Journal of Universal Computer Science, vol. 1, no. 7 (1995), 527-547
submitted: 15/12/94, accepted: 26/6/95, appeared: 28/7/95©Springer Pub. Co.

MSB-First Digit Serial Arithmetic[1]

Asger Munk Nielsen
(Dept. of Mathematics and Computer Science
Odense University, Denmark
asger@imada.ou.dk)

Peter Kornerup
(Dept. of Mathematics and Computer Science
Odense University, Denmark
kornerup@imada.ou.dk)

Abstract: We develop a formal account of digit serial number representations by describing them as strings from a language. A prefix of a string represents an interval approximating a number by enclosure. Standard on-line representations are shown to be a special case of the general digit serial representations. Matrices are introduced as representations of intervals and a finite-state transducer is used for mapping strings into intervals. Homographic and bi-homographic functions are used for representing basic arithmetic operations on digit serial numbers, and finally a digit serial representation of floating point numbers is introduced.

Key Words: Computer Arithmetic, On-line Computation, Number Representations, Redundant Digit sets, Continued Fractions, Intervals.
Category: B.2

1 Introduction

A number is usually represented as a string of digits belonging to some digit set Σ. The number representation specifies a function that maps the string to its value. In the context of this paper a *digit serial number representation* will be a representation, that will allow us to gradually calculate the value of a number, by starting with an empty string and then iteratively calculate better approximations of the number, from increasingly longer prefixes of the string. A prefix of a string represents an approximation of the number, with an associated variance. The variance is determined by the possible extensions of the prefix, and the prefix can be thought of as a representation of an interval that includes the number being approximated. Concatenating the next digit of the string to the prefix, yields a more precise representation of the number, and will effectively narrow down the variance interval. When the whole string has been constructed by successive concatenation (if the string is finite and terminated), the interval will have been narrowed down to a degenerate interval of length zero, i.e. the value of the number. Infinite strings represent computable reals as limits of contracting intervals. A prefix of zero length is a prefix of any number, and has an associated variance interval, with endpoints determined by the largest and

[1] This work has been supported by The Danish Research Councils under the grant no. 5.21.08.02.

smallest representable numbers, this interval will be denoted as *the total variance interval.*

In this paper we will take a formal language approach to describing digit serial operands (e.g. operands are taken to be strings over some alphabet), with this approach the mapping from strings to intervals can be described by means of a finite-state transducer. In [Section 2] we develop a formal account of digit serial number representations, digit serial computable functions and the relation to the equivalent on-line definitions. In [Section 3] a computational model for a function of one variable, based on a matrix representation is developed. In [Section 4], this model is generalized to a cube structure, capable of modelling a function of two variables. In [Section 5] we introduce a representation of digit serial floating point numbers, thereby expanding the set of representable numbers, and discuss normalization problems in on-line and digit serial computation.

2 Digit Serial Representations of Numbers

We will now develop a formal account of digit serial number representations.

Definition 1. Digit Serial Number Representation
Let Σ be a digit set (finite or infinite), τ a terminal symbol, $\mathbb{I}$ the set of intervals with endpoints in $\mathbb{R}$, and $\varphi : \mathbb{S} \to \mathbb{I}$ is a function mapping a string from $\mathbb{S} = \Sigma^* \cup \Sigma^+ \cdot \{\tau\}$ to its associated interval. Then the tuple (Σ, τ, φ) is called a **digit serial number representation** if it obeys the following axioms:

(Contraction axiom)

$$\forall s \in \mathbb{S} : (x \sqsubset s) \wedge (y \sqsubset s) \wedge (||x|| < ||y||) : \varphi(x) \supset \varphi(y) \supseteq \varphi(s)$$

(Limit axiom)

$$\forall s \in \Sigma^* : \lim_{||x|| \to \infty, x \sqsubset s} \Delta(\varphi(x)) = 0$$

(Extensibility axiom)

$$\text{If } y \in \Sigma^* \text{ and } a \in \varphi(y) \text{ then } \exists d \in \Sigma : a \in \varphi(y \cdot d)$$

(Termination axiom)

$$\forall s \in \Sigma^+ : \Delta(\varphi(s \cdot \tau)) = 0$$

Here $x \sqsubset s$ denotes that x is a prefix of the string s, ϵ symbolizes the empty string, $\varphi(\epsilon)$ is the total variance interval, $||x||$ denotes the length of the string x, and the interval norm function $\Delta : \mathbb{I} \to \mathbb{R}$ is defined below.

We will allow intervals of the form $[p_1, p_2]$ where $p_1 > p_2$ specifies an interval that include both plus and minus infinity. If $p_1 = p_2$ the interval is a degenerate interval representing a point. An ordinary interval will be designated by the symbol I_{normal}, and the type that includes plus and minus infinity by the symbol I_∞. In order to treat ∞ as any other number, we use the stereographic

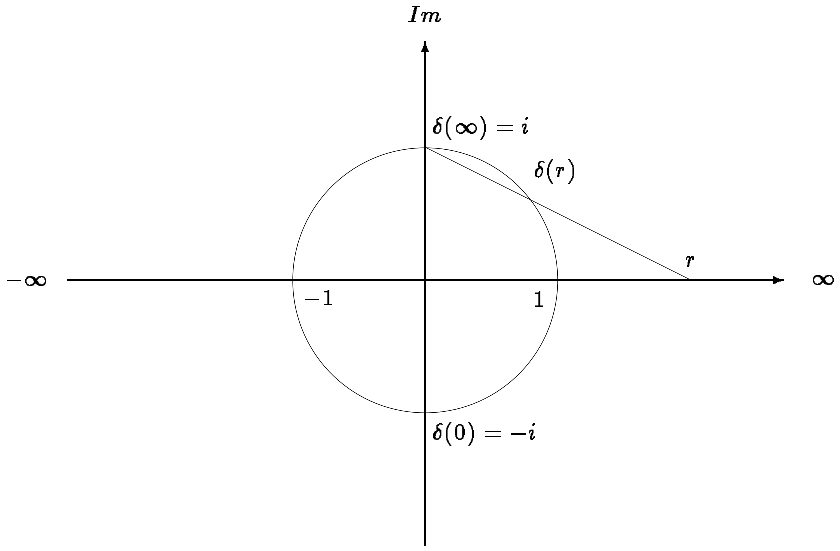


Figure 1: Stereographic representation of $\mathbb{R}$.

representation of the real line as defined in [Vuillemin 90] [see Fig. 1]. With this representation minus and plus infinity are treated as one point, and the interval $| x |> 1$ is the upper part of the circle, and the reciprocal of this interval (e.g. $| x |< 1$) is the lower part. We will adopt an alternative definition of the width of an interval $[x, y]$ by taking $\Delta : \mathbb{I} \to \mathbb{R}$ to be the length of the cord joining $\delta(x)$ and $\delta(y)$. With this definition it is possible for an interval including infinity to have finite length, making the limit axiom attainable in such special cases. Notice also that with this distance function it is possible to extend the concept of continuity for functions defined on intervals of the form I_∞.

Many well known number representations can indeed be placed within the digit serial number representation framework.

