Overflow/Underflow-Free Floating-Point Number Representations with Self-Delimiting Variable-Length Exponent Field

Hidetoshi Yokoo

Department of Computer Science,
Gunma University
Kiryu, Gunma 376
Japan

Abstract

A class of new floating-point representations of real numbers, based on representations of the integers is described. In the class, every representation uses a self-delimiting representation of the integers as a variable length field of exponent, and neither overflow nor underflow appears in practice. The adopted representations of the integers are defined systematically, so that representations of numbers greater than one have both exponent-significand and integer-fraction interpretations. Since representation errors are characterized by the length function of an underlying representation of the integers, superior systems in precision can be easily selected from the proposed class.

1 Introduction

For these ten years, in order to overcome the inherent problem of overflow and underflow in conventional floating-point number representation systems, several representation systems with a variable-length exponent have been developed in Japan.

An earlier work of those appears in [12], in which the exponent may be extended to the entire data word using a field that specifies the length of the exponent. Subsequently, inspired by this work, Hamada [7] found a method of realizing a variable-length exponent without any additional field to specify the length. Hamada's URR (for universal representation of real numbers) has several attractive features which include the followings.

- Neither overflow nor underflow occurs in practice.
- Specifications do not depend on data length.
- The order of numbers coincides with the lexicographic order of their representations.
- Arbitrary real values can be infinitely approximated simply by extending data length.

Despite these, the URR system has not been implemented in any real system, partly because we have found no method of performing arithmetic directly with URR data except for some trivial operations, and partly (or maybe mainly) because we have already had some commercially established number representation systems [8], [15]. The significance of URR should rather be evaluated from the following recognition that it brought. That is, since the invention of URR, we have been recognizing that there is no essential difference between the two main systems — fixed-point and floating-point — for real numbers. To understand this recognition alone, there is no need to know the details of URR, instead the following simple example will suffice.

Consider, for example, the fixed-point representation of 0.15625 in radix 2. This can be represented as 0.00101_2. This number is, in turn, represented as a floating-point form: $1.01_2 \times 2^{-3}$. If we consider numbers between 0 and 1 only, and adopt *unary encoding* to represent the exponent such as

$$\begin{array}{ll} -1 & 1 \\ -2 & 01 \\ -3 & 001 \\ & \vdots \\ -E & \underbrace{00\cdots0}_{E-1}1, \end{array}$$

then the number $1.01_{(2)} \times 2^{-3}$ can be represented as the concatenation of 001 (exponent) and 01 (economized significand[1]). The obtained floating-point representation "00101" coincides with the fractional part of the fixed-point representation. Thus, the string "00101" has both fixed-point and floating-point interpretations. The crucial point is not to make a distinction between the fixed-point and floating-point representations but to separate the use of a predetermined boundary between the exponent and significand from the use of a self-delimiting encoding of the exponent.

[1]For this purpose, more familiar terms *fraction* or *mantissa* are usually used. In this paper, however, we adopt the term *significand* for several reasons.

Self-delimiting encodings have been studied on the several themes of (prefix-free) representations of the integers or encodings of commas between strings [1], [4], [5], [9]. Matula and Kornerup [13] discuss an application of a certain representation of the integers to the representation of rational numbers in computers. Researchers have been — and still are — seeking efficient methods for encoding of countably infinite elements. *Unary encoding* mentioned above, which is well known also in Turing machine theory, is the simplest example of representations of the integers; however, it is less efficient. The efficiency naturally depends on the definition of itself. In the problem of floating-point representation of real numbers, the efficiency of a self-delimiting representation specifying the exponent is expected to influence the precision of the number representation system.

This paper aims to discuss a unifying approach to the use of representations of the integers as the field of exponent, and presents a class of overflow/underflow-free representations of real numbers, which includes URR as a special case. This class may serve as a good answer to an exercise in [10, p.212, Exercise 17 of Sec.4.2.1], which requires a representation whose precision decreases as the magnitude of the exponent increases.[2] As far as the abolition of overflow and underflow is concerned, the level-index system [2, 3, 11] may be preferable. However, it matters little to the present paper whether a system is fixed-point, floating-point, or level-index. Instead, our main interest is the efficient and systematic method of mapping the system to binary strings. In fact, we will define a system in our class based not only on exponent-significand decomposition of real numbers but also on integer-fraction representation. The conversion between these two bases corresponds to the generalization of the floating-point and fixed-point interpretations of a representation. In the class, representation errors can be characterized by the length function of the adopted representation of the integers. As a result, we can find improved variations of URR systematically.

