) and $|k| \le 4194347$, we have $|r_\ell - k\text{LN2L}| \le 2^{-14.4187} + 4194347 \cdot \text{LN2L} < 2^{-14.418}$, so overflow cannot occur when computing $z_\ell := \circ(r_\ell - k\text{LN2L})$. If underflow occurs, then $|z_\ell - (r_\ell - k\text{LN2L})| \le 2^{-1074}$; otherwise, $|z_\ell - (r_\ell - k\text{LN2L})| \le \text{ulp}(r_\ell - k\text{LN2L}) \le \text{ulp}(2^{-14.418}) = 2^{-67}$. Hence in both cases the rounding error when computing z_ℓ satisfies

$$|z_\ell - (r_\ell - k\text{LN2L})| \le 2^{-67}.$$

About the value z. We have $|z_h + z_\ell| \le |z_h| + |z_\ell| = |r_h - k\text{LN2H}| + |z_\ell| \le 2^{-13.528766} + 2^{-14.418} + 2^{-67} \le 2^{-12.9059}$. Hence overflow cannot occur when computing $z := \circ(z_h + z_\ell)$. Furthermore, $|z - (z_h + z_\ell)|$ is at most 2^{-1074} if underflow occurs, and at most $\text{ulp}(2^{-12.9059}) = 2^{-65}$ otherwise. Hence in all cases

$$|z - (z_h + z_\ell)| \le 2^{-65}.$$

Since $|k(\text{LN2H} + \text{LN2L}) - k \log 2 \cdot 2^{-12}| \le 4194347 \cdot 2^{-122.43} \le 2^{-100.429}$, we deduce that

$$\begin{aligned}|z - (r_h + r_\ell - k \log 2 \cdot 2^{-12})| \\ \le 2^{-65} + 2^{-67} + 2^{-100.429} \le 2^{-64.67807}.\end{aligned}$$

Therefore, $r_h + r_\ell = k \log 2 \cdot 2^{-12} + z + \epsilon$ with $|\epsilon| \le 2^{-64.67807}$, which implies further $\exp(r_h + r_\ell) = 2^{k/2^{12}} \cdot \exp(z) \cdot \exp(\epsilon)$. Defining ϵ_1 by $\exp(\epsilon) =: 1 + \epsilon_1$, we have $|\epsilon_1| \le 2^{-64.67806}$, that is,

$$\left| \frac{\exp(r_h + r_\ell)}{2^{k/2^{12}} \cdot \exp(z)} - 1 \right| \le 2^{-64.67806}.$$

Finally, recalling that $|k \log 2 \cdot 2^{-12} - (r_h + r_\ell)| \le 2^{-12.906174}$, we arrive at $|z| \le 2^{-64.67807} + 2^{-12.906174} \le 2^{-12.905}$.

About the values h_1, ℓ_1, h_2, and ℓ_2. Given $i_1, i_2 \in \{0, \ldots, 63\}$, consider the numbers $2^{i_1/2^{12}}$ and $2^{i_2/64}$. These numbers are in $[1, 2)$ and their ulp is thus 2^{-52}.

Let h_1 be the binary64 number nearest $2^{i_1/2^{12}}$, and let ℓ_1 be the binary64 number nearest $2^{i_1/2^{12}} - h_1$. (These two numbers are normal binary64 numbers.) We have $h_1 = 2^{i_1/2^{12}} + \epsilon_1$ with $|\epsilon_1| \le 2^{-53}$, and ℓ_1 is the binary64 number nearest $-\epsilon_1$, in particular, $|\ell_1| \le 2^{-53}$. If $|\epsilon_1| = 2^{-53}$, then $\ell_1 = -\epsilon_1$ and thus $h_1 + \ell_1 = 2^{i_1/2^{12}}$; if $|\epsilon_1| < 2^{-53}$, then $\ell_1 = -\epsilon_1 + \epsilon_1'$ with $|\epsilon_1'| \le 0.5\text{ulp}(\epsilon_1) \le 2^{-107}$ and it follows that $|h_1 + \ell_1 - 2^{i_1/2^{12}}| \le 2^{-107}$. If $i_1 = 0$, then $2^{i_1/2^{12}} = 1$, $h_1 = 1$, and $\ell_1 = 0$, so $h_1 + \ell_1$ is exactly $2^{i_1/2^{12}}$; if $i_1 \ge 1$, then $2^{i_1/2^{12}} \ge 2^{1/2^{12}} > 2^{0.000244}$. Hence in both cases, the relative error of $h_1 + \ell_1$ satisfies

$$\left| \frac{h_1 + \ell_1}{2^{i_1/2^{12}}} - 1 \right| < 2^{-107.000244}.$$

Note also that an exhaustive search over the 64 possible values gives a relative error always less than $2^{-107.0228}$. Furthermore, $h_1 \in \mathbb{F} \cap [1, 2)$ and $\ell_1 \in \mathbb{F}$ is either 0 or such that $|\ell_1| > 2^{-57.53} \ge 2^{-58}$, so that $h_1 \in 2^{-52}\mathbb{Z}$ and $\ell_1 \in 2^{-110}\mathbb{Z}$.

