Variable-Radix Coding of the Reals

Peter Lindstrom
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA, USA
pl@llnl.gov

***Abstract*—Recently proposed real number systems like POSITS and ELIAS codes make use of tapered accuracy resulting from variable-length coding of exponents and significands. Several quite different interpretations of these number systems have been provided, though most often these rely on some combination of fixed- and variable-length codes for exponent and significand. We provide a new perspective on these number systems that unifies known representations while suggesting new ones. Our framework is based on multibit radix representations that encode the exponent in unary, the leading nonzero digit in a variable-length code, and the remaining digits in fixed-length binary code. We show how POSITS, the various ELIAS codes, and IEEE 754 like representations can be expressed in this framework. Moreover, we show that POSITS and the ELIAS γ and δ codes represent the leading digit using the canonical Huffman code for a probability distribution given by Benford's law, which governs the probability of leading digits. We further show that POSITS correspond to the use of a fixed radix while ELIAS δ and ω codes are based on simple sequences of increasing radix. Our approach provides for an intuitive and uniform framework for representing numbers that reveals a visual mapping between codewords and the binary representation of real numbers obscured by prior frameworks. This new interpretation suggests a generalization of POSITS and other number systems and provides simple rules for designing information-theoretically optimal codes.**

***Index Terms*—real number systems, floating point, posits, Elias codes, tapered accuracy, Benford's law, Huffman code**

I. INTRODUCTION

A recent rebirth of research into systems for representing real numbers has spawned several proposals based on the notion of *tapered accuracy* [13], where the number of exponent and significand bits is not fixed. Examples include Gustafson and Yonemoto's POSITS [4], Hamada's URRS [6], Yokoo's number systems [17], and Lindstrom et al.'s [12] real-number extensions of the ELIAS codes [3], which were originally designed for positive integers. All of these number systems essentially encode the same information as conventional IEEE 754 floating point: a sign, a binary exponent, and a significand.

As observed in the recent work by Lindstrom and others, these representations can be thought of as instances of larger classes of number systems that share many properties and that can be described as simple variants of a family of coding schemes. Lindstrom et al. [12] proposed one such framework in which the number systems differ in how the exponent is encoded using one of several possible variable-length codes, most of which are well-known in the information theory community. In another view [11], numbers are encoded one bit at a time as outcomes of comparisons with sequences of numbers that partition finite intervals (binary search) or unbounded intervals (unbounded search). These number sequences often have very simple rules, allowing whole number systems to be defined using only two simple expressions. Yet a third interpretation of POSITS is provided by Gustafson and Yonemoto [4], who invoke the idea of "regime" bits in addition to exponent bits. Frameworks like these are indispensable for rapid prototyping of new representations and for implementing and comparing emerging number systems in target applications like scientific computing and machine learning.

Although prior frameworks and interpretations serve to categorize and contrast number systems, they have some drawbacks. First, it is not trivial to develop a new variable-length prefix code of exponents that has the properties desired for a particular application, and implementing such codes can be tedious and error prone. Second, the partition-based scheme, though simple, is not performant as it requires high-precision arithmetic operations for each bit encoded or decoded. Finally, it is not clear how to generalize the idea of regime and exponent bits to number systems other than POSITS.

In this paper, we propose yet another interpretation of POSITS based on the notion of a fixed base (or radix) other than two. We show how to generalize this representation to those in which the radix is not fixed but varies from one digit to the next, as is needed to describe URR and the ELIAS δ and ω codes. Under this interpretation, the POSIT regime bits serve the purpose of a high-radix signed exponent encoded in unary, with a variable-length encoding of the leading (nonzero) digit. We show how this leading-digit encoding is in a sense optimal for the codes considered and is given by the canonical Huffman code [8] associated with the distribution given by Benford's law of leading digits [1]. This variable-length encoding prevents the wobbling accuracy associated with other high-radix representations [7]. Under this new framework, a number system is parameterized simply in terms of the radix sequence, which in all cases considered here is given by a simple expression. Our framework further suggests generalizations of POSITS to bases not previously considered as well as novel number representations with attractive properties. Finally, we believe this new interpretation is more intuitive than prior frameworks, with the additional benefits of a simple uniform implementation and improved performance over bisection-based bitwise operations.