2 Preliminaries

In most of conventional floating-point systems in radix 2, a real number X can be approximated by

$$X = (-1)^s F \times 2^E, \tag{1}$$

where s is either 0 or 1, E is an integer, and F is a number which is normalized as either

$$1/2 \leq F < 1 \tag{2}$$

or

$$1 \leq F < 2. \tag{3}$$

A number X can be also represented as

$$X = (-1)^s(\sum_{i=0}^{n} a_i 2^i + \sum_{j=1}^{\infty} b_j 2^{-j}), \quad a_i, b_j \in \{0,1\} \tag{4}$$

which is reexpressed as the form

$$X = (-1)^s a_n a_{n-1} \cdots a_0 . b_1 b_2 \cdots \tag{5}$$

in the fixed-point system. For $X \geq 1$, $\lfloor X \rfloor$, the largest integer not greater than X, is said to be the integer part of X. A binary string $b_1 b_2 \cdots$ representing $X - \lfloor X \rfloor$ is said to be the binary fraction of X. Define $|\alpha|$ to be the length of a binary string α. If we set $a_n = 1$ and $\alpha = a_{n-1}a_{n-2}\cdots a_0$ for $X \geq 1$, then the equality $n \equiv |\alpha| = E - 1$ holds for (2) and the equality $n \equiv |\alpha| = E$ holds for (3).

Although URR can be considered as a variation of floating-point representations with the normalization condition (3), no fixed-length field is used to specify the value of E. Instead, it adopts a prefix-free encoding of the integers in order to implement a variable-length exponent without any additional field to specify the length of the part. A prefix-free encoding of the integers is a special case of representations of the integers [4], which are defined in the following way.

A binary encoding or a representation R of the positive integers, $N^+ = \{1, 2, \cdots\}$, is a bijection of N^+ onto a set C of binary codewords, such that any concatenation of any members of C is uniquely decipherable. The following four conditions are important when we consider an application of representations of the integers to floating-point representation of real numbers.

(C1) *prefix condition:* A representation R is prefix-free if no codeword $R(i) \in C$ is the beginning of another codeword in C.

(C2) *lexicographic order:* A codeword $c = c_1 \ldots c_l$, $c_i \in \{0,1\}$, is said to be lexicographically smaller than a codeword $d = d_1 d_2 \ldots d_m, d_j \in \{0,1\}$, if $c_1 < d_1$; or if $c_i = d_i$ for $i < n$ and $c_n < d_n$ for some $n \leq l, m$; or if $c_i = d_i$ for $1 \leq i \leq l$ and $l < m$. The representation R is said to be of lexicographic order if $R(i)$ is lexicographically smaller than $R(j)$ for $i < j$.

(C3) *completeness:* Let $\{0,1\}^*$ denote the set of all finite strings of symbols, each symbol selected from the set $\{0, 1\}$. A uniquely decipherable set C is said to be complete iff adding any new string $c \in \{0,1\}^*$, $c \notin C$, to C gives a set $C' = C \cup \{c\}$ that is not uniquely decipherable.

(C4) *minimal property:* Let $L(i)$ denote the length of $R(i)$ in bits. The representation R is said to be minimal if $L(i) \leq L(i+1)$ for any $i \in N^+$.

The condition (C1) is a sufficient condition for unique decipherability. It is well known [6] that any uniquely decipherable representation must satisfy the Kraft inequality:

$$\sum_{i=1}^{\infty} 2^{-L(i)} \leq 1.$$

As was mentioned in the previous section, the simplest representation of the integers is *unary encoding*. It is

[2] Knuth refers to [14] as an answer to the exercise.

Table 1: Examples of representation of the integers.

i	$U_0(i)$	$U_{00}(i)$	$U_{011}(i)$	$\Omega(i)$
1	0	0	0	0
2	100	100	10000	100
3	101	101	10001	101
4	11000	110000	100100	110000
5	11001	110001	100101	110001
6	11010	110010	100110	110010
7	11011	110011	100111	110011
8	1110000	1101000	10100000	1101000
9	1110001	1101001	10100001	1101001
10	1110010	1101010	10100010	1101010
15	1110111	1101111	10100111	1101111
16	111100000	1110000000	101010000	11100000000
31	111101111	1110001111	101011111	11100001111

convenient for later discussions to define it as

$$U(i) = 1^{i-1}0,$$

where 1^{i-1} denotes a concatenation of $i-1$ "1" bits, although this definition contrasts with the one used in Section 1. Both the unary encodings obviously satisfy the conditions (C1)–(C4), and the difference between them is not essential.

Now, consider the following transformations for a representation R of the integers.