Similarly, with h_2 the binary64 number nearest $2^{i_2/64}$ and ℓ_2 the binary64 number nearest $h_2 - 2^{i_2/64}$, one can check that $|\ell_2| \le 2^{-53}$ and (using now $2^{1/64} = 2^{0.015625}$) that

$$\left| \frac{h_2 + \ell_2}{2^{i_2/64}} - 1 \right| \le 2^{-107.015625}.$$

Note also that an exhaustive search over the 64 possible values gives a relative error always less than $2^{-107.57149}$. Furthermore, $h_2 \in \mathbb{F} \cap [1,2)$ and $\ell_2 \in \mathbb{F}$ is either 0 or such that $|\ell_2| > 2^{-58.98} \geq 2^{-59}$, so that $h_2 \in 2^{-52}\mathbb{Z}$ and $\ell_2 \in 2^{-111}\mathbb{Z}$.

Finally, by definition, $1 \leq h_1 \leq 2^{63/2^{12}} \cdot (1 + 2^{-53}) < 2^{0.015381}$ and $1 \leq h_2 \leq 2^{63/64} \cdot (1+2^{-53}) < 2^{0.984376}$, which implies that the exact product $h_1 h_2$ satisfies $1 \leq h_1 h_2 < 2^{0.999757} < 2$.

About the values p_h and p_ℓ. Since $h_1, h_2 \in \mathbb{F} \cap [1,2)$, there is no underflow/overflow when computing $p_h := \circ(h_1 h_2)$, and $h_1 h_2 - p_h \in \mathbb{F}$. Hence, recalling that $1 \leq h_1 h_2 < 2$, we have $p_h = h_1 h_2 + \epsilon_1$ with $|\epsilon_1| \leq \mathrm{ulp}(h_1 h_2) \leq 2^{-52}$. Since $h_1, h_2, p_h \in 2^{-52}\mathbb{Z}$, we have furthermore $|h_1 h_2 - p_h| \in 2^{-104}\mathbb{Z}$, so there is no underflow/overflow when computing $s := \circ(h_1 h_2 - p_h)$, and we thus have $s = h_1 h_2 - p_h$ and $2^{-52} \geq |s| \in 2^{-104}\mathbb{Z}$.

Next consider the computation $t := \circ(\ell_1 h_2 + s)$. Since $\ell_1 \in 2^{-110}\mathbb{Z}$ and $h_2 \in 2^{-52}\mathbb{Z}$, we have $\ell_1 h_2 + s \in 2^{-162}\mathbb{Z}$; since $|\ell_1| \leq 2^{-53}$ and $1 \leq h_2 \leq 2^{63/64} \cdot (1 + 2^{-53}) < 1.98$, we have also $|\ell_1 h_2 + s| < 2^{-53} \cdot 1.98 + 2^{-52} < 2^{-51.007}$. Hence there is no underflow/overflow when computing $t := \circ(\ell_1 h_2 + s)$, and we have $t \in 2^{-162}\mathbb{Z}$; furthermore, $t = \ell_1 h_2 + s + \epsilon_2$ with $|\epsilon_2| \leq \mathrm{ulp}(\ell_1 h_2 + s) \leq 2^{-104}$, which implies that $|t| \leq 2^{-51.007} + 2^{-104} < 2^{-51.00699}$.

Let us finally consider $p_\ell := \circ(h_1 \ell_2 + t)$. Using $h_1 \in 2^{-52}\mathbb{Z}$, $\ell_2 \in 2^{-111}\mathbb{Z}$, and $t \in 2^{-162}\mathbb{Z}$ gives $h_1 \ell_2 + t \in 2^{-163}\mathbb{Z}$. From $1 \leq h_1 \leq 2^{63/2^{12}} \cdot (1 + 2^{-53}) < 1.010719$ and $|\ell_2| \leq 2^{-53}$, we have also $|h_1 \ell_2 + t| \leq 1.010719 \cdot 2^{-53} + 2^{-51.00699} < 2^{-50.6805}$. Hence there is no overflow/underflow when computing $p_\ell := \circ(h_1 \ell_2 + t)$, and we have $p_\ell \in 2^{-163}\mathbb{Z}$; furthermore, $p_\ell = h_1 \ell_2 + t + \epsilon_3$ with $|\epsilon_3| \leq \mathrm{ulp}(h_1 \ell_2 + t) \leq 2^{-103}$, which implies that $|p_\ell| \leq 2^{-50.6805} + 2^{-103} \leq 2^{-50.680499}$.

Let us now show that $p_h + p_\ell \in [1,2)$. First, if $i_1 = i_2 = 0$, then $h_1 = h_2 = 1$, $\ell_1 = \ell_2 = 0$, $p_h = 1$, $s = t = p_\ell = 0$. Hence in this case $p_h + p_\ell = 1 \in [1,2)$. Assume now that $i_1 \geq 1$ or $i_2 \geq 1$. Then either $h_1 \geq 2^{1/2^{12}} \cdot (1 - 2^{-53})$ or $h_2 \geq 2^{1/64} \cdot (1 - 2^{-53})$, and so $p_h := \circ(h_1 h_2) \geq 2^{1/2^{12}} \cdot (1 - 2^{-53})(1 - 2^{-52}) > 1.0001$. Using the bound $|p_\ell| \leq 2^{-50.680499}$ seen just before, we deduce that $p_h + p_\ell \geq p_h - |p_\ell| \geq 1.0001 - 2^{-50.680499} \geq 1.00009 > 1$. On the other hand, $p_h + p_\ell \leq h_1 h_2 (1 + 2^{-52}) + |p_\ell| \leq 2^{0.999757} \cdot (1 + 2^{-52}) + 2^{-50.680499} < 1.9997 < 2$.