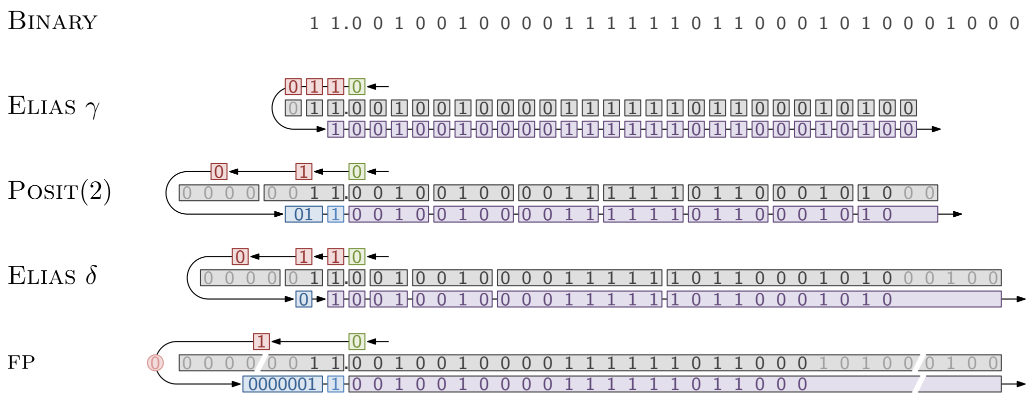

Fig. 1. 32-bit representations of π in several number systems. Codeword bits are split into sign, level, leading digit, and trailing digits. The binary representation in gray is partitioned into digits of possibly varying radix. Each level bit indicates whether the corresponding digit is significant; the implicit FP level terminator is shown circled. Arrows indicate the order that bits are sequenced into a codeword and illustrate the sequential scan for the leading digit.

A. Preliminaries

We consider the task of encoding real numbers $x \in \mathbb{R}$. We may represent $x \neq 0$ as $\langle s, e, m \rangle$ such that $x = (-1)^s \beta^e m$, where s encodes the *sign*, e is the *exponent*, $m \in [1, \beta)$ is the *significand*, and $\beta \geq 2$ is the *radix* (or *base*). $y = \mathcal{E}(x)$ and $x = \mathcal{D}(y)$ encode reals and decode *codewords* (bit strings), respectively. We will sometimes perform arithmetic on codewords y, which is to be understood as integer arithmetic on the two's complement interpretation of y.

As in [4], [11], [12], the number systems expressible in our framework share these properties:

- Zero: $\mathcal{E}(0) = 000\ldots0$.
- One: $\mathcal{E}(1) = 010\ldots0$.
- Not a real: $\mathcal{E}(\perp) = 100\ldots0$.
- Two's complement: $\mathcal{E}(-x) = -\mathcal{E}(x)$.
- Reciprocal symmetry: $\mathcal{E}(2^i) - \mathcal{E}(2^j) = \mathcal{E}(2^{-j}) - \mathcal{E}(2^{-i})$.
- Lexicographic ordering: $x < x' \iff \mathcal{E}(x) < \mathcal{E}(x')$.
- Nesting: $\mathcal{D}(y\ 0) = \mathcal{D}(y)$.
- Rounding: Round to nearest, ties to even.

Reciprocal symmetry implies that for all $i \in \mathbb{Z}$ there are as many representable numbers in $[1, 2^i)$ as there are in $[2^{-i}, 1)$ and that $\mathcal{E}(2^{-i}) = \mathcal{E}(\perp) - \mathcal{E}(2^i)$. Reciprocal closure is otherwise not guaranteed for non-powers-of-two. The nesting property implies that the value associated with a codeword does not change when appending zero-bits.

Because negative numbers are trivially handled via two's complement encoding and $x = 0$ is encoded as all-zero bits, we will for simplicity assume that $0 < x < \infty$. We will often start from the binary representation of x,

$$x = \sum_{i=-\infty}^{\infty} b_i 2^i, \tag{1}$$

where b_i denotes the ith bit. Bits $i \geq 0$ appear to the left of the radix point; bits $i < 0$ appear to the right. We usually omit leading-zero bits in the binary representation. More generally, for $\beta \geq 2$, we use d_i to denote the ith digit. $d_\ell = \lfloor m \rfloor$ denotes the leading nonzero digit such that $m = d_\ell + f_\ell$, where $f_\ell \in [0, 1)$ is the *fraction* given by the trailing digits.

As we shall see later, the radix β needs not be fixed but may vary from one digit to the next. When β is a power of two, we use $w = \log_2 \beta$ to denote the number of bits spanned by the radix. For instance, in hexadecimal number systems such as the IBM 360 floating-point system [16], $\beta = 16$ and $w = 4$.

We will make use of *signed unary* encoding of integers. A nonnegative integer $i \geq 0$ has a signed unary encoding of $1^{i+1}0$, i.e., $i + 1$ one-bits followed by a zero-bit. A negative integer $i < 0$ is encoded as $0^{-i}1$. The leading bit thus indicates the sign of i, and the rightmost bit of opposite value terminates the codeword.