Type-0 transformation: When an integer $i \in N^+$ has an ordinary binary representation 1α, where α is any string of 0's and 1's, the function R_0 is defined by

$$R_0(i) = \begin{cases} 0 & \text{for } i = 1, \\ 1R(|\alpha|)\alpha & \text{for } i \geq 2. \end{cases}$$

The function R_0 is also a representation of the integers.

Type-1 transformation: When an integer i has an ordinary binary representation 1α, the function R_1 is defined by

$$R_1(i) = R(|\alpha|+1)\alpha.$$

The function R_1 is also a representation of the integers.

If we use U for R in the above transformations, then

$$U_0(i) = \begin{cases} 0 & \text{for } i = 1, \\ 1U(|\alpha|)\alpha = 1^{|\alpha|}0\alpha & \text{for } i \geq 2, \end{cases}$$

and

$$U_1(i) = U(|\alpha|+1)\alpha = 1^{|\alpha|}0\alpha,$$

which mean that $U_0(i) = U_1(i)$ for any $i \in N^+$. In general, results obtained from the type-0 and type-1 transformations are not the same. However, it can be easily shown that if a representation R of the integers satisfies one of the conditions (C1)–(C4), then both the representations R_0 and R_1 also satisfy the same condition. Thus, starting from the unary encoding U, we can get infinitely many representations of the integers satisfying the conditions (C1)–(C4) after repeated applications of the type-0 and/or type-1 transformations.

If τ is a binary string and $\sigma = \tau 0$, then the representation U_σ is a representation which is obtained from U_τ by the type-0 transformation. If $\sigma = \tau 1$, U_σ is a representation which is obtained from U_τ by the type-1 transformation. In Table 1, the examples U_0, U_{00}, and U_{011} are shown in the second, third, and fourth columns, respectively.

3 New Floating-Point Representations Based on Representation of the Integers

In this section, first we limit our definition to $X \geq 1$, and then generalize it to $X < 1$. For $X \geq 1$, a floating-point representation system based on a representation of the integers will be defined by two methods: the exponent-significand pair method and the integer-fraction pair method. The relation of these two methods is summarized in Theorem 1. Since the definitions do not depend on data length, we will not note the length in this section.

3.1 Exponent-Significand Pair Method

In this method, a floating-point system based on a representation of the integers is defined as three fields. Like conventional systems, a number X of the form (1) is represented by its sign S_X, exponent, and significand, as shown in Fig.1. The last two fields specify the values of E and F in (1), respectively. The leading exponent bit is denoted by $S_{E(X)}$, which may be called as the exponent sign bit but does not exactly correspond to the sign of E. The sign bit S_X and the exponent sign bit $S_{E(X)}$ are defined in the following:

$$\begin{array}{ll} \text{if } X < -1, & \text{then } S_X = 1 \text{ and } S_{E(X)} = 0, \\ \text{if } -1 \leq X < 0, & \text{then } S_X = 1 \text{ and } S_{E(X)} = 1, \\ \text{if } 0 < X < 1, & \text{then } S_X = 0 \text{ and } S_{E(X)} = 0, \\ \text{if } 1 \leq X, & \text{then } S_X = 0 \text{ and } S_{E(X)} = 1. \end{array} \quad (6)$$

Even in the case where we restrict ourselves to $X \geq 1$, the value of E depends on the normalization condition

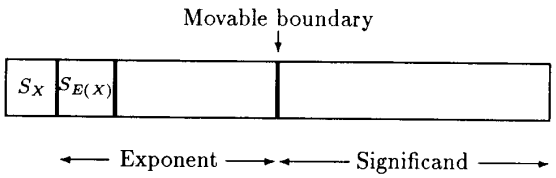

Figure 1: Format of a floating-point representation.

(2) or (3). If we adopt the condition (2), then we have $E \geq 1$ for $X \geq 1$. Under the condition (3), we have $E \geq 0$ for $X \geq 1$. Thus the following two cases are considered.

Condition (2): In this case, E is a positive integer for $X \geq 1$ and we can use a representation of the integers to specify the value of E. Specifically, we put $U_\sigma(E)$ from the 2nd bit on the exponent field. Under the condition (2), F has the form looking like

$$F = 0.1f_2f_3\cdots. \tag{7}$$

Thus we can set the binary string $f_2f_3\cdots$ as the economized significand. Therefore, letting $\mathcal{E}_\sigma^{(2)}(X)$ denote the representation of X for the condition (2), it is encoded as

$$\mathcal{E}_\sigma^{(2)}(X) = 01U_\sigma(E)f_2f_3\cdots \tag{8}$$

for $X \geq 1$, (1), and (7).