Finally, let us bound the relative error of $p_h + p_\ell$ with respect to $(h_1 + \ell_1)(h_2 + \ell_2)$. We have $p_h + p_\ell = (h_1 h_2 + \epsilon_1) + (h_1 \ell_2 + t + \epsilon_3)$ with $t = \ell_1 h_2 - \epsilon_1 + \epsilon_2$, and so $p_h + p_\ell - (h_1 + \ell_1)(h_2 + \ell_2) = \epsilon_2 + \epsilon_3 - \ell_1 \ell_2$. Hence $|p_h + p_\ell - (h_1 + \ell_1)(h_2 + \ell_2)| \leq |\epsilon_2| + |\epsilon_3| + |\ell_1| \cdot |\ell_2| \leq 2^{-104} + 2^{-103} + 2^{-53} \cdot 2^{-53} < 2^{-102.299}$.

In order to deduce a *relative* error bound, let us provide a lower bound on the exact product $(h_1 + \ell_1)(h_2 + \ell_2)$. Note first that if $i_1 = 0$, then $(h_1, \ell_1) = (1, 0)$, which implies that $p_h := \circ(h_1 h_2) = h_2$, $s := \circ(h_1 h_2 - p_h) = 0$, $t := \circ(\ell_1 h_2 + s) = 0$, and $p_\ell := \circ(h_1 \ell_2 + t) = \ell_2$. Hence, in this case, $p_h + p_\ell = h_2 + \ell_2$ is the exact product. Similarly, one can check that if $i_2 = 0$, then $p_h + p_\ell = h_1 + \ell_1$ is also the exact product. Hence it remains to consider the case where $i_1 \geq 1$ and $i_2 \geq 1$. In this case, we have $h_1 \geq 2^{1/2^{12}} \cdot (1 - 2^{-53})$ and $h_2 \geq 2^{1/64} \cdot (1 - 2^{-53})$, which together with the bounds $|\ell_1|, |\ell_2| \leq 2^{-53}$ leads to $(h_1 + \ell_1)(h_2 + \ell_2) > 1.0110603$. Hence, using $2^{-102.299}/1.0110603 < 2^{-102.314869}$, we arrive at the relative error bound

$$\left| \frac{p_h + p_\ell}{(h_1 + \ell_1)(h_2 + \ell_2)} - 1 \right| \leq 2^{-102.314869}.$$

By combining this bound with the relative error bounds $2^{-107.0228}$ and $2^{-107.57149}$ seen before for $h_1 + \ell_1$ and $h_2 + \ell_2$, we deduce that

$$\left| \frac{p_h + p_\ell}{2^{i_2/64 + i_1/2^{12}}} - 1 \right| \leq 2^{-102.2248}.$$

About the values h and ℓ. These values are defined by the four operations $h := \circ(p_h q_h)$, $s := \circ(p_h q_h - h)$, $t := \circ(p_\ell q_h + s)$, and $\ell := \circ(p_h q_\ell + t)$.

From Lemma 6, we know that $|q_\ell| \leq 2^{-51.999}$ and that $q_h = \exp(z) \cdot (1 + \delta) - q_\ell$ with $|z| \leq 2^{-12.905}$ and $|\delta| \leq 2^{-64.902632}$. It follows that $q_h \in [0.99986, 1.000131]$. Hence, since $q_h \in \mathbb{F}$, this implies $q_h \in 2^{-53}\mathbb{Z}$. Together with $p_h = \circ(h_1 h_2) \in \mathbb{F} \cap [1,2]$, this also implies that there is no underflow/overflow when computing $h := \circ(p_h q_h)$ and that $p_h q_h - h$ is in $\mathbb{F}$. Furthermore, $h \geq 0.99986 \cdot (1 - 2^{-52}) \geq 0.999859$, so that $h \in 2^{-53}\mathbb{Z}$, and it then follows from $p_h \in 2^{-52}\mathbb{Z}$ and $q_h \in 2^{-53}\mathbb{Z}$ that $p_h q_h - h \in 2^{-105}\mathbb{Z}$. Hence $s = p_h q_h - h \in 2^{-105}\mathbb{Z}$ and, since $p_h q_h \leq h_1 h_2 \cdot (1 + 2^{-52}) \cdot q_h \leq 2^{0.999757} \cdot (1 + 2^{-52}) \cdot 1.000131 < 2$, we have $h \leq 2$ and $|s| \leq \mathrm{ulp}(p_h q_h) \leq 2^{-52}$.

Using $p_\ell \in 2^{-163}\mathbb{Z}$, $q_h \in 2^{-53}\mathbb{Z}$, and $s \in 2^{-105}\mathbb{Z}$ gives $p_\ell q_h + s \in 2^{-216}\mathbb{Z}$. On the other hand, $|p_\ell q_h + s| \leq 2^{-50.680499} \cdot 1.000131 + 2^{-52} \leq 2^{-50.19424}$. Hence there is no underflow/overflow when computing $t := \circ(p_\ell q_h + s)$, and $t = p_\ell q_h + s + \epsilon_2$ with $|\epsilon_2| \leq \mathrm{ulp}(p_\ell q_h + s) \leq 2^{-103}$. It follows that $|t| \leq 2^{-50.19424} + 2^{-103} \leq 2^{-50.194239}$.