II. Prior Work: Posits

POSITS are by far the most common tapered representation today. The POSIT format defines a family of representations that are parameterized by p, called the "exponent size" in [4]. In this section, we review known interpretations of POSITS to provide some context for our proposed framework.

A. The Regime Interpretation

The best known POSIT interpretation is the authors' own [4]: numbers are given by tuples $\langle s, \ell, t, f \rangle$ where ℓ encodes the "regime"—the most significant bits of the base-2 exponent e—and t encodes the p least significant bits of e:

$$e = 2^p \ell + t. \tag{2}$$

Then $x = (-1)^s\, 2^e\, (1 + f)$. In [4], t is called *the* exponent, which is a bit of a misnomer since it represents only the few trailing bits of e. The encoding of e is done in two parts: a variable-length signed unary code for ℓ followed by p bits for t. The fraction, f, is then appended. The leading one-bit $b_e = 1$ is implied for all $x \neq 0$ and is not stored.

B. The Golomb-Rice Interpretation

Lindstrom et al. [12] realized that POSITS differ from IEEE 754 floating point primarily by using variable-length encoding of the base-2 exponent, e. Whereas IEEE allocates a fixed number of bits for the exponent, which is encoded in binary, POSITS encode the exponent using a generalization

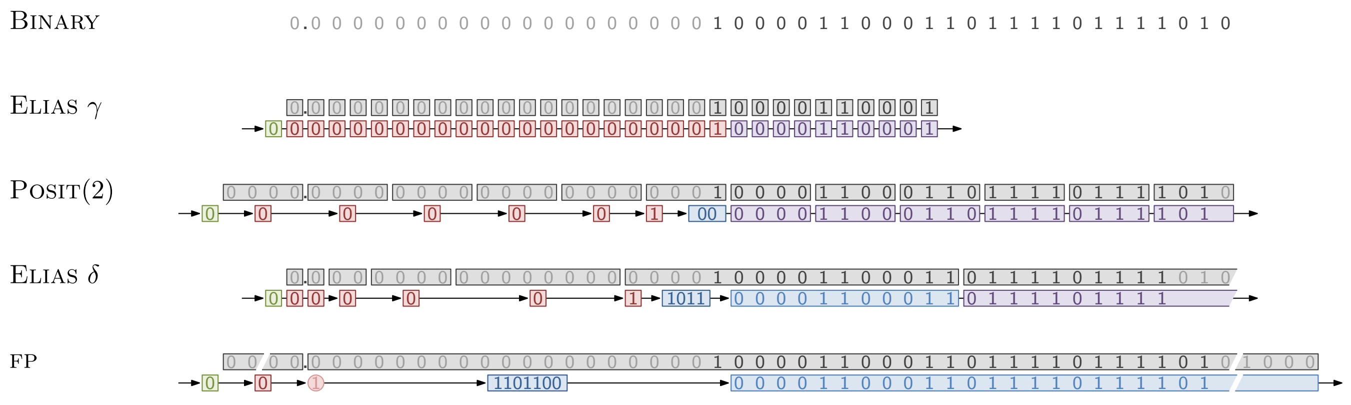

Fig. 2. 32-bit representations of 10^{-6} in several number systems. See Figure 1 for details.

TABLE I
POSIT(2) ($\beta = 16$) CODES FOR A SELECTION OF INTEGERS SEPARATED INTO SIGN, LEVEL, LEADING DIGIT, AND TRAILING DIGITS.

x	$\mathcal{E}(x)$
$-1_{10} = -1_{16}$	1 10 00
$0_{10} = 0_{16}$	0 000...
$1_{10} = 1_{16}$	0 10 00
$2_{10} = 2_{16}$	0 10 01 0
$3_{10} = 3_{16}$	0 10 01 1
$5_{10} = 5_{16}$	0 10 10 10
$8_{10} = 8_{16}$	0 10 11 000
$13_{10} = \mathtt{d}_{16}$	0 10 11 101
$21_{10} = 15_{16}$	0 110 00 0101
$34_{10} = 22_{16}$	0 110 01 0 0010
$256_{10} = 100_{16}$	0 1110 00 0000 0000
$4095_{10} = \mathtt{fff}_{16}$	0 1110 11 111 1111 1111
$4660_{10} = 1234_{16}$	0 11110 00 0010 0011 0100

of Golomb-Rice codes [14]. Golomb-Rice codes for nonnegative integers, z, are parameterized by a power-of-two integer $w = 2^p$. Then $z = wq + r$ with $0 \leq r \leq w - 1$. The quotient $q \geq 0$ is encoded in unary as 1^q0, while the residual r is encoded in binary as p bits. To generalize Golomb-Rice to negative integers, Lindstrom et al. used signed unary encoding of q, so that the code for $q < 0$ begins with one or more zeros terminated by a one-bit. They further show how exponent codes other than Golomb-Rice may be used to generalize the ELIAS codes to numbers other than positive integers.

