Finite Memory Arithmetic and the Number Representations on Computing Machines

Pavel Loskot
ZJU-UIUC Institute
Haining, China
e-mail: *pavelloskot@intl.zju.edu.cn*

***Abstract*—Numerical values on computing machines are normally represented as finite-memory data objects. Such values can be mapped to a finite set of integers, so the computing machines effectively perform only the integer arithmetic operations. Consequently, all programs and algorithms implemented on the computing machines can be modeled by Diophantine equations. This is also true for the computations that are performed by analog computers with a limited resolution. In order to study the Diophantine systems, this paper introduces a dual modulo operator to select the subsets of digits in the string representations of machine numbers, which is also useful when the number equality is replaced by a modulo equivalence. Moreover, it is shown that the solutions of Diophantine equations such as the Fermat Last Theorem can be obtained in the domain of integers that are offset by the same constant real value. The Fermat metric is newly introduced to define the distances between integers and other discrete sets of numbers. Finally, a two-dimensional quantization is devised for mixed arithmetic operations to allow the computations to be equivalently performed either between discrete analog values, or between the integer indices. The key claim of this paper is that all practical computing problems can be exactly and completely represented by an integer arithmetic.**

***Keywords*—dual modulo arithmetic; Fermat last theorem; Fermat metric; natural numbers.**

I. Introduction

Numbers are abstract mathematical objects that can also carry a semantic meaning of quantity. The former leads to rich axiomatic algebraic systems, and the latter enables performing arithmetic operations on computing machines. Numerical algorithms generate and transform numerical values by performing various arithmetic and non-arithmetic operations. Since the computing machines have limited resources, they must execute numerical algorithms in a time and memory efficient manner. The computed numerical values are stored as the precisely defined finite size objects in software and hardware, usually referred to as data structures. Consequently, these values cannot directly represent unconstrained real numbers in a mathematical sense, but it requires specifying a mapping from an infinite set of real numbers that are assumed in the definitions of computing problems into a finite set of selected discrete values that can be effectively represented in software and hardware as indicated in Figure 1.

The problem of representing numerical values on computing machines as integers when these values are stored in finite-memory data structures was introduced and examined in [1, SIGNAL'23]. The current paper extends our conference paper by adding more background knowledge, providing more comprehensive literature review as well as by presenting the new results on the quantization of real values and the related mixed arithmetic operations.

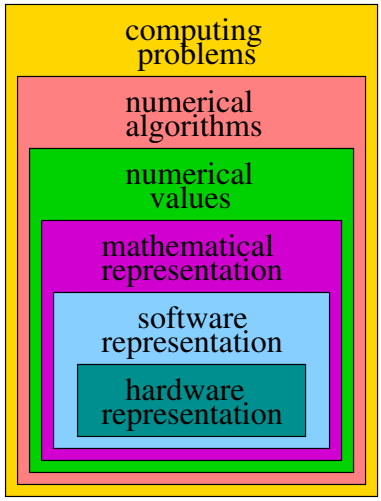

Figure 1. Embedded representations of numbers on computing machines.

Many programming languages used in computing applications such as Fortran, Matlab, Python, Julia and even C define integer and floating point numbers as their native data structures. Some of these languages allow further specifying the number of bytes used for storing the integer values, whether they are stored as signed or unsigned integers, and whether floating point numbers are stored with single or double precision. Furthermore, since the total number of unique values that can be represented by finite size data structures is finite, it limits the largest and the smallest numerical values as well as the precision that can be considered in any given computing application. For example, in Matlab, the functions, realmin, realmax, intmin, intmax and eps can be used to explore what numerical values are actually available to perform the computations.

Majority of real numbers are not computable [2], i.e., there is no algorithm that can express these numbers explicitly with an arbitrary precision (note that the constant π and $\sqrt{2}$ are both computable). On the other hand, a set of integers is said to be computably enumerable, if there exists an algorithm that

eventually lists all the elements in this set, even though such an algorithm may never halt.

Any algorithm described or implemented in any programming language can only compute numbers from a finite set, $\mathcal{N} = \{N_1 < N_2 < \ldots\}$, such that,

$$\forall i: \ -\infty < \inf(\mathcal{N}) \leq N_i \leq \sup(\mathcal{N}) < \infty. \tag{1}$$

Thus, the machine numbers, N_i, and, N_j, can be compared, i.e., ordered, and their smallest difference, $\min_{i\neq j}|N_i - N_j| = \varepsilon_0$, defines the precision. Moreover, the set, $\mathcal{N}$, is necessarily computable [3].

Most computing machines use floating point and fixed point number representations. These representations including the basic arithmetic operations are precisely defined by the IEEE 754 standard [4]. They enable efficient utilization of hardware and software resources to achieve time and space efficiency in implementing and executing the computing algorithms. Universal numbers (unums) were recently introduced as a superset of the IEEE-754 standard. However, their latest version known as posits is no longer the IEEE-754 format compatible, as they were primarily designed to mitigate the hardware constraints in representing the floating point numbers [5]. These modern number representations enable variable-width storage, support the interval (posit) arithmetic, and they also guarantee that the exact result of arithmetic operations is within the defined bounds.

The interval arithmetic assumes that numbers are represented by open or closed intervals rather than typical point values [6]. It then induces corresponding arithmetic operations between such intervals. For instance, if the two numbers have their values within the intervals, $[x_1, x_2]$, and, $[y_1, y_2]$, then their addition and multiplication is defined, respectively, as [7],

$$\begin{aligned} [x_1,x_2] + [y_1,y_2] &= [x_1+y_1, x_2+y_2] \\ [x_1,x_2] \cdot [y_1,y_2] &= [\min(x_1y_1, x_1y_2, x_2y_1, x_2y_2), \\ &\qquad \max(x_1y_1, x_1y_2, x_2y_1, x_2y_2)]. \end{aligned} \tag{2}$$

Similarly, the univariate function values of interval numbers are computed as,

$$f([x_1,x_2]) = [\min(f(x_1), f(x_2)), \max(f(x_1), f(x_2))]. \tag{3}$$

However, the interval arithmetic is not consider further in this paper, although it is an interesting and relevant topic for future investigations.

The multi-precision arithmetic can be implemented recursively by tracking a small number of the most recent digits of the arithmetic operands. Some languages (e.g., Python) support infinite-precision integer arithmetic, or perform the computations at the user-defined precision (e.g., Mathematica). The GNU library [8] is a popular and efficient implementation of the multi-precision arithmetic for integer and floating-point numbers in C programming language. This library is also used in several commercial software products (e.g., Mathematica and Maple). There is another the GNU library that is intended for multiple-precision floating point numbers [9], and another similar library for multiple-precision complex numbers [10]. Both these libraries are supported in all major programming languages. They focus on implementing the basic but fast arithmetic operations involving univariate polynomials and interpolations, however, without controlling the rounding of the results. The smallest and the largest integers and single and double precision floating point numbers are defined in the Matlab toolbox Elementary Matrices, and in the C standard libraries limits.h and float.h.