Consider finally the operation $\ell := \circ(p_h q_\ell + t)$. We have $|p_h q_\ell + t| \leq 2 \cdot 2^{-51.999} + 2^{-50.194239} \leq 2^{-49.541218}$, so there is no overflow. Hence $\ell = p_h q_\ell + t + \epsilon_3$ with $|\epsilon_3| \leq \mathrm{ulp}(p_h q_\ell + t) \leq 2^{-102}$ (which handles the possibility of underflow). It follows that $|\ell| \leq 2^{-49.541218} + 2^{-102} \leq 2^{-49.5412179}$, which together with $h \geq 0.999859$ shows that the ratio ℓ/h satisfies $|\ell/h| \leq 2^{-49.541}$.

We also deduce that $h + \ell = (p_h q_h - s) + (p_h q_h + t + \epsilon_3)$ with $t = p_\ell q_h + s + \epsilon_2$, so that $h + \ell - (p_h + p_\ell)(q_h + q_\ell) = \epsilon_2 + \epsilon_3 - p_\ell q_\ell$. Consequently, $|h + \ell - (p_h + p_\ell)(q_h + q_\ell)| \leq |\epsilon_2| + |\epsilon_3| + |p_\ell| \cdot |q_\ell| \leq 2^{-103} + 2^{-102} + 2^{-50.680499} \cdot 2^{-51.999} \leq 2^{-100.9129}$.

In order to deduce a *relative* error bound, we use the following lower bound on the exact product: with $p_h \geq 1$, $q_h \geq 0.99986$, $|p_\ell| \leq 2^{-50.680499}$, and $|q_\ell| \leq 2^{-51.999}$, we have $(p_h + p_\ell)(q_h + q_\ell) \geq (p_h - |p_\ell|)(q_h - |q_\ell|) \geq 0.999859 \geq 2^{-0.000204}$. Therefore,

$$\left| \frac{h+\ell}{(p_h+p_\ell)(q_h+q_\ell)} - 1 \right| \leq 2^{-100.912696}.$$

About the values e_h and e_ℓ. These values are defined by $e_h = \circ(2^e \cdot h)$ and $e_\ell = \circ(2^e \cdot \ell)$. Appendix A-J shows that when $\rho_1 \leq r_h \leq \rho_2$, we have $2^e \cdot h \in [2^{-991}, \Omega]$, thus the operation $e_h = \circ(2^e \cdot h)$ is exact. Since $|\ell/h| < 2^{-49.541}$ and $h \in [0.999859, 2]$, we have $|\ell| < \Omega$, thus no overflow can occur in $\circ(2^e \cdot \ell)$, but some underflow can occur. We thus have $e_\ell = 2^e \cdot \ell + \varepsilon$ with $|\varepsilon| \leq \alpha$. Using $2^e \cdot h \geq 2^{-1022}$, it follows that

$$\begin{aligned} |e_\ell/e_h| &\leq |\ell/h| + |\varepsilon|/(2^e \cdot h) \\ &\leq 2^{-49.541} + \alpha/2^{-1022} \leq 2^{-49.2999}. \end{aligned}$$

J. About $2^e \cdot h$ for $\rho_1 \leq r_h \leq \rho_2$.

We prove here that if $\rho_1 \leq r_h \leq \rho_2$, no underflow or overflow happens at the end of Algorithm exp_1. More precisely the approximation $2^e \cdot (h + \ell)$ of $\exp(r_h + r_\ell)$ obtained at line 25 of Algorithm exp_1 satisfies:

$$2^{-991} \leq 2^e \cdot (h + \ell) \leq \Omega,$$

moreover the same inequality holds for $2^e \cdot h$ alone, and in turn for $e_h = \circ(2^e \cdot h)$ since 2^{-991} and Ω are in $\mathbb{F}$.

In Appendices A-H and A-I we have analyzed Algorithms 7 and 8 (q_1 and exp_1) and shown that $|e_\ell/e_h| \leq 2^{-49.2999}$ and that the following relative error bounds hold:

$$\begin{aligned} \exp(r_h + r_\ell) &= 2^{k/2^{12}} \cdot \exp(z) \cdot (1 + \varepsilon), \\ p_h + p_\ell &= 2^{i_2/64 + i_1/2^{12}} \cdot (1 + \varepsilon'), \\ q_h + q_\ell &= \exp(z) \cdot (1 + \epsilon''), \\ h + \ell &= (p_h + p_\ell)(q_h + q_\ell)(1 + \varepsilon'''), \end{aligned}$$

where $|\varepsilon| \leq 2^{-64.67806}$, $|\varepsilon'| \leq 2^{-102.2248}$, $|\varepsilon''| \leq 2^{-64.902632}$, and $|\varepsilon'''| \leq 2^{-100.912696}$. Recalling that $k/2^{12} = e + i_2/64 + i_1/2^{12}$, we deduce that

$$2^e \cdot (h + \ell) = \exp(r_h + r_\ell) \cdot (1 + \delta), \qquad (2)$$

where

$$\delta := \frac{(1+\varepsilon')(1+\varepsilon'')(1+\varepsilon''')}{1+\varepsilon} - 1.$$

It can then be checked that $|\delta| \leq \phi := 2^{-63.78598}$.