C. *The Exponential Search Interpretation*

Bentley and Yao [2] showed how the ELIAS codes arise from binary comparisons in unbounded search. In unbounded search for a positive integer x, one scans a monotonic sequence (a_i) until $a_i \leq x < a_{i+1}$. Each comparison between x and a_i corresponds to a codeword bit. Once x has thus been bracketed in a bounded interval, the interval is then successively narrowed, e.g., using binary search, which generates further bits. As an example, POSIT(p) corresponds to the sequence $a_i = 2^{2^p \times i} = 2^{2^p} a_{i-1}$. The representable set $[1, \infty)$ is then extended to all reals via two's complement negation and reciprocation: $a_{-i} = a_i^{-1}$. As shown in [11], number systems can be defined via two simple rules: one for generating the sequence $a_i \to a_{i+1}$, and one for bisecting finite intervals $[a, b) \to c$, with $a < c < b$.

III. FIXED-RADIX REPRESENTATIONS: POSITS

We now arrive at our new fixed-radix interpretation of POSIT(p). Instead of $\beta = 2$, we let the radix (called "useed" in [4]) be $\beta = 2^w$ with $w = 2^p$. Starting from the binary exponent $e = w\ell + t$ (see Eq. (2)) and the binary significand $m \in [1, 2)$, we rewrite $x > 0$ from base 2 to base β as

$$\begin{aligned} x &= 2^e m \\ &= 2^{w\ell+t}\left(1 + \sum_{i=1}^{\infty} m_{-i} 2^{-i}\right) \\ &= (2^w)^\ell \left(\underbrace{2^t + \sum_{i=1}^{t} m_{-i} 2^{t-i}}_{d_\ell} + \underbrace{\sum_{i=t+1}^{\infty} m_{-i} 2^{t-i}}_{f_\ell}\right) \\ &= \beta^\ell (d_\ell + f_\ell). \end{aligned} \quad (3)$$

Thus, ℓ is the base-β exponent, $d_\ell \in \{1, \ldots, \beta - 1\}$ is the leading nonzero digit, and $f_\ell \in [0, 1)$ represents a trailing fraction. Our POSIT encoding of x uses signed unary for the exponent, ℓ, and a variable-length code for $d_\ell = 2^t + r$. This code represents t in p bits followed by t additional bits for r, with $0 \leq t \leq w - 1$. All remaining digits, $d_i \in \{0, \ldots, \beta - 1\}$, of f_ℓ are simply appended w bits at a time. After incorporating the sign bit s of x, the codeword is finally truncated or zero padded to n bits of precision (e.g., $n = 32$).

We note that, for $x \geq 1$, the unary code $1^{\ell+1}0$ indicates that there are $\ell + 1$ significant digits to the left of the radix point, and each codeword one-bit (from left to right) indicates significance of a corresponding base-β digit (from right to left). When $x < 1$, the unary code $0^{-\ell}1$ indicates there are $-\ell$ leading zeros (including the zero just left of the radix point) before the first nonzero digit. Again, the terminating one-bit marks a significant digit. Hence, the leftmost bit of the codeword for ℓ tells us the direction in which to sequentially scan for d_ℓ starting from d_0: 1 for left and 0 for right.

Figures 1 and 2 and Table I illustrate the POSIT(p) encoding of reals for $p = 0$ (where they coincide with the ELIAS γ code) and for $p = 2$. Here the red colored bits encode the base $\beta = 2^{2^p}$ exponent ℓ, which we more generally will refer to as the *level* (see below). Each red one-bit indicates that

TABLE II
The leading-digit distribution predicted by Benford's law and the empirical distributions observed in two numerical applications for base $\beta = 16$. H denotes Shannon entropy in bits/symbol; L^* is the expected (optimal) length of the Huffman code for the empirical distribution; L is the expected length of the Benford code; R^2 is the Pearson correlation with the Benford probability distribution. The Benford code is optimal for MatInv and close to optimal for Euler2D.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	H	L^*	L	R^2
Benford	.250	.146	.104	.080	.066	.056	.048	.042	.038	.034	.031	.029	.027	.025	.023	3.481	3.500		
MatInv	.231	.164	.098	.081	.061	.053	.064	.042	.038	.035	.032	.028	.026	.024	.024	3.499	3.524	3.524	.991
Euler2D	.352	.166	.113	.062	.041	.054	.085	.023	.018	.017	.015	.014	.014	.013	.014	3.040	3.106	3.145	.981