Example 1. Redundant Binary Fixed-Point Numbers.
Let $\Sigma = \{-1, 0, 1\}$, and define φ as:

$$\varphi(d_1 \cdot d_2 \cdots d_k) = [-2^{-k} + \sum_{i=1}^{k} d_i 2^{-i}, 2^{-k} + \sum_{i=1}^{k} d_i 2^{-i}]$$
$$\varphi(d_1 \cdot d_2 \cdots d_n \cdot \tau) = [\sum_{i=1}^{n} d_i 2^{-i}, \sum_{i=1}^{n} d_i 2^{-i}]$$
$$\varphi(\epsilon) = [-1, 1]$$

Notice that the termination symbol could alternatively be defined as an infinite string of zeros.

Example 2. Continued Fractions.
Let $\Sigma = \mathbb{N}^+$. The continued fraction $\frac{p_k}{q_k} = a_1 + \frac{1}{a_2 + \cdots \frac{1}{a_k}}$ can be calculated by the convergent recurrence defined by: $p_0 = q_{-1} = 1$, $p_{-1} = q_0 = 0$, $p_k = a_k p_{k-1} + p_{k-2}$ and $q_k = a_k q_{k-1} + q_{k-2}$, where $c_k = \frac{p_k}{q_k}$ is the k'th *convergent*. From

the theory of continued fractions it is known that odd numbered convergents are smaller than even numbered convergents, thus we can define φ as:

$$\varphi(a_1 \cdot a_2 \cdots a_{2k}) = \left[\frac{p_{2k}+p_{2k-1}}{q_{2k}+q_{2k-1}}, \frac{p_{2k}}{q_{2k}}\right[$$
$$\varphi(a_1 \cdot a_2 \cdots a_{2k+1}) = \left]\frac{p_{2k+1}}{q_{2k+1}}, \frac{p_{2k+1}+p_{2k}}{q_{2k+1}+q_{2k}}\right]$$
$$\varphi(\epsilon) = [1, \infty[.$$

The endpoints of the interval are calculated by letting the next digit to be seen take on its smallest and largest possible value, i.e. one and infinity. The termination symbol could be defined as an infinite string of infinity symbols. Notice that since only positive terms are allowed, this number system is not redundant, if negative terms are allowed ($\Sigma = \mathbb{Z}$) we will have redundant continued fractions, as is easily seen from the following example: The rational number $\frac{2}{7}$ can be expressed as $\frac{1}{3+\frac{1}{2}}$ or $\frac{1}{4+\frac{1}{-2}}$.

Some functions will enable us to compute the value of the function in a digit serial manner.

Definition 2. Digit Serially Computable Function.
A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is *Digit Serially Computable* on the interval $I \subseteq \varphi(\epsilon)$, for a digit serial number representation (Σ, τ, φ) if

$$\forall x \in I : f(x) \subseteq \varphi(\epsilon),$$

and for all infinite strings $x \in \Sigma^*$, with $\varphi(x) \subseteq I$, there exists an infinite string $y = y_1 y_2 y_3 \cdots \in \Sigma^*$ such that

$$f(\varphi(x)) = \varphi(y), \text{ and}$$
$$\forall p > 0 : \exists u : u \sqsubset x : f(\varphi(u)) \subseteq \varphi(y_1 y_2 \cdots y_p)$$

Notice that with this definition, we may have to read an arbitrary number of input digits between the generation of two output digits, thus the output delay can be arbitrarily large, that is the output is not synchronized with the input. It is well known that for special number representations, we can for certain functions compute the output digits with a constant output delay of one, once the first output digit has been computed, i.e. the output is synchronized with the input.

Definition 3. On-Line Computable Function.
A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is *On-Line Computable* on the interval $I \subseteq \varphi(\epsilon)$, for a digit serial number representation (Σ, τ, φ) iff there exists an integer δ such that:

$$\forall x \in I : f(x) \subseteq \varphi(\epsilon)$$

and $\forall x = x_1 x_2 x_3 \cdots \in \Sigma^*$, with $\varphi(x) \in I$, then for all $p \geq 1$ we have:

$$f(\varphi(x_1 x_2 \cdots x_{p+\delta-1})) \subseteq \varphi(y_1 y_2 \cdots y_{p-1})$$
$$\Downarrow$$
$$\exists y_p \in \Sigma : f(\varphi(x_1 x_2 \cdots x_{p+\delta})) \subseteq \varphi(y_1 y_2 \cdots y_p)$$

The integer δ is defined as the *On-Line Delay*.

Theorem 4. *Any function that is Digit Serially Computable on an interval I, is continuous*[2] *on this interval.*

Proof. Let $x = \varphi(r) \in I$ and $y = f(x) = \varphi(s)$, with $r = x_1x_2\cdots \in \Sigma^*$ and $s = y_1y_2\cdots \in \Sigma^*$. Given any $\alpha > 0$, we can by the limit axiom, chose a p such that

$$\Delta(\varphi(y_1y_2\cdots y_p)) < \alpha$$

then by the definition of digit serial computability we have

$$\exists u \sqsubset r : f(\varphi(u)) \subseteq \varphi(y_1y_2\cdots y_p).$$

Now take $v \in \varphi(u)$ and since $x \in \varphi(u)$ we conclude:

$$\Delta([x,v]) \leq \Delta(\varphi(u)) = \beta$$

for some $\beta > 0$. Since $f(v), f(x) \in f(\varphi(u))$ we likewise conclude:

$$\Delta([f(x), f(v)]) \leq \Delta(f(\varphi(u)) \leq \Delta(\varphi(y_1y_2\cdots y_p) < \alpha.$$

Thus

$$\forall\alpha > 0 : \exists\beta > 0 : \Delta([x,v]) < \beta \Rightarrow \Delta([f(x), f(v)]) < \alpha$$

hence f is continuous on I.

Note that the class of On-Line computable functions is a subclass of the digit serially computable functions, thus an On-Line computable function is also continuous, as stated in [Duprat, Herreros and Muller 89]. It is straightforward to generalize the definition of digit serial and on-line computability to functions of two variables. Such functions are of paramount importance, since these functions include dyadic operations like addition and multiplication. When generalizing the definitions to two variables, we have the choice of requiring synchronization between the input variables, as is the case with on-line computable functions, or not to require synchronization. Thus there are three basic classes of MSB-first digit serial algorithms, the most general being the class where no synchronization is enforced, a less general subclass where the input is synchronized and the third subclass containing algorithms with both input and output synchronized. The following theorem proves the converse statement, of that formulated in [Theorem 4], but for a function of two variables.

Theorem 5. *If the function $f : \mathbb{R}^2 \to \mathbb{R}$ is continuous on the domain $I_x \times I_y$, and if for all $x \in I_x$ and $y \in I_y$ we have $f(x,y) \in \varphi(\epsilon)$, then f is digit serially computable for any Digit Serial Number Representation.*

Proof. Let $x, y \in \Sigma^*$ be two infinite strings, with $\varphi(x) \in I_x$ and $\varphi(y) \in I_y$. By extensibility we conclude that there exists an infinite string $z = z_1z_2z_3\cdots \in \Sigma^*$ such that $f(\varphi(x), \varphi(y)) = \varphi(z)$. Because of the contraction axiom, we must have:

$$\varphi(z_1) \supseteq \varphi(z_1z_2) \supseteq \cdots \supseteq \varphi(z_1z_2\cdots z_p) \supseteq \cdots \supseteq \varphi(z)$$

and since $x \sqsubset x$ and $y \sqsubset y$ we get by the continuity of f:

$$\forall p > 0 : \exists u \sqsubset x \wedge \exists v \sqsubset y : f(\varphi(u), \varphi(v)) \subseteq \varphi(z_1z_2\cdots z_p).$$

Thus f is digit serially computable.