The smallest possible value of $r_h + r_\ell$ is obtained for $r_h = \rho_1$ and r_ℓ the smallest value compatible with $|r_\ell/r_h| < 2^{-23.8899}$, namely $r_\ell =$ `-0x1.72b0feb06bbe9p-15`. For these values we get $\exp(r_h + r_\ell)(1 - \phi) > 2^{-991}$, which proves through Eq. (2) that $2^e \cdot (h + \ell) \geq 2^{-991}$. If $\ell \leq 0$, we have $2^e \cdot h \geq 2^e \cdot (h + \ell) \geq 2^{-991}$. If $\ell > 0$, we also know that $|\ell/h| \leq 2^{-49.541}$, thus $2^e \cdot h \geq 2^e \cdot (h + \ell)(1 - 2^{-49.541})$, which yields

$$2^e \cdot h \geq \exp(r_h + r_\ell)(1 + \phi)(1 - 2^{-49.541}).$$

Again with the smallest possible value of $r_h + r_\ell$, we get a right-hand side larger than 2^{-991}.

The largest possible value of $r_h + r_\ell$ is obtained for $r_h = \rho_2$ and r_ℓ the largest value compatible with $|r_\ell/r_h| < 2^{-23.8899}$, namely $r_\ell =$ `0x1.7f09093c9fe5bp-15`. For these values we get $\exp(r_h + r_\ell)(1 + \phi) < \Omega$, which proves that $2^e \cdot (h+\ell) \leq \Omega$. If $\ell \geq 0$, we have $2^e \cdot h \leq 2^e \cdot (h+\ell) \leq \Omega$. If $\ell < 0$, we also know that $|\ell/h| \leq 2^{-49.541}$, which yields

$$2^e \cdot h \leq 2^e \cdot (h + \ell)(1 + 2^{-49.541}/(1 - 2^{-49.541})).$$

Again with the largest possible value of $r_h + r_\ell$, we get a right-hand side smaller than Ω.

K. Proof of Lemma 7

We have seen in Appendix A-J that under the hypothesis of Lemma 7, the values $2^e \cdot h$ and $2^e(h+\ell)$ at the end of Algorithm exp_1 do not underflow nor overflow, and that the operation $e_h = \circ(2^e \cdot h)$ is exact.

Still in Appendix A-J, we have proven:

$$2^e \cdot (h + \ell) = \exp(r_h + r_\ell) \cdot (1 + \delta),$$

with $|\delta| \leq 2^{-63.78598}$.

However, we need a relative error bound in terms of $e_h + e_\ell$, not in terms of $2^e \cdot (h + \ell)$. While we know that $2^e \cdot h$ does not underflow nor overflow, thus $e_h = \circ(2^e \cdot h)$ is exact, and that $2^e \cdot \ell$ does not overflow, $2^e \cdot \ell$ might underflow, thus the operation $e_\ell = \circ(2^e \cdot \ell)$ at line 25 of Algorithm exp_1 might be inexact. It might give a maximal absolute error of α, and since $2^e \cdot h \geq 2^{-991}$, $|\ell/h| < 2^{-49.541}$, this corresponds to an additional relative error:

$$e_h + e_\ell = 2^e \cdot (h + \ell) \cdot (1 + \mu),$$

with $|\mu| < \alpha \cdot 2^{991}/(1 - 2^{-49.541}) \leq 2^{-82.9}$. Thus

$$e_h + e_\ell = \exp(r_h + r_\ell) \cdot (1 + \varepsilon_{\exp}),$$

with $\varepsilon_{\exp} := (1+\delta)(1+\mu)-1$. It can then be checked that $|\varepsilon_{\exp}| < 2^{-63.78597}$, and this concludes the proof of Lemma 7.

L. Proof of Theorem 1 when $r_h < \rho_1$.

This can happen for example for $x = \Omega$ and $y = -\Omega$, where we get $y \log x \approx -2^{1033.47}$. The threshold ρ_0 is such that Eq. (3) yields $x^y < \alpha/2$ for $r_h < \rho_0$ (since $y \log x$ is negative, we need a lower bound of $|y \log x|$). In such a case, the correct rounding is $+0$, except for rounding up where it is α. Note that in this case, yh is in the normal range. Indeed, if yh is in the subnormal range, i.e., $|yh| < 2^{-1022}$,

since $r_h = \circ(yh)$, we have $|r_h| \le 2^{-1022} + \alpha$, which is incompatible with $r_h < \rho_1 \approx -686.909$.

For $r_h < \rho_0$, Algorithm exp_1 returns $e_h = \alpha$ and $e_\ell = -\alpha$. For rounding to nearest, this yields $\circ(e_\ell \pm \varepsilon e_h) = -\alpha$, thus $u = v = +0$ (remember that $(+x) + (-x)$ is $+0$ for rounding to nearest). For rounding toward zero or upwards, this yields $\circ(e_\ell - \varepsilon e_h) = -\alpha$ and $\circ(e_\ell + \varepsilon e_h) = -0$, thus $u = +0$ and $v = \alpha$, and the rounding test fails. For rounding down, this yields $\circ(e_\ell - \varepsilon e_h) = -2\alpha$ and $\circ(e_\ell + \varepsilon e_h) = -\alpha$, thus $u = -\alpha$ and $v = -0$, and the rounding test fails. Thus for rounding to nearest, the correct rounding is returned, and for directed rounding modes, the correct rounding is delegated to the second phase.