Condition (3): In order to specify the value of E, which is greater than or equal to zero, we use

$$U_\sigma^*(E) = \begin{cases} 0 & \text{for } E = 0, \\ 1U_\sigma(E) & \text{for } E \geq 1, \end{cases} \tag{9}$$

for a representation U_σ of the integers. Under the condition (3), F has the form

$$F = 1.f_1f_2\cdots, \tag{10}$$

which is combined with $U_\sigma^*(E)$ to yield the representation. That is, if $\mathcal{E}_\sigma^{(3)}(X)$ denotes the system on the condition (3), then it has the bit pattern

$$\mathcal{E}_\sigma^{(3)}(X) = 01U_\sigma^*(E)f_1f_2\cdots \tag{11}$$

for $X \geq 1$, (1), and (10).

3.2 Integer-Fraction Pair Method

If X has the form (5) with $s = 0$, this method defines a system in our class as the concatenation of the integer and binary fractional parts:

$$\mathcal{F}_\sigma(X) = 01U_\sigma(\lfloor X \rfloor)b_1b_2\cdots \tag{12}$$

for $X \geq 1$. Then we have the following theorem.

Theorem 1 *For any representation U_σ of the integers, we have*

$$\mathcal{F}_{\sigma 0}(X) = \mathcal{E}_\sigma^{(3)}(X), \tag{13}$$
$$\mathcal{F}_{\sigma 1}(X) = \mathcal{E}_\sigma^{(2)}(X) \tag{14}$$

for $X \geq 1$.

Proof: Since $X \geq 1$, X can be represented for $E \geq 0$ as

$$X = 1.f_1f_2\cdots \times 2^E = 1f_1f_2\cdots f_E.f_{E+1}\cdots. \tag{15}$$

Then, we have from (9) and (11)

$$\mathcal{E}_\sigma^{(3)}(X) = \begin{cases} 010f_1f_2\cdots & \text{for } E = 0, \\ 011U_\sigma(E)f_1f_2\cdots & \text{for } E \geq 1. \end{cases}$$

From the definition of $U_{\sigma 0}$ and (15)

$$U_{\sigma 0}(\lfloor X \rfloor) = \begin{cases} 0 & \text{for } E = 0, \\ 1U_\sigma(E)f_1f_2\cdots f_E & \text{for } E \geq 1. \end{cases}$$

It follows from this and (12) that

$$\mathcal{F}_{\sigma 0}(X) = \mathcal{E}_\sigma^{(3)}(X)$$

for $X \geq 1$.

Similarly, for $X \geq 1$ and $E \geq 1$, we have

$$X = 0.1f_2f_3\cdots \times 2^E = 1f_2f_3\cdots f_E.f_{E+1}\cdots. \tag{16}$$

Therefore,

$$U_{\sigma 1}(\lfloor X \rfloor) = U_\sigma(E)f_2f_3\cdots f_E.$$

This and

$$\mathcal{E}_\sigma^{(2)}(X) = 01U_\sigma(E)f_2f_3\cdots$$

prove

$$\mathcal{F}_{\sigma 1}(X) = \mathcal{E}_\sigma^{(2)}(X)$$

for $X \geq 1$. Q.E.D.

We usually interpret any system in the class as a floating-point representation consisting of sign, exponent, and significand. However, it is sometimes more convenient to interpret it as $\mathcal{F}_\sigma$ because we can uniquely identify a system only by σ without referring to the normalization condition. The above simple theorem, which not only gives the generalization of the floating-point and fixed-point interpretations but also provides a basis for the conversion of a system from/to the integer data type, is important also in this respect.

3.3 Extension

We must proceed to the case of $X < 1$. Among possible extensions, we have adopted the following one in order for the class to include URR as a special case. Since the integer-fraction pair method is not effective in $0 < X < 1$, we can extend only the exponent-significand pair method here. In the following, $\overline{S}$ denotes the 1's complement of a binary string S, i.e., if S has the form $s_1s_2\cdots s_N, s_i \in \{0,1\}$, then

$$\overline{S} = \overline{s}_1\overline{s_2}\cdots\overline{s_N},$$

where $\overline{s_i} = 1 - s_i$. Further, for a finite N, $\langle S \rangle$ denotes the 2's complement of S of length N. For an infinite

N, if there exists an integer j such that $s_j = 1$ and $s_{j+1} = s_{j+2} \cdots = 0$, then $\langle S \rangle$ is the concatenation of $\langle s_1 s_2 \cdots s_j \rangle$ and infinite zeros; otherwise, $\langle S \rangle = \overline{S}$.