[2] Continuity is here defined in terms of the stereographic distance measure Δ.

The following algorithm describes how to compute the output of a function in a digit serial manner. The notation: $BooleanExpr1 : ExprSeq1[]BooleanExpr2 : ExprSeq2$; means if only one of the boolean expressions are true then execute the corresponding statement sequence. If both expressions are true, then according to some rule execute one or both statement sequences. Such a rule might be: Select one of the sequences nondeterministically, or alternatively: Execute both in parallel.

Algorithm 1. *Basic Digit Serial Function Evaluation.*

Stimulus: $f : \mathbb{R}^2 \to \mathbb{R}$, $x, y \in \Sigma^* \cdot \{\tau\}$, $n \in \mathbb{N}$.
Response: $z \in \Sigma^n \cup \Sigma^* \cdot \{\tau\}$.
Method:

$p \leftarrow 1$;
$i, j \leftarrow 0$;
while $p \leq n$ *and* $f(\varphi(x_1 \cdots x_i), \varphi(y_1 \cdots y_j)) \neq \varphi(z_1 \cdots z_{p-1} \cdot \tau)$ *do*
    if $\exists z_p \in \Sigma : f(\varphi(x_1 \cdots x_i), \varphi(y_1 \cdots y_j)) \subseteq \varphi(z_1 \cdots z_p)$ *then*
        output z_p;
        $p \leftarrow p + 1$;
    else
        $x_i \neq \tau : i \leftarrow i + 1;\ x_i \leftarrow$ *input from* x
            []
        $y_j \neq \tau : j \leftarrow j + 1;\ y_j \leftarrow$ *input from* y
    end;
end;
if $f(\varphi(x_1 x_2 \cdots x_i), \varphi(y_1 y_2 \cdots y_j)) = \varphi(z_1 z_2 \cdots z_{p-1} \cdot t)$ *then*
    output τ;
end;

□

It is well known that a redundant representation is necessary for a function in general to be on-line computable. The following theorem formalizes the on-line property for the signed-digit radix two representation, and can easily be generalized to other redundant radix representations.

Theorem 6. *If the function $f : \mathbb{R}^2 \to \mathbb{R}$ is digit serially computable for the signed-digit radix two representation with $\Sigma = \{-1, 0, 1\}$ on the domain $I = I_x \times I_y$, where I_x and I_y are of the interval type I_{normal}, and if the partial derivatives of f are bounded on this domain, then there exists a finite integer constant: δ such that f is on-line computable with on-line delay δ.*

Proof. Suppose that the $p - 1$ most significant digits of $z = f(x, y)$ have been calculated from $x_1, x_2, \ldots, x_{p-1+\delta}$ and $y_1, y_2, \ldots, y_{p-1+\delta}$ such that

$$f(\varphi(x_1 x_2 \cdots x_{p-1+\delta}), \varphi(y_1 y_2 \cdots y_{p-1+\delta})) \subseteq \varphi(z_1 z_2 \cdots z_{p-1})$$

Define the domain $D = [a, b] \times [c, d] \subseteq I$ by:

$$a = 0.x_1 x_2 \cdots x_{p-1+\delta} x_{p+\delta} \bar{1}\bar{1}\bar{1} \ldots = -2^{-p-\delta} + \sum_{i=1}^{p+\delta} x_i 2^{-i}$$

$$b = 0.x_1x_2\cdots x_{p-1+\delta}x_{p+\delta}111\ldots = 2^{-p-\delta} + \sum_{i=1}^{p+\delta} x_i 2^{-i}$$

$$c = 0.y_1y_2\cdots y_{p-1+\delta}y_{p+\delta}\bar{1}\bar{1}\bar{1}\ldots = -2^{-p-\delta} + \sum_{i=1}^{p+\delta} y_i 2^{-i}$$

$$d = 0.y_1y_2\cdots y_{p-1+\delta}y_{p+\delta}111\ldots = 2^{-p-\delta} + \sum_{i=1}^{p+\delta} y_i 2^{-i}.$$

The possible variation of $f(x,y)$ within D is easily estimated by:

$$\begin{aligned}|f_{max} - f_{min}| &= \left|\max_{(x,y)\in D}\{f(x,y)\} - \min_{(x,y)\in D}\{f(x,y)\}\right| \\ &\le |b-a|\max_{(x,y)\in I}\left\{\left|\frac{\partial f(x,y)}{\partial x}\right|\right\} + |d-c|\max_{(x,y)\in I}\left\{\left|\frac{\partial f(x,y)}{\partial y}\right|\right\} \\ &= 2^{-p-\delta+1}\left(\max_{(x,y)\in I}\left\{\left|\frac{\partial f(x,y)}{\partial x}\right|\right\} + \max_{(x,y)\in I}\left\{\left|\frac{\partial f(x,y)}{\partial y}\right|\right\}\right)\end{aligned}$$

Take $\delta \ge \left\lceil log_2\left(\max_{(x,y)\in I}\left\{\left|\frac{\partial f}{\partial x}\right|\right\} + \max_{(x,y)\in I}\left\{\left|\frac{\partial f}{\partial y}\right|\right\}\right)\right\rceil + C$, where C is a positive integer constant. Since δ is an integer we get:

$$2^{\delta-C} \ge \left(\max_{(x,y)\in I}\left\{\left|\frac{\partial f}{\partial x}\right|\right\} + \max_{(x,y)\in I}\left\{\left|\frac{\partial f}{\partial y}\right|\right\}\right)$$

and if $C \ge 1$ we get $|f_{max} - f_{min}| \le 2^{-p}$. Now

$$\begin{aligned}f(\varphi(x_1\cdots x_{p+\delta}), \varphi(y_1\cdots y_{p+\delta})) &\subseteq f(\varphi(x_1\cdots x_{p-1+\delta}), \varphi(y_1\cdots y_{p-1+\delta})) \\ &\subseteq \varphi(z_1\cdots z_{p-1}) \\ &= \left[-2^{-p+1} + \sum_{i=1}^{p-1} z_i 2^{-i}, 2^{-p+1} + \sum_{i=1}^{p-1} z_i 2^{-i}\right]\end{aligned}$$

Consulting [Fig.2] immediately leads to the conclusion that no matter where the interval $f(\varphi(x_1\cdots x_{p+\delta}), \varphi(y_1\cdots y_{p+\delta}))$ is placed within $\varphi(z_1\cdots z_{p-1})$ there exists a $z_p \in \Sigma$ such that

$$f(\varphi(x_1\cdots x_{p+\delta}), \varphi(y_1\cdots y_{p+\delta})) \subseteq \varphi(z_1\cdots z_p).$$

Since the function is digit serially computable, for $p = 1$ we have that

$$\begin{aligned}f(\varphi(x_1\cdots x_{p+\delta}), \varphi(y_1\cdots y_{p+\delta})) &\subseteq \varphi(\epsilon) = [-1, 1] \\ &= \left[-2^{-p+1} + \sum_{i=1}^{p-1} z_i 2^{-i}, 2^{-p+1} + \sum_{i=1}^{p-1} z_i 2^{-i}\right]\end{aligned}$$

with a similar argument as the one used above, we may conclude that in the base case $p = 1$ we can find a z_1 such that $f(\varphi(x_1\cdots x_{\delta+1}), (y_1\cdots y_{\delta+1}) \subseteq \varphi(z_1)$. Thus if $\delta + 1$ input digits have been consumed from input x and y, an output digit can be derived in each subsequent iteration, that is $f(x,y)$ is on-line computable.