The threshold ρ_1 is computed such that for $r_h \ge \rho_1$, the value $2^e \cdot h$ at the end of Algorithm exp_1 is larger or equal to 2^{-991}, thus $e_h = \circ(2^e \cdot h)$ does not underflow (see Appendix A-J). In the small range $\rho_0 \le r_h < \rho_1$, Algorithm exp_1 returns $e_h = e_\ell = \mathrm{NaN}$, and like in the case $\rho_2 < r_h \le \rho_3$, these corner cases are deferred to the second phase.

M. Proof of Theorem 1 when $\rho_2 < r_h$.

This can happen for example for $x = \Omega$ and $y = \Omega$, where we get $y \log x \approx 2^{1033.47}$. Note that in this case, yh is in the normal range. Indeed, if yh is in the subnormal range, i.e., $|yh| < 2^{-1022}$, since $r_h = \circ(yh)$, we have $|r_h| \le 2^{-1022} + \alpha$, which is incompatible with $r_h > \rho_2 \approx 709.78267$.

From Eq. (1), we have

$$|\log x| \ge |h + \ell|/(1 + \varepsilon_{\log}),$$

and since $|\ell| \le 2^{-23.89}|h|$ from Lemma 4,

$$|y \log x| \ge |yh|(1 - 2^{-23.89})/(1 + \varepsilon_{\log}).$$

Since $r_h = \circ(yh)$, we have $|yh| \ge |r_h|(1 - 2^{-52})$, thus:

$$|y \log x| \ge |r_h| \frac{(1 - 2^{-23.89})(1 - 2^{-52})}{1 + \varepsilon_{\log}}. \quad (3)$$

For $r_h > \rho_3$, we have $r_h \ge \rho_3 + 2^{-43}$, and it then follows from Eq. (3) that $x^y > 2^{1024}$, thus x^y overflows for all rounding modes.

In that case Algorithm exp_1 returns $e_h = e_\ell = \Omega$. In Algorithm phase_1, for rounding down or toward zero, this will yield $u = v = \Omega$, and since $u = v$ the algorithm returns $u = \Omega$, which is the correct rounding. For rounding up or to nearest, we obtain $u = v = +\infty$, and again since $u = v$ the algorithm returns $u = +\infty$ which is the correct rounding.

On the other hand, we also have:

$$|y \log x| \le |r_h| \frac{(1 + 2^{-23.89})(1 + 2^{-52})}{1 - \varepsilon_{\log}}, \quad (4)$$

and we can check that for $r_h \le \rho_2$, $x^y < \Omega$, thus no overflow happens.

The threshold ρ_2 is computed such that for $r_h \le \rho_2$, the value $2^e \cdot h$ at the end of Algorithm exp_1 is smaller than or equal to Ω, thus $e_h = \circ(2^e \cdot h)$ does not overflow (see Appendix A-J). In the small region $\rho_2 < r_h \le \rho_3$, Algorithm exp_1 returns $e_h = e_\ell = \mathrm{NaN}$, thus in Algorithm phase_1 we get $u = v = \mathrm{NaN}$, and by IEEE 754, the comparison $u = v$ is false, FAIL is returned, and these corner cases are deferred to the second phase.

N. Proof of Theorem 1 when $\rho_1 \le r_h \le \rho_2$.

Since we will use here Lemma 5, we have to check that the assumption $2^{-969} \le |yh| \le 709.7827$ from that lemma holds, or exclude the cases $|yh| < 2^{-969}$ and $709.7827 < |yh|$. The latter case is easy to exclude: if $|yh| > 709.7827$, since $r_h = \circ(yh)$, we deduce that $|r_h| \ge 709.7827 \cdot (1 - 2^{-52}) \ge 709.78269$, which is incompatible with $\rho_1 \le r_h \le \rho_2$. The case $|yh| < 2^{-969}$ is dealt with separately at the end of this subsection, where we show that Theorem 1 holds in this case too.

Step 1 of Algorithm phase_1 computes $h, \ell \leftarrow \log\_1(x)$. According to Lemma 4, we have $|\ell| \le 2^{-23.89}|h|$ and

$$\left| \frac{h + \ell}{\log x} - 1 \right| \le \varepsilon_{\log}$$

with $\varepsilon_{\log} = 2^{-73.527}$ if $x \notin (1/\sqrt{2}, \sqrt{2})$, and $\varepsilon_{\log} = 2^{-67.0544}$ otherwise.

Step 2 of Algorithm phase_1 computes $r_h, r_\ell \leftarrow \mathrm{mul\_1}(h, \ell, y)$. According to Lemma 5, since we assumed $2^{-969} \le |yh| \le 709.7827$, then $|r_h| \in [2^{-970}, 709.79]$, $|r_\ell| \le 2^{-14.4187}$, $|r_\ell / r_h| \le 2^{-23.8899}$, $|r_h + r_\ell| \le 709.79$, and

$$|r_h + r_\ell - y \log x| \le \varepsilon_{\text{mul}} \quad (5)$$

with $\varepsilon_{\text{mul}} = 2^{-63.799}$ if $x \notin (1/\sqrt{2}, \sqrt{2})$, and $\varepsilon_{\text{mul}} = 2^{-57.580}$ otherwise.