For $0 < X < 1$, the sign bit S_X and the exponent sign bit $S_{E(X)}$ are both fixed to 0 according to (6). Under the condition (2), noting that the inequalities $0 < X < 1$ correspond to $E \leq 0$, define

$$\mathcal{E}_\sigma^{(2)}(X) = 00\overline{U_\sigma(-E+1)} f_2 f_3 \cdots \qquad (17)$$

for $0 < X < 1$ and (7). For the condition (3), define

$$\mathcal{E}_\sigma^{(3)}(X) = 00\overline{U_\sigma^*(-E-1)} f_1 f_2 \cdots \qquad (18)$$

for $0 < X < 1$ and (10), since $E \leq -1$.

To conclude the definitions, we must deal with the case $X < 0$. For either the condition (2) or (3), the extension is quite straightforward, that is, for $X < 0$

$$\mathcal{E}_\sigma^{(2)}(X) = < \mathcal{E}_\sigma^{(2)}(-X) >, \qquad (19)$$

$$\mathcal{E}_\sigma^{(3)}(X) = < \mathcal{E}_\sigma^{(3)}(-X) >, \qquad (20)$$

which are consistent with (6).

In all the definitions mentioned above, every system is defined as a semi-infinite binary string. However, we can eliminate trailing zeros to yield a finite length representation, and conversely any finite length representation is interpreted as that extended to the right with infinite extra zeros.

4 URR System

Although the URR system [7] has several desirable properties as was mentioned in Section 1, its precision decreases rapidly as a number to be represented increases. Since the original definition of URR uses a bisection method, it is not easy to consider the improvements along the line with the definition. In the proposed class, $\mathcal{E}_0^{(3)}(X)$ is identical with URR. In this section, we will first cite the original definition of URR as it is in order to emphasize the distinction between both the ways of definition. Then, we will give a brief remark on the identity of $\mathcal{E}_0^{(3)}(X)$ with URR.

In the original definition of URR, a binary string S corresponds to a semi-closed interval, say $[a, b)$, which is represented as $I(S)$. This interval is then divided into two intervals $[a, c)$ and $[c, b)$ by a third value c, which is determined by the following four steps.

(a) Rough cut

$$I(1) = [-\infty, 0) \text{ and } I(0) = [0, +\infty).$$

$$I(10) = [-\infty, -1), I(11) = [-1, 0),$$
$$I(00) = [0, 1), \text{ and } I(01) = [1, +\infty).$$

$$I(100) = [-\infty, 2), I(101) = [-2, -1),$$
$$I(110) = [-1, -0.5), I(111) = [-0.5, 0),$$
$$I(000) = [0, 0.5), I(001) = [0.5, 1),$$
$$I(010) = [1, 2), \text{ and } I(011) = [2, +\infty).$$

After the last eight subdivisions, the procedure proceeds to the step (b) for those in which the 2nd bit is equal to the 3rd bit. Otherwise, it proceeds directly to the equal-difference cut (step(d)).

(b) Double-exponential cut

Let $p^+(m)$ and $p^-(m)$ denote 2^{2^m} and 2^{-2^m}, respectively. If $m \geq 0$, the interval is partitioned recursively as:

$$I(10^{m+2}) = [-\infty, -p^+(m)) \text{ at } -p^+(m+1),$$
$$I(11^{m+2}) = [-p^-(m), 0) \text{ at } -p^-(m+1),$$
$$I(00^{m+2}) = [0, p^-(m)) \text{ at } p^-(m+1),$$
$$I(01^{m+2}) = [p^+(m), +\infty) \text{ at } p^+(m+1)$$

with incrementing m by one until one of the following intervals is obtained:

$$I(10^{m+2}1) = [-p^+(m+1), -p^+(m)),$$
$$I(11^{m+2}0) = [-p^-(m), -p^-(m+1)),$$
$$I(00^{m+2}1) = [p^-(m+1), p^-(m)),$$
$$I(01^{m+2}0) = [p^+(m), p^+(m+1)),$$

where neither 0 nor $\pm\infty$ are included at the bounds of the interval.

(c) Equal-ratio cut

The interval $I(S) = [a, b)$ is divided at $\sqrt{ab}$ as

$$I(S0) = [a, \sqrt{ab}) \text{ and } I(S1) = [\sqrt{ab}, b).$$

This is performed m times for the final value of m in step (b). After the completion of this cut, the ratio of the upper bound to the lower bound is 2.