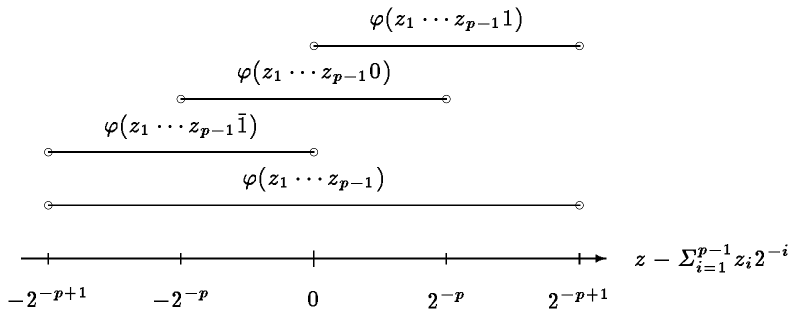


Figure 2: The variation of the output $(z = f(x, y))$.

[Theorem 6] tells us, that for radix two with $\Sigma = \{-1, 0, 1\}$ we can modify [Algorithm 1] to withhold production of output digits until $\delta + 1$ input digits have been read from inputs x and y, and in the subsequent iterations one output digit is generated whenever one input digit is consumed from both operands, this modified algorithm will be an on-line algorithm. This basic algorithm can intuitively be brought to compute the output of an on-line computable function, with an on-line delay that is equal to the theoretical lower bound.

3 Homographic Matrix Coding

As discussed, digit serial numbers can effectively be described with the aid of intervals. In this section we will show, that the intervals corresponding to a digit string can be represented by matrices.

Notation 7. Interval-matrix and Matrix-interval Homomorphism.
An *interval matrix* $A = \begin{bmatrix} p & s \\ q & t \end{bmatrix}$ $p, q, r, s \in \mathbb{Z}$, is a matrix representing an interval: $\Phi(A)$, with endpoints $\frac{p}{q}$ and $\frac{s}{t}$, where the homomorphism $\Phi : \mathbb{M}_{2\times 2} \to \mathbb{I}$ is dependent on the representation. In some cases it is sufficient to define Φ as

$$\Phi(A) = \left[\frac{p}{q}, \frac{s}{t}\right]$$

Notation 8. Digit-set Isomorphism.
Let $\eta : \Sigma \cup \{\tau, \epsilon\} \to \Sigma_M \cup \{T, E\}$ be an isomorphism that maps digits, an empty string or the termination symbol into 2×2 matrices. $T = \eta(\tau)$ is denoted as the *termination matrix* and $E = \eta(\epsilon)$ as *the total variance matrix.*

For a specific digit serial representation, it is now possible to construct a matrix product representation of a digit serial number.

Definition 9. Homographic Matrix-coding.
If (Σ, τ, φ) is a digit serial number representation, η a digit-set isomorphism, and Φ a matrix-interval homomorphism, then $C = (\Sigma, \eta, \Phi)$ is a *homographic matrix coding* if the following is obeyed:

$\Phi(E) = \varphi(\epsilon)$, and
If $\Phi(EX_1X_2\cdots X_p) = \varphi(x_1x_2\cdots x_p)$, $x_i \in \Sigma$ and $X_i = \eta(x_i)$ for $i = 1, 2, \ldots, p$, then

$$\Phi(EX_1X_2\cdots X_pD) = \varphi(x_1x_2\cdots x_pd)$$

and

$$\Phi(EX_1X_2\cdots X_pT) = \varphi(x_1x_2\cdots x_p\tau)$$

where $d \in \Sigma,\ D = \eta(d)$.

The termination matrix acts in the same way as the termination symbol, in the sense that it turns some interval matrix into an interval matrix representing a degenerate interval.

Example 3. Continued Fractions ($\Sigma = \mathbb{N}^+, \varphi(\epsilon) = [1, \infty[$)

$$\begin{aligned}
\begin{bmatrix} p_k + p_{k-1} & p_k \\ q_k + q_{k-1} & q_k \end{bmatrix} &= \begin{bmatrix} a_kp_{k-1} + p_{k-2} + p_{k-1} & a_kp_{k-1} + p_{k-2} \\ a_kq_{k-1} + q_{k-2} + q_{k-1} & a_kq_{k-1} + q_{k-2} \end{bmatrix} \\
&= \begin{bmatrix} p_{k-1} + p_{k-2} & p_{k-1} \\ q_{k-1} + q_{n-2} & q_{k-1} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ a_k & a_k - 1 \end{bmatrix} \\
&= \begin{bmatrix} p_0 + p_{-1} & p_0 \\ q_0 + q_{-1} & q_0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ a_1 & a_1 - 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & 1 \\ a_k & a_k - 1 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ a_1 & a_1 - 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & 1 \\ a_k & a_k - 1 \end{bmatrix} \\
&= EA_1A_2\cdots A_k.
\end{aligned}$$

Now taking the termination matrix as: $T = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$ we see that $C_{cf} = (\Sigma, \eta, \Phi)$ is a homographic matrix coding, where $\eta(a_i) = E^{-1}\begin{bmatrix} a_i & 1 \\ 1 & 1 \end{bmatrix} E = \begin{bmatrix} 1 & 1 \\ a_i & a_i - 1 \end{bmatrix}$. The matrix-interval homomorphism Φ, is as defined previously, but with the end-points interchanged when k is odd. Note that the convergents can be computed as:

$$\begin{bmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{bmatrix} = \begin{bmatrix} p_{n-1} & p_{n-2} \\ q_{n-1} & q_{n-2} \end{bmatrix} \begin{bmatrix} a_n & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} a_1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} a_2 & 1 \\ 1 & 1 \end{bmatrix} \cdots \begin{bmatrix} a_n & 1 \\ 1 & 1 \end{bmatrix}.$$

Example 4. Redundant Binary Fixed-Point Numbers. ($\Sigma = \{-1, 0, 1\}, \varphi(\epsilon) =]-1, 1[$)
It is easily verified that choosing η in the following way makes $C_{fp} = (\Sigma, \eta, \Phi)$ a homo-graphic matrix coding:

$$\eta(\epsilon) = E = \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\eta(b) = 2E^{-1}\begin{bmatrix}1 & b\\ 0 & 2\end{bmatrix}E = \begin{bmatrix}-b+3 & -b+1\\ b+1 & b+3\end{bmatrix} \text{ for all } b \in \Sigma$$
$$\eta(t) = T = \begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}.$$

Note that the sum $\frac{p_n}{q_n} = \sum_{i=1}^{n} b_i 2^{-i}$ can be computed as:

$$\begin{bmatrix}1 & p_n\\ 0 & q_n\end{bmatrix} = \begin{bmatrix}1 & b_n + 2p_{n-1}\\ 0 & 2q_{n-1}\end{bmatrix} = \begin{bmatrix}1 & p_{n-1}\\ 0 & 2q_{n-1}\end{bmatrix}\begin{bmatrix}1 & b_n\\ 0 & 2\end{bmatrix} = \begin{bmatrix}1 & b_1\\ 0 & 2\end{bmatrix}\cdots\begin{bmatrix}1 & b_n\\ 0 & 2\end{bmatrix}.$$

In this paper we have taken a formal language approach, when describing the behaviour of digit serial arithmetic, it therefore falls natural to implement the generation of the appropriate transformations as a finite-state transducer. The automaton acts as a translator in the sense, that it translates input symbols into matrices, in such a way that when these matrices are multiplied in the order they are received, the product will be an interval matrix representing the interval corresponding to the input string. From the transformations and the knowledge of how one symbol can proceed another symbol, a state table can be constructed.