Step 3 of Algorithm phase_1 computes $e_h, e_\ell \leftarrow \mathrm{exp\_1}(r_h, r_\ell)$. The assumptions $|r_\ell / r_h| < 2^{-23.8899}$ and $|r_\ell| < 2^{-14.4187}$ of Lemma 7 are fulfilled (by Lemma 5), underflow and overflow cases are detected within Algorithm exp_1 (see Appendices A-L and A-M), thus Lemma 7 applies, and we have

$$\left| \frac{e_h + e_\ell}{\exp(r_h + r_\ell)} - 1 \right| < 2^{-63.78597}.$$

Let us define $r := r_h + r_\ell$. Thus

$$|e_h + e_\ell - e^r| < 2^{-63.78597} \cdot e^r. \quad (6)$$

By Eq. (5), we can write $r = y \log x + \sigma$ with $|\sigma| \le \varepsilon_{\text{mul}}$. Taking the exponential yields

$$e^r = x^y \cdot e^\sigma,$$

which gives

$$|e^r - x^y| \le |e^{-\sigma} - 1| \cdot e^r. \quad (7)$$

Combining Eq. (6) and Eq. (7) yields

$$|e_h + e_\ell - x^y| \le (2^{-63.78597} + |e^{-\sigma} - 1|) \cdot e^r.$$

The largest value of $|e^{-\sigma}-1|$ is obtained for $\sigma = -\varepsilon_{\text{mul}}$, and we obtain

$$|e_h + e_\ell - x^y| \le \mu \cdot e^r \tag{8}$$

with $\mu = 2^{-62.7924}$ if $x \notin (1/\sqrt{2}, \sqrt{2})$, and $\mu = 2^{-57.5605}$ otherwise. Now we want a bound in terms of e_h, e_ℓ, which are known, not in terms of e^r, which is unknown. Eq. (6) rewrites as $e^r < |e_h + e_\ell| + 2^{-63.78597} e^r$, which yields $e^r < 1/(1 - 2^{-63.78597}) \cdot |e_h + e_\ell|$. Moreover we want a bound in terms of e_h alone, since the left and right bounds u, v computed in steps 6-7 of Algorithm phase_1 use an error bound in terms of εe_h. Using the hypothesis $|e_\ell/e_h| \le 2^{-49.2999}$, it follows that

$$e^r < (1 + 2^{-49.2999})/(1 - 2^{-63.78597}) \cdot |e_h|.$$

Combining this inequality with Eq. (8), we obtain

$$|e_h + e_\ell - x^y| \le \tau \cdot e_h, \tag{9}$$

with $\tau = 2^{-62.7923}$ if $x \notin (1/\sqrt{2}, \sqrt{2})$, and $\tau = 2^{-57.5604}$ otherwise.

About the case $|yh| < 2^{-969}$. In this case we analyze separately Algorithms mul_1 and exp_1.

First in mul_1, since $|yh| < 2^{-969}$ and $r_h = \circ(yh)$, we have $|r_h| \le 2^{-969}$ since $2^{-969} \in \mathbb{F}$. Since r_h is the rounding of yh, by definition $|r_h - yh| < \text{ulp}(yh) \le 2^{-1022}$, thus $|s| \le 2^{-1022}$. Now since $|\ell| \le 2^{-23.89}|h|$, we have $|y\ell| < 2^{-992.89}$, thus $|y\ell + s| < 2^{-992.89} + 2^{-1022} \le 2^{-992.88}$, and $|r_\ell| \le 2^{-992.88} \cdot (1 + 2^{-52}) \le 2^{-992}$. In summary we have in this case $|r_h| \le 2^{-969}$ and $|r_\ell| \le 2^{-992}$.

Now if we analyze Algorithm exp_1 with such tiny values of r_h, r_ℓ, we see that $k = 0$, which yields $z_h = r_h$, $z_\ell = r_\ell$, $z = \circ(r_h + r_\ell)$ and thus $|z| \le (2^{-969} + 2^{-992})(1 + 2^{-52}) \le 2^{-968.9}$. In addition, $k = 0$ implies $e = i_2 = i_1 = 0$, $h_2 = h_1 = 1$, $\ell_2 = \ell_1 = 0$, $p_h = 1$, $s = 0$, $t = p_\ell = 0$, and at the end of the algorithm, $h = q_h$, $s = 0$, $t = 0$, $\ell = q_\ell$, thus $e_h = q_h$ and $e_\ell = q_\ell$.