(d) Equal-difference cut

Each interval is divided at $(a+b)/2$ as

$$I(S0) = [a, (a+b)/2) \text{ and } I(S1) = [(a+b)/2, b).$$

This is performed an arbitrary number of times until the desired binary representation is obtained.

It is obvious that $I(S0^k)$ at $k \to \infty$ converges to the lower bound of the interval. This is denoted by $V(S)$. That is, for $I(S) = [a, b)$,

$$V(S) = \lim_{k \to \infty} I(S0^k) = a.$$

For example, the representation of 7.5 in the system is as follows.

- rough cut

$$I(011) = [2, +\infty),$$

- double-exponential cut

$$I(0111) = [4, +\infty),$$

$$I(01110) = [4, 16),$$

– equal-ratio cut

$$I(011100) = [4, 8),$$

– equal-difference cut

$$I(0111001) = [6, 8),$$
$$I(01110011) = [7, 8),$$
$$I(011100111) = [7.5, 8).$$

Consequently, $V(011100111) = 7.5$.

Here, we note how 7.5 is represented in our class. Since $7.5 = 2^2 \times (15/8) = 7 + 0.5$, and 15/8 and 0.5 are represented by 1.111 and 0.1 respectively in the fixed-point binary system, we have

$$\mathcal{E}_0^{(3)}(7.5) = 01 \cdot 1100 \cdot 111 = 011100111$$

and

$$\mathcal{F}_{00}(7.5) = 01 \cdot 110011 \cdot 1 = 011100111$$

because $U_0^*(2) = 1100$ and $U_{00}(7) = 110011$. In these three systems, i.e., URR, $\mathcal{E}_0^{(3)}$, and $\mathcal{F}_{00}$, 7.5 is represented by 011100111.

It is not so difficult to show that the URR system falls on $\mathcal{E}_0^{(3)}(X)$ in the proposed class. As an example, we give a remark only on the case with $X \geq 2$. Let 1α denote the ordinary binary representation of the exponent E of $X \in [p^+(m), p^+(m+1))$. Under the condition (3), the length of α is m bits. Since $U_0^*(E) = 1U_0(E) = 11U(|\alpha|)\alpha = 1^{m+1}0\alpha$, we have

$$\mathcal{E}_0^{(3)}(X) = 01^{m+2}0\alpha f_1 f_2 \cdots \tag{21}$$

for $X \geq 2$ and (10). In (21), it is quite obvious that the string $f_1 f_2 \cdots$ corresponds to the equal-difference cut. Noting that the exponent E of $X \geq 2$ can be represented as $2^m + n$ with $n = 0, 1, \ldots, 2^m - 1$, we see that URR encodes m by the unary encoding in the double-exponential cut and n as an m-bit binary number in the equal-ratio cut. Thus, it follows that the string α in (21) corresponds to the equal-ratio cut of URR.

The identity of URR and $\mathcal{E}_0^{(3)}(X)$ can be proved also in an information theoretical manner. It is known that if a representation of the integers with a given length function satisfies the conditions (C1)–(C3) then the representation is unique. Based on this fact, the present author [16] has shown that URR and $\mathcal{E}_0^{(3)}(X)$ are the same.

5 Representation Errors

When we use fixed length words to implement a system in our class, it is impossible to represent all the real numbers, and therefore representation errors may occur. Let $\Delta(X)$ denote the difference between a value X intended for the representation and the value corresponding to the bit pattern with a fixed length. We will evaluate the error characteristics by the maximum relative error $Er(X) = \Delta(X)/X$, where X corresponds to the mid-point between two discrete points.

In evaluating the error characteristics, we will use the formats $\mathcal{F}_\sigma$ instead of $\mathcal{E}_\sigma^{(2)}$ or $\mathcal{E}_\sigma^{(3)}$, because $\mathcal{F}_\sigma$s are independent of the normalization condition. If 64-bit words are used to represent $\mathcal{F}_\sigma(X)$, we have from (12)

$$\log|Er(X)| = |U_\sigma(\lfloor X \rfloor)| - 63 - \log X, \tag{22}$$

where $\log(\cdot)$ means $\log_2(\cdot)$. We must find an efficient representation U_σ for all over the natural numbers so as to gain higher precision. Of course, however, no representation of the integers is optimal in the sense that it attains the minimum length of representation for any integer.