the corresponding digit in gray is significant (i.e., is not a leading zero). The blue colored bits encode the leading digit $d_\ell = 2^t + r$ as a pair $\langle t, r\rangle$ in $p+t$ bits. For $p = 0$, $d_\ell = 1$ when $x \neq 0$, and therefore the leading-digit code has zero length. The FP representation illustrates both POSIT(7) and a slight variation on IEEE 754 single precision. We here have as radix the rather large number $\beta = 2^{128}$ (i.e., 128-bit digits). In this base, FP allows only two digits: one to the left and one to the right of the radix point. This limitation on number of digits makes FP not a universal code, as numbers larger than 2^{128} cannot be represented. Whereas POSIT(7) allows for more digits and terminates the signed unary exponent code with a bit (shown circled), this terminal bit is implicit in FP. What we usually think of as the binary exponent in IEEE floating point represents the exponent t in the leading-digit code in FP.

A. POSITS *vs. Hexadecimal Floating Point*

Although POSITS arose independently of our derivation, is there a rationale for how they encode numbers, and what are the ramifications of the POSIT coding scheme? For instance, how does the case $p = 2, \beta = 16$ compare to IBM hexadecimal floating point [16], which also uses base-16 encoding? There are two key differences: (1) POSITS encode the exponent in signed unary rather than biased binary, allowing common exponents near zero to be represented using fewer bits. (2) POSITS use a variable-length code for the leading digit, d_ℓ. This code uses p bits when $d_\ell = 1$ and $p + 2^p - 1$ bits when $d_\ell = \beta - 1$, i.e., fewer bits are assigned to smaller digits than to larger ones. This is contrary to IBM floating point, which unconditionally uses $2^p = 4$ bits to encode the leading digit.

With respect to exponent coding, we note that a variable-length code that assigns short codewords to small (in magnitude) exponents is more suitable from a statistical accuracy standpoint when numbers near one are more common. Empirically, this tends to be the case in numerical applications [9], [12], e.g., because equations are often solved in dimensionless form [10]. Such rescaling is also done to avoid under- and overflow, especially in mixed-precision computations [5].

Regarding leading-digit coding, we note that IBM floats suffer from *wobbling accuracy* [7], a phenomenon where the relative accuracy changes abruptly (by four bits) as x traverses a power of β and the leading digit flips from $\beta - 1$ to 1. IBM floats further waste bit pattern 0000 (i.e., every sixteenth codeword), which for normalized $x \neq 0$ never occurs as the leading digit. In contrast, POSITS disallow $d_\ell = 0$ and use fewer bits to encode $d_\ell = 1$ than $d_\ell = \beta - 1$, which eliminates waste and largely cancels wobbling accuracy.

B. *Benford Code*

One question remains: Is the particular leading-digit code employed by POSITS justifiable? If we knew the distribution of leading digits, then we could devise a code that minimizes the expected code length. As is well-known from information theory, such a *minimum-redundancy* code is given by the greedy Huffman construction algorithm [8]. Perhaps nonintuitively, the empirical probability distribution of leading digits is often not uniform but is skewed toward small digits. This observation was codified into *Benford's law* [1], which states

$$P_\beta(d) = \log_\beta(d+1) - \log_\beta(d) = \log_\beta(1 + 1/d) \quad (4)$$

Benford's law arose from observations of how early pages of logarithm tables were more worn than late pages. Is this law applicable to numerical computations? Table II lists the probabilities given by Benford for $\beta = 16$ and those observed for every evaluated expression in two numerical computations: the inversion of a Vandermonde matrix and the Euler2D shock propagation mini-application from [12]. The observed distributions are a remarkable match with Benford's law, with Pearson correlation exceeding 0.98. Note that no special rescaling of numbers was done in either application.

As a consequence of our observation of leading-digit distributions, we propose the variable-length *Benford code*, which we define as the canonical Huffman code [14], [15] associated with Benford's probability distribution given by Eq. (4). The Benford code is uniquely defined for any radix β and is, by construction, lexicographic. The lexicographic property ensures that the ordering of real numbers is preserved by their codewords. As an example, the seven Benford codewords for $\beta = 8$ are shown in Figure 3(right).