In general, the algorithms described in various programming languages represent the numbers as strings of digits in a given basis. In particular, the number, $N \in \mathcal{N}$, in basis, B, is represented as,

$$\begin{aligned} N &= \sum_{i=i_{\min}}^{i_{\max}} D_i \times B^i \\ &\leftrightarrow \quad D_{i_{\max}} D_{i_{\max}-1} \cdots D_1 D_0 \,.\, D_{-1} \cdots D_{i_{\min}} \end{aligned} \tag{4}$$

where the digits, $D_i \in \{0, 1, \ldots, 9, \mathrm{A}, \mathrm{B}, \mathrm{C}, \ldots B-1\}$, and the orders, $i_{\min} \leq 0 \leq i_{\max}$. Furthermore, it is customary to place a decimal point between the digits, D_0, and, D_{-1}, which divides the digits into an integral and a fractional part, respectively. More importantly, the decimal point has a purely syntactical meaning to align the numbers in the arithmetic operations and comparisons.

In programming and computing applications, the most common bases are decimal ($B = 10$), hexadecimal ($B = 16$), and binary ($B = 2$). However, internally, the numbers are stored much more efficiently in a byte-size oriented basis, i.e., $B = 2^{8\times\#\text{bytes}-1}$, with one bit reserved for a sign to indicate whether the number is negative. The total number of bytes used for each numerical value is usually fixed for different classes (types) of numbers including short and long integers, and single and double precision floating point numbers. The conversions between the string notation and the internal representation are performed automatically by the compiler.

The textbook [4] provides a comprehensive overview of the number systems that are used on computers. The computability of functions of natural numbers is established in [11]. The mismatch between the exact mathematical description and practical implementation of algorithms using the approximate number representations has been studied in [12] including the methods how to mitigate the discrepancies. A construction of the large-scale real numbers, which are suitable for software implementation is considered in [13]. The binary approximations of real-numbers are investigated in [14]. Other representations of real numbers such as binary expansions, Dedekind cuts and Cauchy sequences are compared in [3]. The p-adic number systems allow defining real-numbers as the arithmetic of rational numbers [15]. The logical statements involving comparisons of real-numbers are studied with the help of satisfiability modulo theories in [16]. The article [17] argues that finite precision is often sufficient in many practical engineering applications. The most important number-theoretic theorems and conjectures can be found in [18].

In this paper, it is argued that the number systems commonly used on computers can be assumed to be integer-valued,

which also includes single and double precision floating point numbers and the corresponding arithmetic operations. Consequently, the computing machines are inherently governed by integer algebras and arithmetic. The paper contributions are formulated as three claims, a proposition and several lemmas. In particular, in Section II, dual modulo operator is introduced to select or to discard the digits in the string representations of numbers. It can be exploited to define equivalences between numbers in integer arithmetic. In addition, it is proposed that natural numbers can be offset by a real-valued constant, and still be considered as being integers. The Fermat last theorem (FLT) is studied in Section III as an example of Diophantine equation involving integers. The Fermat metric is newly defined, which is then used to compute the distances between natural numbers, and to cluster natural numbers into subsets. A new Section IV has been added to devise a quantization scheme that supports arithmetic operations. Discussion is provided in Section V, and the paper is concluded in Section VI.

II. Machine Integers and Arithmetic

The key observation is that finite-size data structures can only represent a finite set of numerical values. Constraining machine computations to the numbers, $\mathcal{N}$, has several fundamental consequences. First, the results of arithmetic operations can overflow the limits, $\inf(\mathcal{N})$, or, $\sup(\mathcal{N})$. Second, the results of arithmetic operations can underflow the precision, ε_0, so the results may have to be truncated, rounded, or otherwise approximated. Third, the decimal point to align the numbers can be arbitrarily placed in-between any digits as long as the placement is consistent in the number system and the arithmetic considered. This is formalized in the following claim.

Claim 1. *The machine numbers that are allocated a finite memory space can be uniquely represented by a set, $\mathcal{N}$, which is isomorphic to a finite ordered subset of integers, $\mathcal{Z} \subseteq \mathbb{Z}$.*

The important consequence is that (without a formal proof) any machine arithmetic is isomorphic to the integer arithmetic. However, implementing such a integer arithmetic at large scale and precision to be efficient and also error-free is non-trivial. It raises important questions about how to optimally represent mathematical models on computers under the number representation constraints as well as how to model the underlying computations.

The memory allocated by the compilers of programming languages allows adding only a finite number of digits before and after the decimal point. If the numbers are padded by zero-digits from both ends, the numbers are represented by the strings of the same length, and the decimal point becomes a hypothetical construct. The non-zero digits at the right end of the number string represent the precision (resolution), whereas the first non-zero digits from the left represent the scale.

The algorithms usually contain many logical statements (predicates). These statements involve comparisons of numerical values. Even analog computers can compare numerical values, e.g., against a threshold, only with a certain resolution. In general, any two numerical values can only be compared reliably, if they are represented with an infinite precision. On the other hand, two integers are said to be exactly equal, provided that all digits in their string representations are the same. The exact comparison can be rather restrictive in some applications, where the differences in scale and precision could be or must be tolerated. Specifically, if the differences are tolerated in precision (the right-end sub-strings), it is equivalent to comparing the quantized values. On the other hand, if the differences are tolerated at scale (the left-end sub-strings), it induces the periodicity, and it is equivalent to comparing the periodically repeated values.

Mathematically, removing the right-end or the left-end sub-strings from the string representation of a number can be expressed by a canonical modulo operator. In particular, for any integer a, and any positive integer b, let, $(a \bmod b) = (|a| \bmod b) \in \{0,1,\ldots,b-1\}$, be a reminder after the integer division of a by b. Note that this can be readily extended to the real numbers as, $0 \leq (a \bmod b) = (|a| \bmod b) < b$, assuming the real division of, $a \in \mathcal{R}$, by an integer, b. Then, the numbers, a_1, and, a_2, are said to be equivalent in the sense of congruence, provided that, $a_1 \equiv a_2 \ (\bmod b)$. Both equality (indicated by the symbol, $=$) and equivalence (indicated by the symbol, $\equiv$) satisfy the axiomatic properties of reflexivity, symmetry, and transitivity, and the equality implies the equivalence. Moreover, if the two numbers are not congruent after comparing their modulo values, then these numbers are not equal even without using modulo operator.