Let $L_\sigma(i)$ denote the length of $U_\sigma(i)$ in bits. If we set $l = \lfloor \log i \rfloor$, then we have, for example,

$$L_0(i) = 2l + 1, \tag{23}$$

$$L_{00}(i) = \begin{cases} 1 & \text{for } i = 1 \\ l + 2\lfloor \log l \rfloor + 2 & \text{for } i \geq 2, \end{cases} \tag{24}$$

$$L_{011}(i) = l + \lfloor \log(l+1) \rfloor + 2\lfloor \log\log 2(l+1) \rfloor + 1. \tag{25}$$

Define

$$\log^{(1)} i = \log i,$$
$$\log^{(k)} i = \log\log^{(k-1)} i \quad \text{for } k \geq 2$$

and

$$\log^m i = \sum_{k=1}^{m-1} \log^{(k)} i + 2\log^{(m)} i. \tag{26}$$

From the definitions of the type-0 and type-1 transformations, the difference between $L_\sigma(i)$ and $\log^m i$ for $m = |\sigma|$ is upperbounded by a constant which is independent of i. This is denoted by $L_\sigma(i) \simeq \log^m i$. Naturally, this is true of (23), (24), and (25).

In order to evaluate the efficiency of U_σ, let us consider the representations Φ and Ω of the integers which are defined by

$$\Phi(0) = 0,$$
$$\Phi(i) = 1\Phi(|\alpha|)\alpha \quad \text{for } i \geq 1 \tag{27}$$

and

$$\Omega(i) = \Phi(|\alpha|)\alpha \quad \text{for } i \geq 1, \tag{28}$$

where an integer $i \in N^+$ has a binary representation 1α [9]. Some examples of the representation Ω are shown in the rightmost column in Table 1. The representation Ω satisfies the conditions (C1)–(C4) and we can define new number representation systems:

$$\mathcal{E}_\Phi^{(3)}(X) = 01\Phi(E) f_1 f_2 \cdots$$

for $X \geq 1$, (1), and (10), and

$$\mathcal{F}_\Omega(X) = 01\Omega(\lfloor X \rfloor)b_1 b_2 \ldots$$

for $X \geq 1$ and (5). Following the proof of Theorem 1, it can be easily shown that $\mathcal{E}_\Phi^{(3)}(X) = \mathcal{F}_\Omega(X)$ for $X \geq 1$. Since $|\alpha| \simeq \log i$ for an integer i whose binary representation is 1α, and

$$|\Phi(i)| \simeq |\Phi(\log i)| + \log i$$

from (27), the length of Ω is represented by

$$\begin{aligned} |\Omega(i)| &\simeq \log^* i \\ &\triangleq \log i + \log^{(2)} i + \log^{(3)} i + \cdots, \end{aligned}$$

where only the positive terms are included in the sum. Comparing this with (26), we can conclude that, for any representation $\mathcal{F}_\sigma$, there exists a positive number X_m such that higher precision than $\mathcal{F}_\Omega(X)$ is never realized by $\mathcal{F}_\sigma(X)$ for $X \geq X_m$.

This observation, however, is quite unsatisfactory to compare exact relative error characteristics for a practical range of X. To make an exact comparison, we have derived the lengths of representations of the integers and plotted the relative errors as the functions of $\log X$ for $X \geq 1$. Figure 2 shows two examples of $|U_\sigma(\lfloor X \rfloor)| - 63 - \lfloor \log X \rfloor$ and $|\Omega(\lfloor X \rfloor)| - 63 - \lfloor \log X \rfloor$. In order to avoid dense and hard-to-read results, these staircase functions are plotted instead of (22) and other examples are eliminated. Figure 2 shows that the representation $\mathcal{F}_{011}(X) = \mathcal{E}_{01}^{(2)}(X)$ has relatively high precision for medium values of X. In comparison with other systems in the proposed class, $\mathcal{F}_{011}(X)$ has still higher precision for a wide range of X. In fact, if we imagine a representation of the integers which attains the minimum length of the three representations in Fig.2, i.e.,

$$L(i) = \min\{L_{00}(i), L_{011}(i), |\Omega(i)|\},$$

then the corresponding Kraft sum exceeds unity even for the small values of i, that is,

$$\sum_{i=1}^{2^{16}} 2^{-L(i)} > 1.$$

This suggests that it is not easy to improve the precision of the representation $\mathcal{F}_{011}(X)$.