Let $\beta = 2^{2^p}$ for some integer $p \geq 0$. We conjecture that the Benford code for a digit $d = 2^t + r$, with $1 \leq d \leq \beta - 1$, $0 \leq r \leq 2^t - 1$ is given by the p-bit representation of t followed by the t-bit representation of r. Whereas we have exhaustively verified that this code is indeed the canonical Huffman code for $0 \leq p \leq 4$, which covers all the standard POSITS, we do not yet have a proof for general p. However, we note that the Benford distribution satisfies

$$\sum_{d=2^t}^{2^{t+1}-1} P_\beta(d) = \log_\beta 2 = 2^{-p} \quad (5)$$

for all $0 \leq t \leq 2^p - 1$. In other words, as a group, each of the 2^p binades is equally likely, and therefore using a p-bit prefix to first select the binade given by t is reasonable. Moreover, $P(d) = P(2d) + P(2d+1) > P(d+1)$, so the

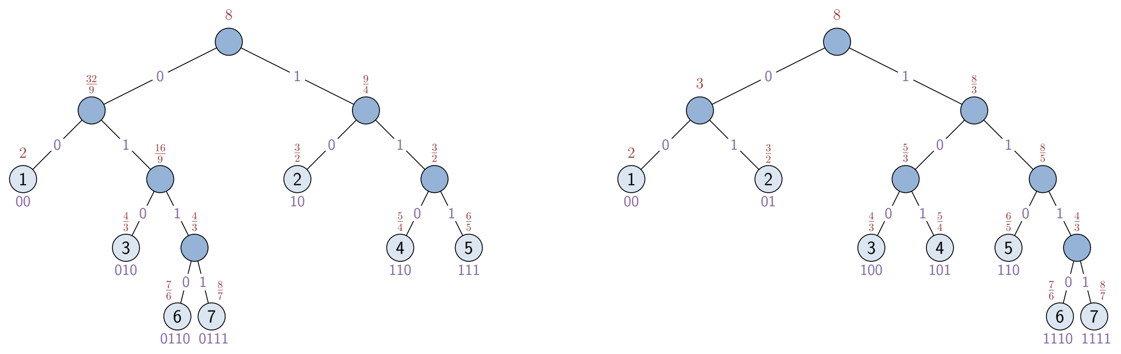

Fig. 3. Prefix code trees corresponding to Huffman code (left) and Benford code (right) for leading digits $1 \le d \le 7$ in base $\beta = 8$. Codewords are shown below each leaf node and have the same length for the same digit in the two codes. Node weights $\beta^{P(d)}$ given by digit probabilities $P(d)$ are shown above nodes and are formed as products of child weights. These weights drive the Huffman tree construction by iteratively pairing the two nodes with lowest weight.

last two numbers in a binade have a combined probability that exceeds the probability of the first number in the binade. This implies that a Huffman tree construction of symbols within a binade proceeds level by level and forms a complete binary tree. That is, a Huffman code restricted to a binade of 2^t numbers is of fixed length t. Thus, the encoding of r uses a fixed number of t bits. These two observations are strong evidence but not sufficient proof that the POSIT leading-digit code also is the Benford code for arbitrary p. For instance, the bottom-up Huffman construction generally mixes digits from different binades, although this does not preclude that the code used by POSITS is the equivalent canonical Huffman code. We leave completing the proof as future work. Finally, using Eq. (5), it is easy to show that the expected Benford code length is $(2^p - 2p + 1)/2$, which saves half a bit or more over coding the leading digit using a fixed 2^p number of bits.

IV. VARIABLE-RADIX REPRESENTATIONS: ELIAS CODES

In the previous section, we assumed that β is fixed. By relaxing this constraint and allowing β_i to vary from one digit d_i to the next, we expand the space of number representations. For simplicity, we here limit β_i to powers of two, $\beta_i = 2^{w_i}$, although this is not strictly a requirement—e.g., we could as radices use the Fibonacci numbers. We do, however, impose one additional constraint: $\beta_{-i} = \beta_{i-1}$, which ensures reciprocal symmetry. Here the subscript i refers to the place of each digit: $i = 0$ is the digit just left of the radix point; $i = -1$ is the digit just right of the radix point. Aside from this relaxation, the coding scheme proceeds just as in the fixed-radix case.

Since the radix is no longer constant, we cannot speak of a base-β exponent. Rather, we introduce the more general concept of *level*, ℓ, which relates to the number of variable-radix factors in the product that scales a number. Let $\beta_i = \frac{a_{i+1}}{a_i} \quad \forall i \in \mathbb{Z}$. Here a_i is a sequence point from unbounded search (Section II-C). Then, since $a_0 = 1$, we have as telescoping products

$$a_\ell = \begin{cases} \prod_{i=0}^{\ell-1} \beta_i & \text{if } \ell > 0 \\ 1 & \text{if } \ell = 0 \\ \prod_{i=\ell}^{-1} \beta_i^{-1} & \text{if } \ell < 0 \end{cases}, \tag{6}$$

TABLE III
SEARCH SEQUENCE a_i, $i \ge 0$, AND CORRESPONDING RADIX WIDTH w_i IN BITS FOR SEVERAL NUMBER SYSTEMS.