Example 5. Finite-state Transducer for Redundant Base 2 Fixed-Point Digit-Serial Numbers.
A table [see Tab. 1] can be constructed from the previously developed transformations, based on this table a finite-state transducer can be devised [see Fig. 3].

State	Symbol	Next-state	$(\alpha, \beta, \gamma, \delta)$	Interval Type
Start	–	Fraction	$(-1, 1, 1, 1)$	I_{normal}
Fraction	$\bar{1}$	Fraction	$(4, 2, 0, 2)$	I_{normal}
	0	Fraction	$(3, 1, 1, 3)$	I_{normal}
	1	Fraction	$(2, 0, 2, 4)$	I_{normal}
	τ	Terminated	$(1, 1, 1, 1)$	*point*
Terminated	–	Terminated	$(1, 0, 0, 1)$	*point*

Table 1: Transition/output table for redundant base 2 fixed-point digit-serial numbers.

The table describes the translation of an operand in the form of a digit string into a sequence of matrices. The operand has a state variable, initially assigned the state labeled *start*, associated with it. Digits are read one by one from the operand, upon reading the symbol s, produce the transformation matrix corresponding to the symbol s in the present state. When the matrix multiplication has been completed move to the state designated by *next-state*. The column labeled *interval type* conveys information about how the interval matrix generated by the subsequent matrix multiplications is to be interpreted, I_{normal} designates an ordinary interval that does not include both plus and minus infinity, *point* designates a degenerate interval representing a point.

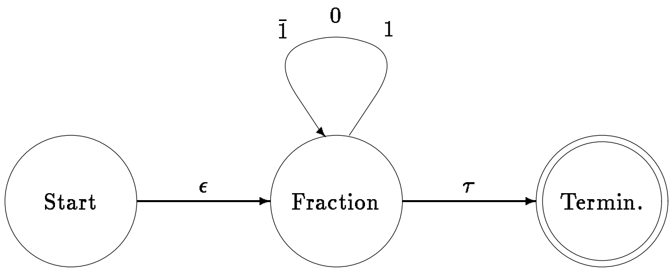


Figure 3: State machine for radix 2 fixed-point numbers.

We now investigate how to compute the output of a function of the form: $f(x) = \frac{ax+b}{cx+d}$ in a digit serial manner. The parameters $a, b, c, d \in \mathbb{Z}$ are chosen such that f is digit serially computable. Evaluating the function at a point $x = \frac{p}{q}$ yields $f(\frac{p}{q}) = \frac{a\frac{p}{q}+b}{c\frac{p}{q}+d} = \frac{ap+bq}{cp+dq}$, and equivalently for $x = \frac{s}{t}$ we get $f(\frac{s}{t}) = \frac{as+bt}{cs+dt}$. Now since the function is monotonic (assumed increasing), when evaluating the function on the interval $[\frac{p}{q}, \frac{s}{t}]$ we get a new interval $[\frac{ap+bq}{cp+dq}, \frac{as+bt}{cs+dt}]$. This computation can effectively be modeled by matrix multiplication.

Definition 10. Homographic Function Matrix
The matrix $F = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ $a, b, c, d \in \mathbb{Z}$, is defined as a matrix representation of the digit serially computable function $f(x) = \frac{ax+b}{cx+d}$, in the sense that if $\Phi(EX_1X_2 \cdots X_p) = \varphi(x_1x_2 \cdots x_p)$ then $\Phi(FEX_1X_2 \cdots X_p) = f(\varphi(x_1x_2 \cdots x_p))$

Notice that F is defined in such a way that multiplying from the right with an interval matrix yields a new interval matrix. That is

$$FA = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} p & s \\ q & t \end{bmatrix} = \begin{bmatrix} ap+bq & as+bt \\ cp+dq & cs+dt \end{bmatrix}. \quad (1)$$

Furthermore if F, G are matrices representing the functions f and g, then FG represents $f \circ g$ and F^{-1} represents f^{-1}.

Since the two different ways of representing digit serial numbers are equivalent, it is straightforward to state analogous definitions and proofs to those of [Section 2] in terms of a homographic matrix coding. For instance the condition $f(\varphi(x_1x_2 \cdots x_i)) \subseteq \varphi(y_1y_2 \cdots y_p)$ can be stated as:

$$\Phi(FEX_1X_2 \cdots X_i) \subseteq \Phi(EY_1Y_2 \cdots Y_p) \quad (2)$$

Observation 11. Let F be a function matrix representing a function that is continuous and monotone on an interval represented by the matrix B, then $\Phi(A) \subseteq \Phi(B) \iff \Phi(FA) \subseteq \Phi(FB)$

Assuming [3] that for all $Y_j \in \Sigma_M$ we have that Y_j^{-1} represents a function that is continuous and monotone on the interval represented by $\Phi(Y_j \cdots Y_p)$, and similarly assume that E^{-1} represents a continuous monotone function on the interval $\Phi(E)$. From (2) and [Observation 11] we arrive at:

$$\Phi(Y_{p-1}^{-1} \cdots Y_2^{-1} Y_1^{-1} E^{-1} F E X_1 X_2 \cdots X_i) \subseteq \Phi(Y_p). \tag{3}$$

Similarly the termination test: $f(\varphi(x_1 x_2 \cdots x_i) = \varphi(y_1 y_2 \cdots y_{p-1} \cdot t)$ can be stated as:

$$\Phi(Y_{p-1}^{-1} \cdots Y_2^{-1} Y_1^{-1} E^{-1} F E X_1 X_2 \cdots X_i) = \Phi(T). \tag{4}$$

Thus an algorithm for computing $f(x)$ in some point $x = \varphi(x_1 x_2 \cdots x_n t)$, could proceed as follows: Let F be the function on matrix form. Transform the function matrix as: $F \leftarrow E^{-1} F E$. Consume input by setting: $F \leftarrow F X_i$, and when possible produce output by $F \leftarrow Y_p^{-1} F$. Where $X_i = \eta(x_i)$ and $Y_i = \eta(y_i)$.

An algorithm based on an automaton could alternatively be used, by assigning a state variable to both the input and output variable. In the case of *input*, upon receiving the symbol s do the multiplication $F \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$ with the matrix corresponding to the symbol s in the present state, then move to the next state. When performing *output*, we first need to check if output can be generated. This is done by checking if for any of the symbols in the present state we have:

$$\left[\frac{\alpha}{\gamma}, \frac{\beta}{\delta}\right] \supseteq \Phi(F).$$

If this holds for some symbol then the symbol is output, and the multiplication

$$\begin{bmatrix} \delta & -\beta \\ -\gamma & \alpha \end{bmatrix} F$$

is performed. The range of the function evaluated on the interval corresponding to the prefix of the operand that has been seen so far (specified by $\Phi(F)$), can be computed by knowing the interval type of the operand, and by the knowledge of whether the function is monotonically increasing or decreasing, which is computable from the sign of the determinant of F.