If the machine numbers, $N_i = \sum_{i=0}^{L-1} D_i B^i$, are represented by the strings of L digits in some basis B, then the first L_1 digits and the last L_2 digits, $(L_1 + L_2) < L$, can be zeroed by applying a dual modulo operator, which is introduced next.

Definition 1. *The dual modulo operator has two parameters, m_1, and, m_2, and it is defined as the difference,*

$$\begin{aligned} N_i \ \mathrm{Mod}(m_1, m_2) &= (N_i \bmod m_1) - (N_i \bmod m_2) \\ &= \underbrace{0\cdots0}_{L_1} D_{L-L_1-1} \cdots D_{L_2+1} D_{L_2} \underbrace{0\cdots0}_{L_2}. \end{aligned} \tag{5}$$

where $m_1 = B^{L-L_1}$, and $m_2 = B^{L_2}$.

It should be noted that representing the numbers in different bases does not change their semantic meaning. There is a one-to-one mapping between different representations, so they are mathematically equivalent. However, they are not equivalent in the string operations.

The modular arithmetic with dual modulo operator has similar properties as the arithmetic involving canonical modulo operator. In particular, given the integers, a, b, m_1, and m_2, then,

$$\begin{aligned} a \ \mathrm{Mod}(0, m_2) &= a - (a \bmod m_2) \\ a \ \mathrm{Mod}(m_1, 1) &= a \bmod m_1 \\ a \ \mathrm{Mod}(m_1, m_1) &= 0. \end{aligned} \tag{6}$$

Furthermore, it is straightforward to prove that,

$$\begin{aligned} a+b &\equiv a \text{ Mod}(m_1,m_2)+b \text{ Mod}(m_1,m_2) \text{ (Mod}(m_1,m_2)) \\ a-b &\equiv a \text{ Mod}(m_1,m_2)-b \text{ Mod}(m_1,m_2) \text{ (Mod}(m_1,m_2)) \\ a\cdot b &\equiv a \text{ Mod}(m_1,m_2)\cdot b \text{ Mod}(m_1,m_2) \text{ (Mod}(m_1,m_2)). \end{aligned} \tag{7}$$

However, in general, assuming the integer division with a reminder, one has,

$$a/b \not\equiv a \text{ Mod}(m_1,m_2)/b \text{ Mod}(m_1,m_2) \text{ (Mod}(m_1,m_2)). \tag{8}$$

The Chinese reminder theorem [18] can be restated for the dual modulo operator as follows. If m_{11} and m_{12} are co-prime, and,

$$\begin{aligned} N_i &\equiv a_1 \text{ (Mod}(m_{11},m_2)) \\ N_i &\equiv a_2 \text{ (Mod}(m_{12},m_2)) \end{aligned} \tag{9}$$

for some integers, N_i, and, m_2, then there is a unique integer, a, such that,

$$N_i \equiv a \text{ (Mod}(m_{11}m_{12},m_2)). \tag{10}$$

The proof is based on the property that, if $N_i \equiv a \pmod{m_1}$, then also, $N_i \equiv a$ (Mod(m_1,m_2)).

It is useful to consider how the machine integers used in algorithms are the approximations of infinite precision real-numbers that are normally obtained from the mathematical analysis. The dual modulo operator defined in (5) produces a finite-length integer, x Mod(m_1,m_2), from a real number, $x \in \mathcal{R}$. This introduces a periodicity due to truncation from the left (specified by the parameter, m_1), and the quantization due to truncation from the right (specified by the parameter, m_2).

Furthermore, define countably infinite integer sets,

$$\tilde{\mathbb{N}}_x = \{x, x+1, x+2, \ldots\} \tag{11}$$

which are parameterized by a finite real-valued constant, $x \in \mathcal{R}$, so that, $\tilde{\mathbb{N}}_0$, is the set of natural numbers. Such integer sets can provide the exact solutions to some integer (Diophantine) problems, which otherwise do not have any such a solution. More importantly, for all finite x, the integers, $\tilde{\mathbb{N}}_x$, satisfy Peano axioms except the first one, i.e., that the first element in the integer set represents a zero [18]. Peano arithmetic includes both addition and multiplication, and it is known to be incomplete and undecidable whilst its consistency cannot be proven as can be shown by the Gödel's incompleteness theorems. However, when multiplication is dropped, Peano arithmetic becomes Presburger arithmetic, which is consistent, complete and decidable. More importantly, both Peano as well as much weaker Presburger arithmetic can be used to define the set of natural integers.

III. Case Study: FLT Problems

The FLT is the most famous special case of a Diophantine equation. It is considered here for illustration of the effects of the number representations used on computing machines. The FLT states that there are no positive integers a, b, c, and $n > 2$, such that,

$$a^n + b^n = c^n. \tag{12}$$

This has been first verified numerically on computers for very large exponents until a formal mathematical proof was established only recently [18]. It is straightforward to show that the case of coefficients, a, b, and c being rational numbers (i.e., they are expressed as the integer fractions) is only a special case of the more general FLT. This is also the case when the exponent, n, is a rational number, and so neither this special case have any solution among the integer coefficients. Specifically, if $n = m/(m+k)$, where m and k are integers, $k \neq -m$, we get, $a^k + b^k = (ab/c)^k$, i.e., a special case of the FLT. Interestingly, the FLT can be equivalently rewritten as a condition for the arithmetic average of the integer products, i.e.,

$$a^n b^n c^n = \frac{1}{2}\left((a^n+1)(b^n+1)(c^n-1)+(a^n-1)(b^n-1)(c^n+1)\right). \tag{13}$$

Another equivalent formulation of the FLT involves the infinite sets, $S_1 = \{a^n + b^n : \ a,b \in \mathbb{N}_0\}$, and, $S_2 = \{c^n : \ c \in \mathbb{N}_0\}$; the FLT then states that the intersection, $S_1 \cap S_2 = \emptyset$ (empty set), for all the exponents, $n > 2$.

In addition, constraining the maximum difference as, $|a^n + b^n - c^n| \leq 1$, has a trivial solution, $a = 1$, and, $b = c$, for $\forall n \geq 1$. Note also that the Fermat Number Transform (FNT) resembles the Discrete Fourier Transform (DFT), however, the former assumes the sums modulo a prime [19].

More importantly, the original formulation of the FLT can be modified as follows to allow the solutions to exist.