Code	a_i	w_i	$(w_i)_{i \ge 0}$
ELIAS γ	2^i	1	$(1, 1, 1, \ldots)$
POSIT(p)	$2^{2^p \times i}$	2^p	$(2^p, 2^p, 2^p, \ldots)$
URR	$2^{\lfloor 2^{i-1} \rfloor}$	$\lceil 2^{i-1} \rceil$	$(1, 1, 2, 4, 8, \ldots)$
ELIAS δ	$2^{2^i - 1}$	2^i	$(1, 2, 4, 8, 16, \ldots)$
ELIAS ω	$2 \uparrow\uparrow i$	$2 \uparrow\uparrow i - 2 \uparrow\uparrow (i-1)$	$(1, 1, 2, 12, 65520, \ldots)$

such that $x = (-1)^s a_\ell (d_\ell + f_\ell)$. For instance, $a_3 = \beta_2 \beta_1 \beta_0$ and $a_{-2} = (\beta_{-1}\beta_{-2})^{-1} = (\beta_0 \beta_1)^{-1}$. Of course, when $\beta_i = \beta$ is fixed, $a_\ell = \beta^\ell$. As in the case of fixed-radix codes, we encode ℓ in signed unary; all other details of encoding remain the same. Figures 1 and 2 illustrate the ELIAS δ code as an example of a variable-radix code, with $\beta_i = 2^{2^i}$.

We parameterize a number system by the radix sequence (β_i), which due to symmetry needs only be specified for $i \ge 0$. In practice, we prefer number systems for which each radix is a power of two, in which case the radix width (w_i) parameterizes the system, with $\beta_i = 2^{w_i}$. Table III lists example codes and their corresponding sequences (a_i) and (w_i), which in all cases are simple expressions.

A. ELIAS ω

For ELIAS ω, $a_i = 2 \uparrow\uparrow i = 2^{2 \uparrow\uparrow (i-1)}$ is given by tetration [11], with $2 \uparrow\uparrow -1 = 0$. This results in w_i being the difference of two powers of two, e.g., $w_3 = 2 \uparrow\uparrow 3 - 2 \uparrow\uparrow 2 = 16 - 4 = 12$. Hence, the Benford code for d_3 is not based on a leading p-bit exponent, but we must resort to general Huffman coding. Whereas the Huffman code is fixed for a given w_i and can be precomputed and stored in a lookup table, doing so for ELIAS ω is impractical as the size of the lookup table for $w_4 = 65520$, which is needed to code modest numbers $x \ge 2^{16}$, already exceeds the number of atoms in the universe. We further note that ELIAS ω is the first code we have encountered that does not use Benford coding of the leading digit, as can be verified by examining the codewords for $\beta_3 = 2^{12}$. This suggests that ELIAS ω is intrinsically suboptimal (for distributions that follow Benford's law) and that a better number system exists that *does* use Benford code for the leading digit.

V. Discussion

The number systems representable in our framework encode a number $x \neq 0$ in two's complement as:

- A **sign** bit: 0 for nonnegative and 1 for negative numbers.
- A **level** (or exponent, when the radix is fixed) encoded in signed unary that marks with one bit each significant digit to the left of the radix point ($|x| \geq 1$) or each leading zero to the right of the radix point ($|x| < 1$).
- A **leading digit** in variable-length Benford code, which is particularly simple when $\log_2 \log_2 \beta$ is an integer.
- All **trailing digits** in fixed-length binary.

Each number system in this family is uniquely defined by a sequence of radices (β_i). Our number systems are *universal* in the sense that each real number x corresponds to a unique codeword. This codeword's length is no more than a constant factor of the length of the binary representation of x.

Our framework immediately suggests several new number systems that deserve future investigation, some of which are illustrated in the accuracy plot in Figure 4 for $n = 32$:

- A generalization of Posits to radix widths w that are not whole powers of two. For instance, $\beta = 2^3 = 8$ falls somewhere between Posit(1) ($\beta = 2^{2^1} = 4$) and Posit(2) ($\beta = 2^{2^2} = 16$). As discussed earlier, the Benford code for any β is unique and, when β is fixed, can be efficiently implemented as a small lookup table.
- Generalization to the case $\beta = 10$ for decimal coding.
- The code given by $w_i = i+1$ grows the radix more slowly than Elias δ, keeping the dynamic range reasonable while extending it over Posits. Such a number system avoids the need for Posits to increase p with precision to achieve a high enough dynamic range while also ensuring maximum accuracy of numbers near one.
- Elias δ, with $\beta_i = 2^{2^i}$, can be thought of as traversing the family Posit(p), with $p = i$. A hybrid scheme caps the radix by using $\beta_i = 2^{2^{\min\{i,p\}}}$.
- A modification of Elias δ that repeats radices, say, k times, to inhibit too-rapid growth: $w_i = 2^{\lfloor i/k \rfloor}$.
- Completely arbitrary, non-power-of-two radices.