4 The Cube

In this section we consider how to evaluate a function of two variables. The collective variance associated with two variables is a Cartesian product of two intervals. The variance of the function amounting from evaluating the function in all points of the domain will be characterized by a surface in $\mathbb{R}^3$. In the following we will consider a generalization of the homographic function $f(x) = \frac{ax+b}{cx+d}$ to a similar function of two variables (a bi-homographic function):

$$f(x_2, x_1) = \frac{a_{111} x_2 x_1 + a_{101} x_2 + a_{011} x_1 + a_{001}}{a_{110} x_2 x_1 + a_{100} x_2 + a_{010} x_1 + a_{000}}, \; a_i \in \mathbb{Z} \tag{5}$$

[3] This is indeed not an unrealistic assumption, as readily verified from the previous examples.

With an appropriate choice of coefficients in (5), the function can compute the standard arithmetic operations $x+y$, $x-y$, $x \cdot y$ and x/y.

A matrix representation will not suffice for a function of two variables, but as we will show, such a function can be represented by a *cube* [see Fig. 4]. This type of computational unit, has been examined for non-redundant as well as redundant continued fraction representations in [Kornerup and Matula 85, Kornerup and Matula 88, Kornerup and Matula 89, Kornerup and Matula 90], based on an algorithm developed by Gosper in [Gosper 72].

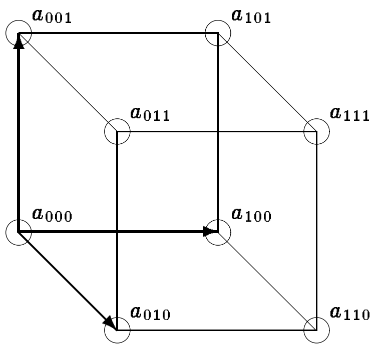


Figure 4: Cube modelling a function of two variables.

If we evaluate $f(x_2, x_1)$ on the endpoints of an interval defined by $x_1 \in [\frac{p_{1,1}}{q_{1,1}}, \frac{p_{1,0}}{q_{1,0}}]$ (respectively $x_2 \in [\frac{p_{2,1}}{q_{2,1}}, \frac{p_{2,0}}{q_{2,0}}]$) we get:

$$f(x_2, x_1 = \frac{p_{1,b_1}}{q_{1,b_1}}) = \frac{(a_{111}p_{1,b_1} + a_{101}q_{1,b_1})x_2 + (a_{011}p_{1,b_1} + a_{001}q_{1,b_1})}{(a_{110}p_{1,b_1} + a_{100}q_{1,b_1})x_2 + (a_{010}p_{1,b_1} + a_{000}q_{1,b_1})}\ ,\ b_1 \in \mathbb{B} \tag{6}$$

respectively

$$f(x_2 = \frac{p_{2,b_2}}{q_{2,b_2}}, x_1) = \frac{(a_{111}p_{2,b_2} + a_{011}q_{2,b_2})x_1 + (a_{101}p_{2,b_2} + a_{001}q_{2,b_2})}{(a_{110}p_{2,b_2} + a_{010}q_{2,b_2})x_1 + (a_{100}p_{2,b_2} + a_{000}q_{2,b_2})}\ ,\ b_2 \in \mathbb{B}. \tag{7}$$

The updating taking place in equations (6) (respectively (7)) can be described by the following matrix multiplications:

$$F_{1xx}X_1 = \begin{bmatrix} a_{111} & a_{101} \\ a_{110} & a_{100} \end{bmatrix} \begin{bmatrix} p_{1,1} & p_{1,0} \\ q_{1,1} & q_{1,0} \end{bmatrix} = \begin{bmatrix} a'_{111} & a'_{101} \\ a'_{110} & a'_{100} \end{bmatrix} = F'_{1xx}. \tag{8}$$

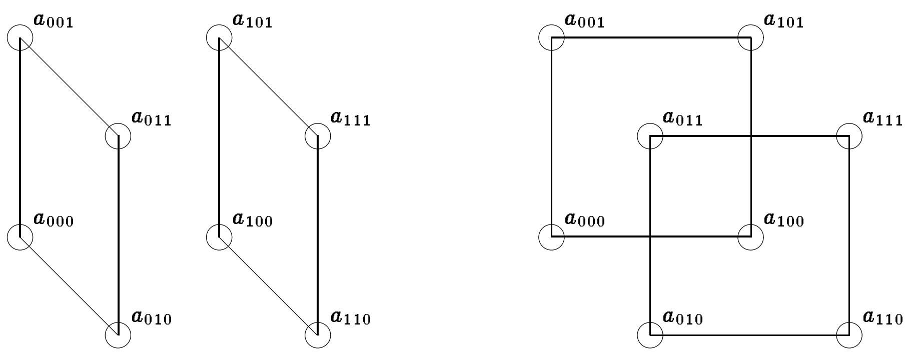
$$F_{0xx}X_1 = \begin{bmatrix} a_{011} & a_{001} \\ a_{010} & a_{000} \end{bmatrix} \begin{bmatrix} p_{1,1} & p_{1,0} \\ q_{1,1} & q_{1,0} \end{bmatrix} = \begin{bmatrix} a'_{011} & a'_{001} \\ a'_{010} & a'_{000} \end{bmatrix} = F'_{0xx} \tag{9}$$

respectively:

$$F_{x1x}X_2 = \begin{bmatrix} a_{111} & a_{011} \\ a_{110} & a_{010} \end{bmatrix} \begin{bmatrix} p_{2,1} & p_{2,0} \\ q_{2,1} & q_{2,0} \end{bmatrix} = \begin{bmatrix} a'_{111} & a'_{011} \\ a'_{110} & a'_{010} \end{bmatrix} = F'_{x1x} \tag{10}$$

$$F_{x0x}X_2 = \begin{bmatrix} a_{101} & a_{001} \\ a_{100} & a_{000} \end{bmatrix} \begin{bmatrix} p_{2,1} & p_{2,0} \\ q_{2,1} & q_{2,0} \end{bmatrix} = \begin{bmatrix} a'_{101} & a'_{001} \\ a'_{100} & a'_{000} \end{bmatrix} = F'_{x0x} \tag{11}$$

The matrices F_{1xx} and F_{0xx} (respectively F_{x1x} and F_{x0x}) can be found as faces of the cube [see Fig. 5]. With this computation model it is now straight forward to expand the interval matrices X_1 (respectively X_2) in terms of the homographic matrix coding developed in [Section 3]. For instance we can expand X_1 as $EX_{1,1}X_{2,1}\cdots X_{n,1}$, with $X_{i,1} \in \Sigma_M$. Thus the process of consuming input from an operand, corresponds to performing two matrix multiplications in parallel.