Claim 2. *For every n, there exist infinitely many natural integers a, b, c, m_1 and m_2 satisfying the congruence,*

$$a^n + b^n \equiv c^n \text{ (Mod}(m_1,m_2)). \tag{14}$$

For example, assuming the first 100 natural numbers represented as the strings of $l = 9$ digits in the basis, $B = 8$, and, $B = 10$, respectively, the total number of solutions, n_l, and, n_r, of (14), for the first l_1 digits, and for the last $l_2 = l - l_1$ digits is given in Table I. We can observe that, always, $n_\text{l} > n_\text{r}$, since the number strings often contain zeros at the left end in order to make up the given string width, l.

TABLE I. The number of solutions of (14) among the first 100 integers

	$B=8$				$B=10$			
	$n=3$		$n=4$		$n=3$		$n=4$	
(l_1,l_2)	(3,6)	(4,5)	(3,6)	(4,5)	(3,6)	(4,5)	(3,6)	(4,5)
n_l	69627	22278	5505	2318	1284	44532	10666	3622
n_r	212	644	730	2076	198	207	230	596

The following FLT formulation utilizes the integers with a fixed offset that were defined in (11).

Claim 3. *For any integer exponent, $n \geq 1$, the equation,*

$$a^n + b^n = c^n \tag{15}$$

has infinitely many solutions among the integers, $a,b,c \in \cup_x \tilde{\mathbb{N}}_x$, for specific real-values, $x > 0$.

Proof. Let $c = y \in \mathcal{R}$, $a = y - d_1$, and, $b = y - d_2$, where d_1 and d_2 are the arbitrarily chosen positive natural integers. Then, for any n, the polynomial (15) has at least one real-valued solution, $y > \max(d_1, d_2)$. Let $d_0 = \lfloor y \rfloor$ (floor function), so that $x = (y - d_0) < 1$. This defines the positive integer solution, $c = d_0 + x$, $a = d_0 - d_1 + x$, and $b = d_0 - d_2 + x$, from the integer set, $\tilde{\mathbb{N}}_x$. ■

The Euler's conjecture from the year 1769 states that the n-th power of a positive integer cannot be expressed as a sum of a fewer than n, n-th powers of other integers. This has been disproved in 1966 by finding the counterexample, $27^5 + 84^5 + 110^5 + 133^5 = 144^5$, using a computer search [20]. Another counterexample, $2682440^4 + 15365639^4 + 18796760^4 = 20615673^4$, was obtained in [21] using theoretical methods. More precise formulation of determining the number of m-th power summands is known as the Waring's problem [22]. The further generalization of the Euler-Fermat equation was examined in [23] by assuming a weighted sum of the m-th power summands with the integer weights. Another variation of the Euler's sum was recently considered in [24].

A weaker form of the Euler's conjecture can be stated as follows.

Proposition 1. *For any natural integer, n, there exists an integer, $m \geq n$, such that the expression,*

$$\sum_{i=1}^{m} a_i^n = b^n \tag{16}$$

is satisfied for a set of natural integers, $\{a_1, a_2, \ldots, a_m\} \cup \{b\}$.

The proof of Proposition 1 appears to be rather non-trivial, except when $n = 1$ and $n = 2$ (Pythagorean theorem); the interested readers are referred to [21]–[24] for more in-depth investigations of this problem. However, it is easy to find examples among the smallest integers satisfying expression (16), i.e.,

$$\begin{aligned} 3^2 + 4^2 &= 5^2 \ \ (m = n = 2) \\ 3^3 + 4^3 + 5^3 &= 6^3 \ \ (m = n = 3) \\ 2^4 + 2^4 + 3^4 + 4^4 + 4^4 &= 5^4 \ \ (m = n + 1 = 5) \\ 19^5 + 43^5 + 46^5 + 47^5 + 67^5 &= 72^5 \ \ (m = n = 5). \end{aligned} \tag{17}$$

In general, the sequence, $a^n + b^n$, obtained by enumerating all natural integers, a, and, b, becomes rapidly very sparse as the exponent, n, is increased. Moreover, given n, it is easy to show that the best approximation of $(a^n + b^n)$ by c^n is obtained for, $c = \lfloor (a^n + b^n)^{1/n} \rceil$ (rounding function). It motivates the following distance metric for the pairs of natural integers.

Definition 2. *The Fermat metric for positive numbers, a, and, b, is computed as,*

$$\mathcal{F}_n(a,b) = a^n + b^n - \lfloor (a^n + b^n)^{1/n} \rceil^n \tag{18}$$

where $n = 2, 3, \ldots$ is a natural number, and, $\mathcal{F}_1(a,b) = 0$, for any a and b. The Fermat distance between the numbers, a, and, b, is then the absolute value of the Fermat metric, i.e.,

$$D_n(a,b) = |\mathcal{F}_n(a,b)|. \tag{19}$$

Note that rounding in (18) can be replaced with the floor function or the ceil function to obtain the Fermat metric values that are always positive or always negative, respectively. The floor and ceil functions can be also used to find the largest integer, $c_{\max}$, and the smallest integer, $c_{\min}$, respectively, that are bounded as, $c_{\max}^n < a^n + b^n < c_{\min}^n$, for $n > 2$.

The distributions of the Fermat metric values by enumerating all pairs of natural integers up to 10^5 are shown in Figure 2, for $n = 2$ and $n = 3$, respectively. It can be observed that the Fermat metric values are spread much more evenly for $n = 2$, and these distributions have no obvious symmetry.

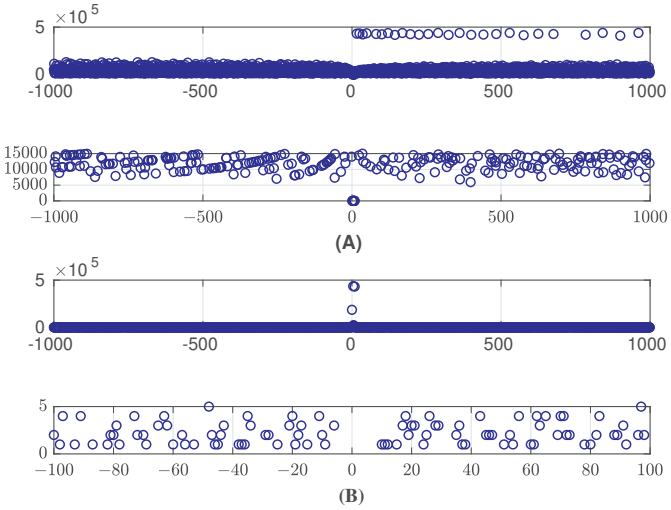

Figure 2. The counts of the Fermat distance values for all pairs of natural integers up to 10^5, and the exponent $n = 2$ (A), and $n = 3$ (B).