It thus remains to analyze Algorithm q_1 for tiny inputs, namely, $|z| \le 2^{-968.9}$. First, $q = \circ(Q_4 z + Q_3)$ is rounded either to $Q_3 \approx 0.17$, or to one of the two adjacent binary64 numbers, thus $|q| \le 2^{-2}$. Similarly $q' = \circ(qz + Q_2)$ is rounded either to $Q_2 = 1/2$, or to one of the two adjacent binary64 numbers, thus $|q'| \le 1$. In $h_0 = \circ(q'z + Q_1)$, $q'z$ is so tiny that h_0 is either $Q_1 = 1$ or one of the two adjacent binary64 numbers, thus $|h_0| \le 2$. Then in $h_1, \ell_1 = \text{ExactMul}(z, h_0)$, we have $|h_1|, |\ell_1| \le 2^{-967.9}$. We thus perform the last operation $q_h, q_\ell = \text{FastSum}(Q_0, h_1, \ell_1)$, with $Q_0 = 1$ and $|h_1|, |\ell_1| \le 2^{-967.9}$. For rounding to nearest-even, it necessarily gives $q_h = 1$ and $|q_\ell| \le 2^{-965}$. Since we have $e_h = q_h = 1$ and $e_\ell = q_\ell$ at the end of exp_1, it is easy to check that the rounding test from Algorithm phase_1 will succeed, since $\circ(e_\ell \pm \varepsilon e_h)$ rounds to $\pm\varepsilon$, and $\circ(e_h \pm \varepsilon)$ rounds to 1, which is the expected correct rounding for $\exp(y \log x)$ with tiny $y \log x$. For directed roundings, we will either have $e_h = 1$ and tiny e_ℓ, or $e_h = 1 - 2^{-53}$ and $e_\ell \approx 2^{-53}$, or $e_h = 1 + 2^{-52}$ and $e_\ell \approx -2^{-52}$. In the first case ($e_h = 1$ and tiny e_ℓ), the rounding test will fail, since $\circ(e_\ell \pm \varepsilon e_h)$ rounds to about $\pm\varepsilon$, and $1 - \varepsilon$, $1 + \varepsilon$ round to different values. In the second case ($e_h = 1 - 2^{-53}$ and $e_\ell \approx 2^{-53}$), $\circ(e_\ell \pm \varepsilon e_h)$ rounds to about $2^{-53} \pm \varepsilon$, thus we have approximately $u = \circ(1 - \varepsilon)$ and $v = \circ(1 + \varepsilon)$, which again round to different values. The third case is similar.

Note that this reasoning also holds when $z = 0$, which can happen when $x = 1$, for example. In this case we have $q_h = 1, q_\ell = 0$, thus $e_h = 1, e_\ell = 0$, $\circ(e_\ell \pm \varepsilon e_h) = \pm\varepsilon$, and the rounding test simplifies to $\circ(1 - \varepsilon) = \circ(1 + \varepsilon)$, which only holds for rounding to nearest.

In summary, when $|yh| < 2^{-969}$, Algorithm phase_1 returns the correct rounding for rounding to nearest-even, and returns FAIL for directed rounding modes, thus Theorem 1 holds.

About the rounding test. We know from (9) that

$$e_h + e_\ell - \tau e_h \le x^y \le e_h + e_\ell + \tau e_h,$$

from which we deduce by monotonicity of rounding:

$$\circ(e_h + e_\ell - \tau e_h) \le \circ(x^y) \le \circ(e_h + e_\ell + \tau e_h).$$

But since there is no instruction that directly computes $\circ(e_h + e_\ell \pm \tau e_h)$, in Algorithm phase_1, we instead compute $u := \circ(e_h + \circ(e_\ell - \varepsilon e_h))$ and, similarly, $v := \circ(e_h + \circ(e_\ell + \varepsilon e_h))$. We thus have an additional rounding error in $\circ(e_\ell \pm \varepsilon e_h)$. Here, ε will be chosen slightly larger than τ (and thus a fortiori in $[\tau, 2\tau]$, say) in order to ensure that $u \le \circ(x^y) \le v$ despite these two extra rounding errors.

Let us now determine ε. It suffices to have $\circ(e_\ell - \varepsilon e_h) \le e_\ell - \tau e_h$ and $e_\ell + \tau e_h \le \circ(e_\ell + \varepsilon e_h)$, since by monotonicity of rounding, this will ensure that $u \le \circ(x^y) \le v$. Now, defining for simplicity $u' := e_\ell - \varepsilon e_h$ and $v' := e_\ell + \varepsilon e_h$, we have $\circ(u') \le u' + \text{ulp}(u')$ and $v' - \text{ulp}(v') \le \circ(v')$, and so it suffices that ε be such that $-\varepsilon e_h + \text{ulp}(u') \le -\tau e_h$ and $\tau e_h \le \varepsilon e_h - \text{ulp}(v')$. Now, $|u'|, |v'| \le |e_\ell| + \varepsilon e_h \le (2^{-49.2999} + 2\tau) e_h < 2^{-49.2905} e_h$ for $\varepsilon \le 2\tau \le 2 \cdot 2^{-57.5604}$. If $|u'| \ge 2^{-1022}$ then $\text{ulp}(u') = 2^{-52}|u'| \le 2^{-101.2905} e_h$; if $|u'| < 2^{-1022}$, then $\text{ulp}(u') = 2^{-1074} \le 2^{-83} e_h$, since $e_h \ge 2^{-991}$. Consequently, even if underflow occurs, we have $\text{ulp}(u') \le 2^{-83} e_h$ and, similarly, $\text{ulp}(v') \le 2^{-83} e_h$. Hence it suffices that ε be such that $\varepsilon \ge \tau + 2^{-83}$. It can then be checked that the values $\text{RU}(2^{-62.792})$ and $\text{RU}(2^{-57.560})$ taken for ε in Algorithm phase_1 satisfy this sufficient condition as well as the constraint $\varepsilon \le 2\tau$.