(a) (b)

Figure 5: Input transformation. (a) Input from variable x_1. By decomposing the cube we get 2 matrices F_{1xx} and F_{0xx}. (b) Input from variable x_2. Decomposition yields the matrices F_{x1x} and F_{x0x}

Observation 12. If the function $f(x_2, x_1)$, modeled by the cube F, is evaluated on the domain: $]\frac{p_{1,1}}{q_{1,1}}, \frac{p_{1,0}}{q_{1,0}}[\times]\frac{p_{2,1}}{q_{2,1}}, \frac{p_{2,0}}{q_{2,0}}[$, then the value of the function evaluated on a corner of the domain, can be found from F as:

$$f(x_2 = \frac{p_{2,b_2}}{q_{2,b_2}}, x_1 = \frac{p_{1,b_1}}{q_{1,b_1}}) = \frac{a_{b_2 b_1 1}}{a_{b_2 b_1 0}}\ ,\ b_1, b_2 \in \mathbb{B}$$

In the process of transforming the cube during input, we perform two matrix multiplications, and we are thus faced with a notational problem when we want to state a matrix equivalent output equation. To overcome this problem we will use the symbols $\otimes_1$ (respectively $\otimes_2$) to denote the infix operator that performs two matrix multiplications: $F_{1xx}X_1$ and $F_{0xx}X_1$ (respectively $F_{x1x}X_2$ and $F_{x0x}X_2$), when consuming input from x_1 (respectively x_2). We will use a braced notation to denote input transformation from both variables according to some order. In order to compare the cube, evaluated on a domain, against an interval, we introduce a function $\Phi_C : cube \to \mathbb{I}$ that maps a cube into an interval, as defined by:

$$\Phi_C \left(F \left\{ \begin{matrix} \otimes_1 EX_1X_2 \cdots X_i \\ \otimes_2 EY_1Y_2 \cdots Y_j \end{matrix} \right\} \right) = f(\varphi(x_1x_2 \cdots x_i), \varphi(y_1y_2 \cdots y_j)). \tag{12}$$

Thus the condition $f(\varphi(x_1x_2 \cdots x_i), \varphi(y_1y_2 \cdots y_j)) \subseteq \varphi(z_1z_2 \cdots z_p)$ can be stated as:

$$\Phi_C \left(F \left\{ \begin{matrix} \otimes_1 EX_1X_2 \cdots X_i \\ \otimes_2 EY_1Y_2 \cdots Y_j \end{matrix} \right\} \right) \subseteq \Phi(EZ_1Z_2 \cdots Z_P). \tag{13}$$

In [Section 3], output generation for the one-dimensional case was established as multiplying with the inverse of a matrix from Σ_M. This process was equivalent to function composition, as it will be in the case of two variables. Let F be a cube modeling the function $f : \mathbb{R}^2 \to \mathbb{R}$, and let $G = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$ be a matrix that represents a function $g : \mathbb{R} \to \mathbb{R}$. The composition $g \circ f : \mathbb{R}^2 \to \mathbb{R}$ is:

$$\begin{aligned}
&(g \circ f)(x_2, x_1) \\
&= \frac{\alpha f(x_2, x_1) + \beta}{\gamma f(x_2, x_1) + \delta} \\
&= \frac{(\alpha a_{111} + \beta a_{110})x_2x_1 + (\alpha a_{101} + \beta a_{100})x_2 + (\alpha a_{011} + \beta a_{010})x_1 + (\alpha a_{001} + \beta a_{000})}{(\gamma a_{111} + \delta a_{110})x_2x_1 + (\gamma a_{101} + \delta a_{100})x_2 + (\gamma a_{011} + \delta a_{010})x_1 + (\gamma a_{001} + \delta a_{000})} \\
&= \frac{a'_{111}x_2x_1 + a'_{101}x_2 + a'_{011}x_1 + a'_{001}}{a'_{110}x_2x_1 + a'_{100}x_2 + a'_{010}x_1 + a'_{000}}.
\end{aligned}$$

The transformation is clearly equivalent to the matrix multiplications GF_{1xx} and GF_{0xx} or GF_{x1x} and GF_{x0x}, thus in the case of output we have a choice between two possible ways of expressing the output transformation.

Assuming as in [Section 3], that X_l^{-1}, Y_m^{-1} and E^{-1} represents continuous monotone functions, we may transform (13) into:

$$\Phi_C \left(Z_{p-1}^{-1} \cdots Z_2^{-1} Z_1^{-1} E^{-1} \otimes_1 F \left\{ \begin{matrix} \otimes_1 EX_1X_2 \cdots X_i \\ \otimes_2 EY_1Y_2 \cdots Y_j \end{matrix} \right\} \right) \subseteq \Phi(Z_P). \tag{14}$$

Similarly by replacing Z_p with T, and inclusion by equality in (14) we get the cube equivalent equation of the termination test: $f(\varphi(x_1x_2 \cdots x_i), \varphi(y_1y_2 \cdots y_j) = \varphi(z_1z_2 \cdots z_{p-1}\tau)$.

Thus an algorithm for computing $f(x, y)$ at some point $(\varphi(x_1x_2 \cdots x_n\tau)$, $\varphi(y_1y_2 \cdots y_m\tau))$, could proceed as follows: Let F be the cube representation of the function f. Transform the function matrix as: $F \leftarrow E^{-1} \otimes_1 F$, $F \leftarrow F \otimes_1 E$ and finally $F \leftarrow F \otimes_2 E$. Consume input from x by setting: $F \leftarrow F \otimes_1 X_i$, similarly from y by $F \leftarrow F \otimes_2 Y_j$. When possible produce output by $F \leftarrow Z_p^{-1} \otimes_1 F$.

As described in [Section 3], an algorithm based on a finite-state transducer can be devised, with the difference that we will have to perform cube transformations rather that simple matrix multiplications, and that we now have two input variables instead of one, each having an associated state variable. In the case of *input consumption* from variable i, the table is to be interpreted as: Upon receiving a symbol s, perform the cube transformation

$$F \otimes_i \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$$

with the matrix corresponding to the symbol s in the present state, when the transformation has been completed move to the state designated by *next-state*. When performing *output*, we first need to check if output can be generated. This is done by checking if for any of the symbols in the present state we have:

$$\left[\frac{\alpha}{\gamma}, \frac{\beta}{\delta}\right] \supseteq \Phi_C(F)$$

if this holds for some symbol then the symbol is send to the output, and the cube transformation

$$\begin{bmatrix} \delta & -\beta \\ -\gamma & \alpha \end{bmatrix} \otimes_1 F$$

is performed. The range of the function on some domain (specified by $\Phi_C(F)$), can be found by examination of the function evaluated on the boundary of the domain if the function is monotone with respect to each variable on this domain. The domain has 4 edges that can be treated independently. The function values at the endpoints of these 4 intervals can be found in the cube, and the interval type can be resolved by knowing the interval type of the operand that is not constant along the edge under consideration, and by examining whether or not a sign change in the denominator occurs along the edge, this is again computable from the cube.

5 Digit Serial Floating Point Operands

If the operand of some function has a fixed point format, the set of representable number will be very limited, if a larger set is needed we can tag information about the position of the radix point onto the operands. In traditional on-line algorithms this is done by coupling exponent information to the first digit of the mantissa part, where the mantissa part is to be kept quasi-normalized, in the sense that the two first digits should have the same sign, and the first digit can not take on the value zero [Watanuki and Ercegovac 81, Watanuki and Ercegovac 83, Owens 83]. A problem with this type of normalization is that in certain computations cancellation might occur (e.g. if two almost equal numbers are subtracted), requiring extensive normalisation. If this computation is feeding another computation unit, the other units delivering input to the same computation unit will have to delay their output, in order to keep the input to the unit synchronized [see Fig. 6]. Thus the set of representable numbers have been enlarged, at the expense of a variable output delay, due to possible normalization requirements.

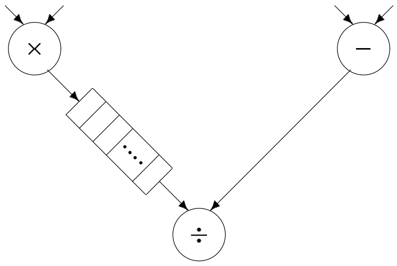


Figure 6: Computation requiring buffering.