The Fermat distance can be used to cluster natural integers into subsets. Figure 3 shows a dendrogram of the Fermat distances, $\mathcal{F}_2(a,b)$. The corresponding assignment of the first 50 natural numbers into the four subsets based on the distances, D_2, D_3, D_4, and D_5 are then shown in Figure 4.

IV. Case Study: Mixed Arithmetic Aided Quantization

The unconstrained real-valued numbers that are used in mathematically defined computing models must be eventually represented as numerical values that are constrained by the representations in software and hardware. As discussed above, the numerical values that are represented as finite-memory data objects can be equivalently considered to represent finite sets of integers. The process of approximating continuous real values by discrete values is referred to as quantization. The normally assumed equidistant quantization levels may become problematic, when the number of quantization levels that can be represented by the finite-size data structures is relatively small. A common solution is to scale-down the real values,

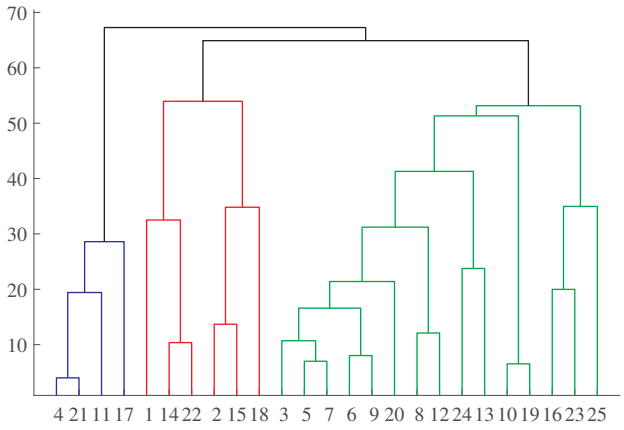

Figure 3. The dendrogram of natural numbers constructed assuming the Fermat distance, $D_2(a,b)$, defined in (19).

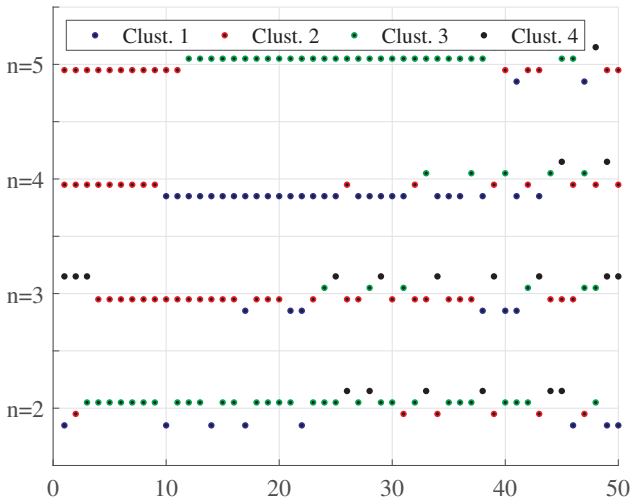

Figure 4. The first 50 natural numbers partitioned into four clusters (subsets) using the Fermat distances, $D_n(a,b)$, for $n = 2,3,4$, and 5.

and to limit these values to a finite-length interval prior to quantization. However, this reduces the resolution as well as the dynamic range that can be represented by the quantized values. Here, we consider designing a quantization scheme that is constrained by the modular and mixed arithmetic in order to obtain a non-uniform quantization with more effective mapping of real numbers to unique positive integers.

In particular, any real number, $v \in \mathcal{R}$, can be expressed as,

$$\begin{aligned} v &= q \cdot w + r, \\ r &= v \bmod w \end{aligned} \tag{20}$$

where the quotient, $q \in \mathbb{Z}$, and, $-w < r < w$, denotes the reminder for the chosen real positive parameter, $w \in \mathcal{R}^+$. Since q can be stored as a signed integer, or it can be completely dropped in some applications, our task is to define the quantization function as the mapping,

$$Q: \ [-w,w] \mapsto \{r_0, r_1, \ldots, r_K\} \tag{21}$$

which maps the reminder, r, to a unique value from the set of $(K+1)$ values, such that,

$$-w \le r_0 < r_1 < \cdots < r_K \le w. \tag{22}$$

It is customary to select the quantized values that minimize the Euclidean distance to the input sample, r, i.e.,

$$r_k = \text{argmin}_{0 \le k \le K} \|r - r_k\|. \tag{23}$$

The quantization levels, r_k, should be defined as an increasing function of the quantization labels, k, i.e., $r_{k_1} < r_{k_2}$, if and only if, $k_1 < k_2$. Furthermore, it is desirable to control the non-uniformity of the quantization levels, and to also compute the index, k, from the quantized value, r_k. These requirements can be satisfied by an n-th order polynomial having a single root with multiplicity, n. Hence, define,

$$r_k = \mathcal{J}_{a,z,n}(k) = a(k+z)^n \tag{24}$$

where $a > 0$ and $z \in \mathcal{R}$ are the real constants. Note that the function, $\mathcal{J}_{a,z,n}(k)$, has an even symmetry about the vertical axis at the point, $k = -z$, if k is even, and it has an odd symmetry about the point, $k = -z$, if k is odd. Moreover, for any n, the minimum of $|\mathcal{J}_{a,z,n}|$ occurs at $k = -z$.

In addition, function (24) is invertible, i.e.,

$$k = \mathcal{J}^{(-1)}_{a,z,n}(r_k) = \begin{cases} \pm\sqrt[n]{r_k/a} - z, & r_k > 0, \ n - \text{even} \\ \sqrt[n]{r_k/a} - z, & n - \text{odd}. \end{cases} \tag{25}$$

More importantly, the quantization can be further constrained to achieve the equivalence between the arithmetic of real quantization values, r_k, and the arithmetic of corresponding quantization indices, k. In particular, provided that,

$$r_{k_1} \bigcirc r_{k_2} = r_{k_3} \tag{26}$$

where $\bigcirc$ denotes a binary arithmetic operation, such as addition, subtraction, multiplication or division, it is required that the corresponding quantization indices satisfy another arithmetic relation (in general, different from the one used for quantization values above), i.e.,

$$k_1 \ \square \ k_2 = k_3. \tag{27}$$

For instance, assuming binary addition for both the quantized values and the corresponding quantization indices, it is required that,

$$\begin{aligned} r_{k_1} + r_{k_2} &= r_{k_3} \\ \mathcal{J}_{a,z,n}(k_1) + \mathcal{J}_{a,z,n}(k_2) &= \mathcal{J}_{a,z,n}(k_3) \\ a(k_1+z)^n + a(k_2+z)^n &= a(k_3+z)^n. \end{aligned} \tag{28}$$

The solution of (28) is formalized by the following lemma.