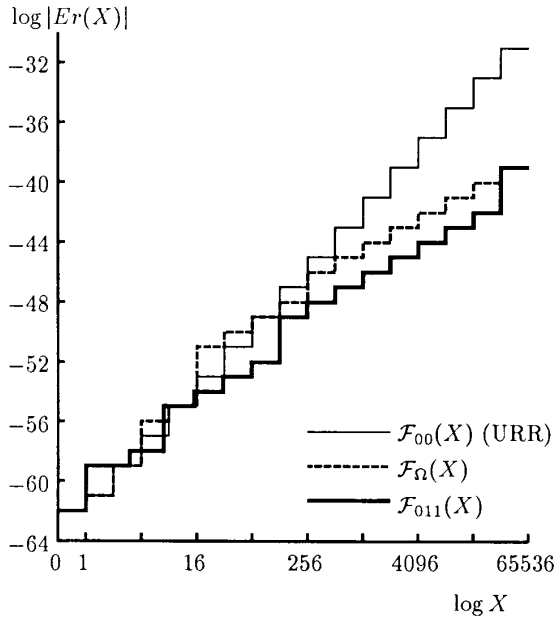

Figure 2: Comparison of representation errors.

6 Conclusion

This paper has presented a class of floating-point number representation systems with self-delimiting variable-length exponent field. Our systems are all available even for those numbers whose exponent are greater than 2^{12}, which we must give up representing in most of conventional systems.

It should be noted that, for a fixed length word, the total number of representable numbers is the same, whether the system is fixed-point, floating-pont, or any other. Thus, in addition to the development of new systems, it is important to consider the efficient and systematic use of the word. To do this and to evaluate the performance, we have incorporated the information theoretical notions such as the Kraft inequality. Consequently, we have succeeded in finding a superior system in precision, because, in the proposed class, the representation errors can be characterized by the length function of an underlying representation of the integers.

In any system of the proposed class, once determined the boundary between the exponent and significand, the arithmetic operations can be performed just as if the operands were conventional floating-point numbers. However, the determination of the boundary is slightly complicated in comparison with the conventional systems. Thus, there still remains much work for future exploration in finding more direct methods of arithmetic operations or in implementing arithmetic by hardware.

Acknowledgment

The author would like to thank Prof. M. Iri of the University of Tokyo for his continued interest and encouragement.

References

[1] Apostolico, A. and Fraenkel, A. S.: "Robust transmission of unbounded strings using Fibonacci representations," *IEEE Trans. Inform. Theory*, Vol.IT-33, No.2, pp.238-245, 1987.

[2] Clenshaw, C. W. and Olver, F. W. J.: "Beyond floating point," *J. ACM,* Vol.31, No.2, pp.319-328, 1984.

[3] Clenshaw, C. W. and Olver, F. W. J.: "Level-index arithmetic operations," *SIAM J. Numer. Anal.,* Vol.24, No.2, pp.470-485, 1987.

[4] Elias, P.: "Universal codeword sets and representations of the integers," *IEEE Trans. Inform. Theory,* Vol.IT-21, No.2, pp.194-203, 1975.

[5] Even, S and Rodeh, M.: "Economical encoding of commas between strings," *Commun. ACM,* Vol.21, No.4, pp.315-317, 1978.

[6] Gallager, R. G.: *Information theory and reliable communication,* Wiley, New York, 1986.

[7] Hamada, H.: "Data length independent real number representation based on double exponential cut," *J. Inform. Process.,* Vol.10, No.1, pp.1-6, 1986.

[8] IBM: *IBM System/370 Principles of Operation,* GA22-7000-8, 1981.

[9] Knuth, D. E.: "Supernatural Numbers," *The Mathematical Gardner* (D. A. Klarner, ed.), Prindle Weber and Schmidt, Boston, pp.310-325, 1980.

[10] Knuth, D. E.: *The Art of Computer Programming,* 2: *Seminumerical Algorithms,* 2nd. ed. Addison-Wesley, Reading, Mass, 1981.

[11] Lozier, D. W. and Olver, F. W. J.: "Closure and precision in level-index arithmetic," *SIAM J. Numer. Anal.,* Vol.27, No.5, pp.1295-1304, 1990.

[12] Matsui, S. and Iri, M.: "An overflow/underflow-free floating-point representation of numbers," *J. Inform. Process.,* Vol.4, No.3, pp.123-133, 1981.

[13] Matula, D. W. and Kornerup, F.: "An order preserving finite binary encoding of the rationals," *Proc. 6th IEEE Symposium on Computer Arithmetic,* pp.201-209, 1983.

[14] Morris, R.: "Tapered floating point: a new floating-point representation," *IEEE Trans. Comput.,* Vol.C-20, No.6, pp.1578-1579, 1971.

[15] Stevenson, D. et al.: "A proposed standard for binary floating-point arithmetic," *IEEE Computer,* Vol.14, No.3, pp.51-62, 1981.

[16] Yokoo, H.: "A class of number representation systems based on multiple exponential cut (in Japanese)," *Trans. of the Institute of Electronics, Information and Communication Engineers,* Vol.J72-A, No.12, pp.1998-2004, 1989.