VI. Conclusion

We have proposed a new framework for defining universal number systems based on the notion of radix sequences and tapered accuracy, in which numbers whose magnitude is close to one are most accurately represented. Our framework allows modeling known number systems such as Posits, Elias codes, and simple variations on IEEE floating point. It allows for an intuitive mapping between the usual binary representation of a number and its codeword, and allows new number systems to be defined in a natural way by specifying only the radix sequence. Our framework relies on the newly introduced Benford code to encode leading digits, which is an optimal prefix code for distributions that follow Benford's law. Finally, we show how Posits make use of this code and suggest a natural extension of the Posit family to new members worthy of consideration.

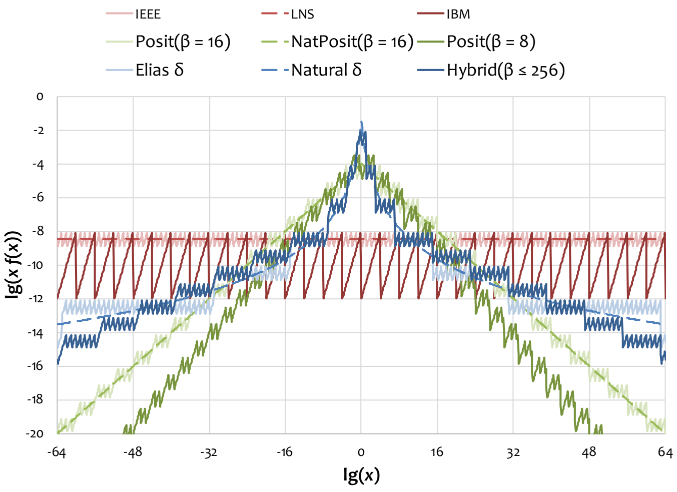

Fig. 4. Relative accuracy [11] in bits vs. exponent. $f(x) = F'(x) \approx \frac{\Delta F}{\Delta x}$ is the probability density with $F(x) = 2^{-n}\mathcal{E}(x)$ for n bits of precision. Natural systems [11] avoid the wobbling accuracy exemplified by IBM floats.

VII. Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was supported by the LLNL-LDRD Program under Project No. 17-SI-004.

References

[1] F. Benford. The law of anomalous numbers. *Proceedings of the American Philosophical Society*, 78(4):551–572, 1938.
[2] J. L. Bentley and A. C. Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82–87, 1976.
[3] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975.
[4] J. L. Gustafson and I. T. Yonemoto. Beating floating point at its own game: Posit arithmetic. *Supercomputing Frontiers and Innovations*, 4(2):71–86, 2017.
[5] A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham. Harnessing GPU tensor cores for fast FP16 arithmetic to speed up mixed-precision iterative refinement solvers. In *ACM/IEEE SC*, pages 603–613, 2018.
[6] H. Hamada. URR : Universal representation of real numbers. *New Generation Computing*, 1:205–209, 1983.
[7] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, second edition, 2002.
[8] D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.
[9] M. Klöwer, P. D. Düben, and T. N. Palmer. Posits as an alternative to floats for weather and climate models. In *Conference for Next Generation Arithmetic*, pages 2.1–2.8, 2019.
[10] H. P. Langtangen and G. K. Pedersen. *Scaling of Differential Equations*. Springer, 1st edition, 2016.
[11] P. Lindstrom. Universal coding of the reals using bisection. In *Conference for Next Generation Arithmetic*, pages 7:1–7:10, 2019.
[12] P. Lindstrom, S. Lloyd, and J. Hittinger. Universal coding of the reals: Alternatives to IEEE floating point. In *Conference for Next Generation Arithmetic*, pages 5:1–5:14, 2018.
[13] R. Morris. Tapered floating point: A new floating-point representation. *IEEE Transactions on Computers*, C-20(12):1578–1579, 1971.
[14] D. Salomon. *Variable-Length Codes for Data Compression*. Springer-Verlag, 2007.
[15] E. S. Schwartz and B. Kallick. Generating a canonical prefix encoding. *Communications of the ACM*, 7(3):166–169, 1964.
[16] E. M. Schwarz, R. M. Smith, and C. A. Krygowski. The S/390 G5 floating point unit supporting hex and binary architectures. In *IEEE Symposium on Computer Arithmetic*, pages 258–265, 1999.
[17] H. Yokoo. Overflow/underflow-free floating-point number representations with self-delimiting variable-length exponent field. *IEEE Transactions on Computers*, 41(8):1033–1039, 1992.