Lemma 1. *For any integers, $k_1, k_2, k_3 \in \mathbb{Z}$, eq. (28) has exactly one real solution in z, if n is odd, and there are exactly two real solutions, when n is even.*

Proof. For addition, the multiplicative constant, $a > 0$, can be eliminated, and we need to solve, $(k_1+z)^n + (k_2+z)^n = (k_3+z)^n$, for z. The function, $\mathcal{J}_{a,z,n}(k)$, defined in (24) is strictly convex for n being even, and non-negative for any k and z. The

sum of convex functions remains convex [25]. The minimum of the left-hand side addition occurs for $z = -(k_1 + k_2)/2$, which is independent of n. It is then straightforward to show that any two convex functions, which have an unlimited support intersect at exactly two points. On the other hand, the function, $\mathcal{J}_{a,z,n}(k)$, for n-odd, is strictly increasing everywhere, and there is exactly one point where two such functions with an infinite support intersect. The solution can be obtained in a closed form for $n \leq 3$, otherwise it can be computed numerically. ■

Note that the scaling by a used in (28) affects numerical accuracy of the solution, especially for larger values of n. Moreover, $z = -k_2$ is the solution of (28), if and only if, $k_1 = k_3$, for any exponent, n. In addition, provided that the quantization values were defined as, $\mathcal{J}_{a,z,n} = k^n - b$, where $b \in \mathbb{Z}$, then eq. (28) becomes, $k_1^n + k_2^n = k_3^n + b$, and it can be satisfied for any integer coefficients k_1, k_2 and k_3, by appropriate choice of b.

Assuming multiplication instead of summation in (28), we need to solve,

$$a^2(k_1 + z)^n(k_2 + z)^n = a(k_3 + z)^n \tag{29}$$

for z, which is formalized by the next lemma.

Lemma 2. *For any integers, $k_1, k_2, k_3 \in \mathbb{Z}$, eq.* (29) *has exactly two real solutions,*

$$z = \frac{1}{2a}\Big(1 - ak_1 - ak_2 \pm \sqrt{(1 - ak_1 - ak_2)^2 - 4(ak_1ak_2 - ak_3)}\Big) \tag{30}$$

provided that, $(1 - ak_1 - ak_2)^2 > 4(ak_1ak_2 - ak_3)$, and n is odd. If n is even, there are two more solutions,

$$z = \frac{-1}{2a}\Big(1 + ak_1 + ak_2 \pm \sqrt{(1 + ak_1 + ak_2)^2 - 4(ak_1ak_2 + ak_3)}\Big) \tag{31}$$

provided that, $(1 + ak_1 + ak_2)^2 > 4(ak_1ak_2 + ak_3)$.

Proof. Eq. (29) can be converted to a quadratic equation, for which the solution is well known. ■

The properties stated in Lemma 1 and Lemma 2 can be exploited to define the mixed arithmetic, such that binary computations between analog quantized values can be equivalently performed as binary computations between the corresponding discrete indices as illustrated in Figure 5. It may be desirable, but not necessary to assume that the same binary operation is used in both the analog and the digital domains. For example, the quantization levels can be defined, so that the sum, $r_1 + r_2 = r_3$, in the analog domain is fully equivalent to computing the sum, $k_1 + k_2 = k_3$, in the discrete domain. This is formalized by the following lemma.

Lemma 3. *Consider a binary arithmetic operation, $k_3 = k_1 \square\ k_2$, such as addition, subtraction, multiplication, or division. Then, for any two integers k_1 and k_2 from the set, $\{0, 1, 2, \ldots, K\}$, and a given exponent, n, there exist, $z \in \mathcal{R}$, and, $a \in \mathcal{R}^+$, so that, a binary computation, $\mathcal{J}_{a,z,n}(k_1) \bigcirc \mathcal{J}_{a,z,n}(k_2) = \mathcal{J}_{a,z,n}(k_3)$, is exact, where the function, $\mathcal{J}_{a,z,n}$, has been defined in* (24).

Proof. The key result for addition and multiplication has been proven in Lemma 1 and Lemma 2, respectively. However, now, the proof is further constrained by requiring that $k_3 = k_1 \square\ k_2$. The cases of subtraction and division can be converted to addition and multiplication, respectively; for the latter, one should consider, $k_3 = k_1/(1 + k_2)$, in order to avoid division by zero, when $k_2 = 0$. ■

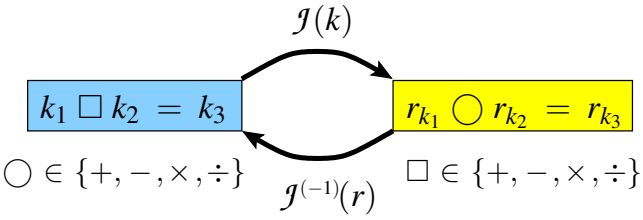

Figure 5. Binary computations for discrete indices (on the left) can be uniquely mapped to binary computations between the corresponding quantized values (on the right).

In practice, it is useful to compute a look-up table of z values that are indexed by the integer tuples, (k_1, k_2) as shown in Table II and Table III assuming binary operations of addition and multiplication, respectively. For example, if $k_1 = 1$ and $k_2 = 3$, then $k_3 = 1 + 3 = 4$, and we can numerically compute, $z \doteq 5.01$; then, $a(k_1 + z)^3 \doteq 435.78$, $a(k_2 + z)^3 \doteq 1030.72$, and $a(k_3 + z)^3 \doteq 1466.50 = 435.78 + 1030.72$. This calculation can be verified for all other rows in Table II and Table III.

More importantly, the quantization levels generated by function (24) are symmetric as shown in the next lemma. This can be used to calculate the quantized values also for negative indices.

Lemma 4. *The addition of quantized values has the following symmetry:*

$$\mathcal{J}_{a,-z,n}(-k_1) + \mathcal{J}_{a,-z,n}(-k_2) = \begin{cases} \mathcal{J}_{a,z,n}(k_1) + \mathcal{J}_{a,z,n}(k_2), & n - even \\ -\mathcal{J}_{a,z,n}(k_1) - \mathcal{J}_{a,z,n}(k_2), & n - odd. \end{cases} \tag{32}$$

Proof. The property is a direct consequence of the even or odd symmetry of function $\mathcal{J}_{a,z,n}$ defined in (24), for n being even or odd, respectively. ■

It should be emphasized that the calculated values of z are functions of the indices k_1 and k_2 as well as of the binary operations considered. The real-valued offsets, z, of the integer indices in the analog domain are necessary in order to enable a one-to-one mapping between the binary calculations in the discrete and the analog domains, respectively. Consequently, such a mapping defines a two-dimensional non-uniform vector quantization as indicated in Figure 6 assuming the examples

TABLE II. The index addition, $k_1 + k_2$, corresponding to the analog summation, $r_1 + r_2 = \mathcal{J}_z(k_1) + \mathcal{J}_z(k_2)$, for $a = 2.0$ and $n = 3$

k_1	k_2	r_1	r_2	$r_1 + r_2$	z
0	0	0	0	0	0
0	1	0.00	2.00	2.00	0.00
0	2	−0.00	16.00	16.00	−0.00
0	3	−0.00	54.00	54.00	−0.00
1	0	2.00	0.00	2.00	0.00
1	1	113.89	113.89	227.79	2.84
1	2	258.27	443.89	702.16	4.05
1	3	435.78	1030.72	1466.50	5.01
2	0	16.00	−0.00	16.00	−0.00
2	1	443.89	258.27	702.16	4.05
2	2	911.16	911.16	1822.32	5.69
2	3	1453.61	1994.59	3448.20	6.99
3	0	54.00	−0.00	54.00	−0.00
3	1	1030.72	435.78	1466.50	5.01
3	2	1994.59	1453.61	3448.20	6.99
3	3	3075.17	3075.17	6150.34	8.54

TABLE III. The index multiplication, $k_1 \cdot k_2$, corresponding to the analog multiplication, $r_1 \cdot r_2 = \mathcal{J}_z(k_1) \cdot \mathcal{J}_z(k_2)$, when $a = 2.0$ and $n = 3$

k_1	k_2	r_1	r_2	$r_1 \cdot r_2$	z
0	0	1.00	1.00	1.00	+0.79
0	1	−0.01	1.22	−0.01	−0.15
0	2	−0.00	16.00	−0.00	−0.00
0	3	−0.00	54.00	−0.00	−0.00
1	0	1.22	−0.01	−0.01	−0.15
1	1	1.00	1.00	1.00	−0.20
1	2	1.00	11.54	11.54	−0.20
1	3	1.00	43.60	43.60	−0.20
2	0	16.00	−0.00	−0.00	−0.00
2	1	11.54	1.00	11.54	−0.20
2	2	10.13	10.13	102.77	−0.28
2	3	9.51	38.56	366.81	−0.31
3	0	54.00	−0.00	−0.00	−0.00
3	1	43.60	1.00	43.60	−0.20
3	2	38.56	9.51	366.81	−0.31
3	3	35.75	35.75	1278.73	−0.38

in Table II and Table III. For $n = 1$, the quantized values are spaced uniformly, and they become more expanded when n is increased. The quantization values can be also scaled by adjusting the parameter, a.

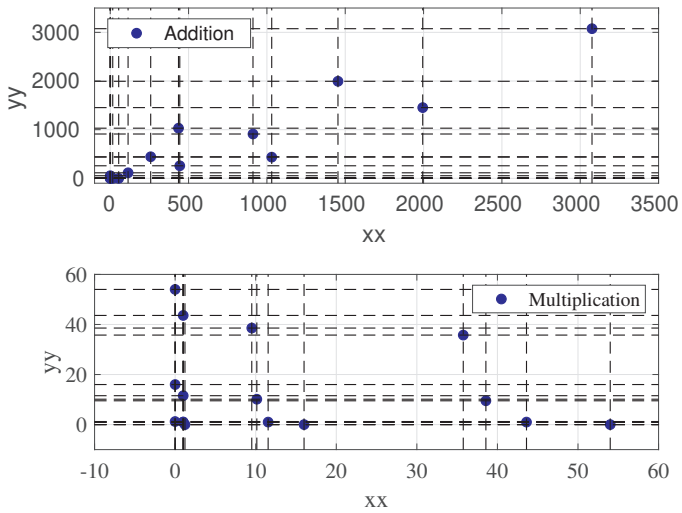

Figure 6. A two-dimensional vector quantization levels for the examples in Table II and Table III.

V. Discussion

The main claim of this paper is that all numerical values that are represented by the finite-memory data structures on any computing machine can be considered to be integers. It is a mathematical property, which is unaffected how the values are actually stored in software and hardware, e.g., using the fixed point or floating point number representations. This has a number of fundamental consequences. In particular, mathematical models on computing machines can be represented by systems of Diophantine equations. These equations have been studied extensively in the literature, and there is a rich mathematical theory available to understand their properties.

It may be claimed that any computer program or algorithm implemented in actual software and hardware can be modeled as a generator of the integer sequences. The integer sequences, in turn, can be generated by multivariate polynomials with the integer coefficients and integer variables. Such integers are then referred to as Diophantine sets; these sets are computably enumerable by Diophantine polynomials (the MRDP theorem). Unfortunately, there is no algorithm that could find the integer roots of general polynomials in several variables with the integer coefficients, or just to determine, if such a solution might even exist. Hence, there is a large gap between mathematical description based on the real analysis, and the actual implementation of algorithms on the computers [12].

Furthermore, once the numbers on computing machines are considered to be integers, the meaning of decimal point becomes mainly syntactical to allow aligning the operands in arithmetic operations. Improving the accuracy of machine numbers by p-adic representations [15] and by Diophantine approximations [26] is impractical, since the corresponding arithmetic operations would require more time and consume more memory. There are several efficient multi-precision arithmetic C-libraries that are available as open-source. Crucially, in many practical applications, the finite accuracy is often sufficient [17]. There are, however, scenarios when the full accuracy is required, for example, in cryptography [27].

The FLT was considered as an example of a Diophantine equation. Several modifications of the FLT were considered to allow the solutions to exist, and to explore how the different number representations affect the solution. Exchanging the numbers equality for their modulo equivalence immediately provides many solutions for the FLT, and likely for other Diophantine problems. The number congruence defined by a newly introduced double modulo operator induces periodicity, when the number is cut-off from the left, and it induces quantization, when the number is cut-off from the right. Whereas quantization problems have been studied extensively, it would be very useful to identify the scenarios where the induced periodicity is not only acceptable, but also useful, for example, to increase the range of the numerical values that can be considered.

Another promising strategy for solving Diophantine equations is to offset all the integers by a constant real value. This strategy can provide a solution for the roots of at lest some

multivariate Diophantine polynomials including the FLT. The proof was presented for the FLT, but it is likely also feasible for more general cases. The caveat is that the offsetted integers break down the validity of arithmetic operations, unless the offset is determined for each pair of numbers involved in the specific arithmetic operation. The offsetted integers can be exploited to define the mixed arithmetic aided quantization, so that arithmetic operations can be performed either in the analog domain, or equivalently, in the discrete domain for the integer indices. This was achieved by assuming a simple exponential function mapping the integer indices to the discrete real values, which can be also used for quantization. Other functions could be considered for other binary arithmetic operations and even more complex computations.

The future work can define and prove other mathematical properties of the machine numbers and the associated integer arithmetic. It can lead to more efficient design of integer-based models and architectures for large-scale computing machines as well as the improved approximations of complex mathematical models. The focus should be on transforming mathematical models using unconstrained numerical values to their constrained representations on the computing machines with limited resources. The interval arithmetic, which was briefly mentioned, but otherwise not considered, could be developed further towards this goal. Moreover, the rich theory involving integer sets including Diophantine equations, ordinal and cardinal arithmetic could be utilized to more formally describe the key characteristics of computing systems as the integer processing machines. There are also many fundamental theoretical results that can be adopted for the integer representations of computer programs, which are provided by Presburger and Peano arithmetic, interval arithmetic, and the Gödel's incompleteness theorems.

VI. Conclusion

The paper investigated the problem of representing the numerical values on computing machines as the finite-memory data objects. Such numbers can be then treated as integers. The other key ideas introduced in the paper were defining the equivalences between the numbers assuming only subsets of digits in their number string representations, and considering the sets of natural numbers offset by the real-valued constants. The dual modulo operator was introduced, which allows removing the most significant and the least significant digits. The basic properties of arithmetic involving the dual modulo operator were presented. The FLT was studied as an example of Diophantine equation. The Fermat metric was introduced to measure the distances between natural numbers as well as to cluster general sets of any numbers. Finally, a two-dimensional non-uniform quantization was introduced to support the mixed arithmetic with binary operations that can be equivalently performed in the analog and the discrete domains.

Acknowledgment

This work was funded by a research grant from Zhejiang University.

References

[1] P. Loskot, "On machine integers and arithmetic," in *Proc. 8th International Conference on Advances in Signal, Image and Video Processing (SIGNAL), Barcelona, Spain*, March 13–17, 2023, pp. 63–66.

[2] T. Rado, "On non-computable functions," *Bell System Technical Journal*, vol. 41, no. 3, pp. 877–884, May 1962.

[3] X. Zheng and R. Rettinger, "Weak computability and representation of reals," *Mathematical Logic Quarterly*, vol. 50, no. 4–5, pp. 431–442, Sep. 2004.

[4] R. T. Kneusel, *Numbers and Computers*, 2nd ed. Springer International Publishing, Cham, Switzerland, 2017.

[5] J. L. Gustafson and I. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, pp. 71–86, Jun. 2017.

[6] R. E. Moore, *Methods and Applications of Interval Analysis*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1979.

[7] T. Hickey, Q. Ju, and M. H. van Emden, "Interval arithmetic: From principles to implementation," *Journal of the ACM*, vol. 48, no. 5, pp. 1038–1068, Sep. 2001.

[8] T. Granlund, "The GNU multiple precision arithmetic library, GNU MP 6.2.1," https://gmplib.org, Jan. 2020, accessed: 2023-01-30.

[9] G. Hanrot, V. Lefévre, P. Pélissier, P. Théveny, and P. Zimmermann, "The GNU floating-point computations with correct rounding library GNU MPFR 4.2.1," https://www.mpfr.org/, Aug. 2023, accessed: 2023-09-30.

[10] A. Enge, M. Gastineau, P. Théveny, and P. Zimmermann, "The GNU multi-precision c library rounding library GNU MPC 1.3.1," https://www.multiprecision.org/mpc/, Dec. 2022, accessed: 2023-09-30.

[11] M. Wrocławski, "Representations of natural numbers and computability of various functions," in *Conference on Computability in Europe*, 2019, pp. 298–309.

[12] R. Krebbers and B. Spitters, "Type classes for efficient exact real arithmetic in Coq," *Logical Methods in Computer Science*, vol. 9, no. 1:01, pp. 1–27, Feb. 2013.

[13] R. O'Connor, "A monadic, functional implementation of real numbers," *Mathematical Structures in Computer Science*, vol. 17, no. 1, pp. 129–159, 2007.

[14] J. van der Hoeven, "Computations with effective real numbers," *Theoretical Computer Science*, vol. 351, pp. 52–60, 2006.

[15] F. Q. Gouvêa, *p-adic Numbers: An Introduction*, 3rd ed. Springer, Cham, Switzerland, 2020.

[16] G. Kremer, F. Corzilius, and E. Ábrahám, "A generalised branch-and-bound approach and its application in SAT modulo nonlinear integer arithmetic," in *Computer Algebra in Scientific Computing*, vol. 9890, 2016, pp. 315–335.

[17] NASA/JPL Edu, "How many decimals of Pi do we really need?" https://www.jpl.nasa.gov/edu/news/2016/3/16/how-many-decimals-of-pi-do-we-really-need, Oct. 2022, accesssed: 2023-01-30.

[18] T. Gowers, J. Barrow-Green, and I. Leader, *The Princeton Companion to Mathematics*. Princeton University Press, Princeton, NJ, USA, 2008.

[19] M. Křížek, , F. Luca, and L. Somer, *17 Lectures on Fermat Numbers: From Number Theory to Geometry*. Springer New York, USA, 2001, ch. Fermat Number Transform and Other Applications, pp. 165–186.

[20] L. J. Lander and T. R. Parkin, "Counterexample to Euler's conjecture on sums of like powers," *Bulletin of AMS*, p. 1079, Nov. 1966.

[21] N. D. Elkies, "On A4+B4+C4=D4," *Mathematics of Computation*, vol. 51, no. 184, pp. 825–835, Oct. 1988.

[22] W. K. Hayman, "Waring's theorem and the super Fermat problem for numbers and functions," *Complex Variables and Elliptic Equations*, vol. 59, pp. 85–90, Jan. 2014.

[23] L. N. Vaserstein and E. R. Wheland, "Vanishing polynomial sums," *Communications in Algebra*, vol. 31, no. 2, pp. 751–772, 2003.

[24] T. Cai and Y. Zhang, "A variety of Euler's sum of powers conjecture," *Czechoslovak Mathematical Journal*, vol. 71, pp. 1099–1113, May 2021.

[25] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004.

[26] B. Church, "Diophantine approximation and transcendence theory," Apr. 2019, Lecture Notes.

[27] J. Hoffstein, J. Pipher, and J. H. Silverman, *An Introduction to Mathematical Cryptography*, 2nd ed. Springer, New York, NY, USA, 2014.