We will now show how a floating point can be introduced in the redundant binary fixed point number representation considered in [Section 3] and [Section 4]. We will require normalization of the form:

$$b_0 b_1 \geq 0 \wedge b_0 \neq 0, \tag{15}$$

such that the mantissa part can represent any number in the intervals defined by:

$$\varphi(1) = [1.0\bar{1}\bar{1}\bar{1}\ldots, 1.1111\ldots] = [\frac{1}{2}, 2] \tag{16}$$
$$\varphi(\bar{1}) = [\bar{1}.\bar{1}\bar{1}\bar{1}\ldots, \bar{1}.0111\ldots] = [-2, -\frac{1}{2}]$$

We introduce two new symbols l and u in order to specify negative, respectively positive valued exponents. If the mantissa part is preceded by n symbols of the u type respectively the l type, then the value of the whole string is to be interpreted as 2^n respectively 2^{-n} times the mantissa part. The strings are given the following interpretation:

$$\varphi(u) = [1, -1] \;\; \varphi(l) = [-1, 1] \tag{17}$$
$$\varphi(u^n) = [2^{n-1}, -2^{n-1}] \;\; \varphi(l^n) = [-2^{-n+1}, 2^{-n+1}] \;, \; n \geq 1 \tag{18}$$

$$\varphi(u^n b_0 \cdots b_{-k}) = [(-2^{-k} + \sum_{i=0}^{k} b_{-i} 2^{-i}) 2^n, (2^{-k} + \sum_{i=0}^{k} b_{-i} 2^{-i}) 2^n], k > 0 \tag{19}$$

$$\varphi(l^n b_0 \cdots b_{-k}) = [(-2^{-k} + \sum_{i=0}^{k} b_{-i} 2^{-i}) 2^{-n}, (2^{-k} + \sum_{i=0}^{k} b_{-i} 2^{-i}) 2^{-n}], k > 0. \tag{20}$$

In the case $k = 0$ in (19) respectively (20) we get one of the intervals (17) multiplied by 2^n respectively 2^{-n}. We take $\varphi(l^+\tau) = [0, 0]$ to represent zero, and $\varphi(u^+\tau) = [+\infty, -\infty]$ to represent infinity ∞. With this coding it is possible to represent any number in the interval $]-\infty, +\infty[$ as well as ∞. As seen from [Fig. 7] the first symbol of a digit string corresponds to a wide interval, with the four intervals collectively covering $\mathbb{R} \cup \{\infty\}$. Furthermore the intervals are overlapping

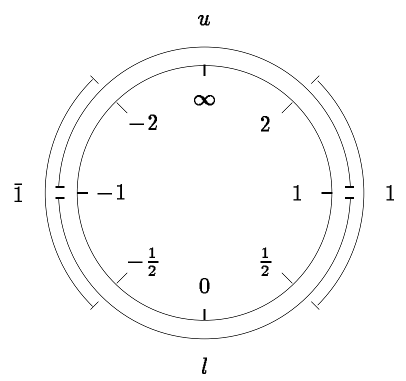


Figure 7: Intervals corresponding to the possible first symbols of a string.

indicating that there is redundancy in the exponent notation, which in turn will speed up the processing of output digits. It is easily seen that this representation satisfies [Definition 1] of a Digit Serial Number Representation. The state table [Tab. 2] describes a transducer that translates symbols into matrices, that when multiplied together will result in an interval matrix representing the interval corresponding to the input string. Notice that the automaton automatically enforces normalization of the mantissa part. The interpretation of the state table is equivalent to that of the state machine in [Section 3]. Notice that in any state there are at most four possible symbols, making a two bit encoding of the symbols feasible.

A floating point unit operating on such a digit serial representation, will accept asynchronous operands, thus buffering due to normalization problems is no longer an issue. Moreover since the exponent bits have been given an interval representation much like the mantissa bits, the exponent bits can be consumed and generated in a digit serial manner, making the unit fully data driven. Another advantage is that the length of the exponent is unlimited, enabling representation of arbitrarily large or small numbers. However, since the encoding of the exponent is unary the representation is most useful if operands are scaled into a range with small absolute values of exponents.

6 Conclusion

Traditional on-line algorithms are based on a recursive formulation of a residual. The residual can to some extent be thought of as a scaled approximation of the midpoint of the interval corresponding to the variation of the function evaluated on some domain, i.e. $f(\varphi(x_1x_2\cdots x_{p+\delta}), \varphi(y_1y_2\cdots y_{p+\delta}))$. Fixing the on-line delay δ to some constant, gives an upper bound on the approximation error and hence the width of the interval. If the on-line delay is chosen sufficiently large, the output digits can be found by a round to nearest operation

State	Symbol	Next-state	$(\alpha,\beta,\gamma,\delta)$	Interval Type
Start	u	Upper	$(1,-1,1,1)$	I_∞
	l	Lower	$(-1,1,1,1)$	I_{normal}
	$\bar{1}$	Negative	$(-2,-1,1,2)$	I_{normal}
	1	Positive	$(1,2,2,1)$	I_{normal}
Upper	u	Upper	$(3,-1,-1,3)$	I_∞
	$\bar{1}$	Negative	$(-3,0,5,4)$	I_{normal}
	1	Positive	$(4,5,0,-3)$	I_{normal}
	τ	Terminated	$(1,-1,-1,1)$	$infinity$
Lower	l	Lower	$(3,1,1,3)$	I_{normal}
	$\bar{1}$	Negative	$(4,5,0,3)$	I_{normal}
	1	Positive	$(3,0,5,4)$	I_{normal}
	τ	Terminated	$(1,1,1,1)$	$zero$
Positive	0	Mantissa	$(3,1,0,4)$	I_{normal}
	1	Mantissa	$(1,0,1,3)$	I_{normal}
	τ	Terminated	$(1,1,1,1)$	$point$
Negative	$\bar{1}$	Mantissa	$(3,1,0,1)$	I_{normal}
	0	Mantissa	$(4,0,1,3)$	I_{normal}
	τ	Terminated	$(1,1,1,1)$	$point$
Mantissa	$\bar{1}$	Mantissa	$(4,2,0,2)$	I_{normal}
	0	Mantissa	$(3,1,1,3)$	I_{normal}
	1	Mantissa	$(2,0,2,4)$	I_{normal}
	τ	Terminated	$(1,1,1,1)$	$point$
Terminated	–	Terminated	$(1,0,0,1)$	$previous$

Table 2: Transition/output table for finite state transducer, translating abstract symbols into intervals.

performed on the residual. Thus an on-line algorithm implicitly keeps a center-radius representation of an interval that includes the actual variation interval of the function.

The Cube F can be thought of as a residual as well, but in contrast to ordinary on-line algorithms the interval $f(\varphi(x_1x_2\cdots x_{p+\delta}),\varphi(y_1y_2\cdots y_{p+\delta}))$ is represented by its actual endpoints, thus all information needed when determining output digits is represented as explicit variables in the model. This in turn gives a more powerful algorithm, since we can perform actual interval inclusion tests. An on-line algorithm with *optimal* on-line delay can easily be devised, where the digit selection function could be implemented as a test for interval inclusion, or since the actual midpoint of the function variation interval can be computed, a round to nearest could be performed as in traditional on-line algorithms. The optimal on-line delay for certain functions has been examined in [Duprat, Herreros and Muller 89]. A less obvious consequence of the explicit representation, is that we no longer need to keep input and output synchronous, thus by imposing synchronization restrictions on the cube algorithm for radix number systems , we can realize algorithms in the three different synchronization classes as depicted in figure [Fig. 8].

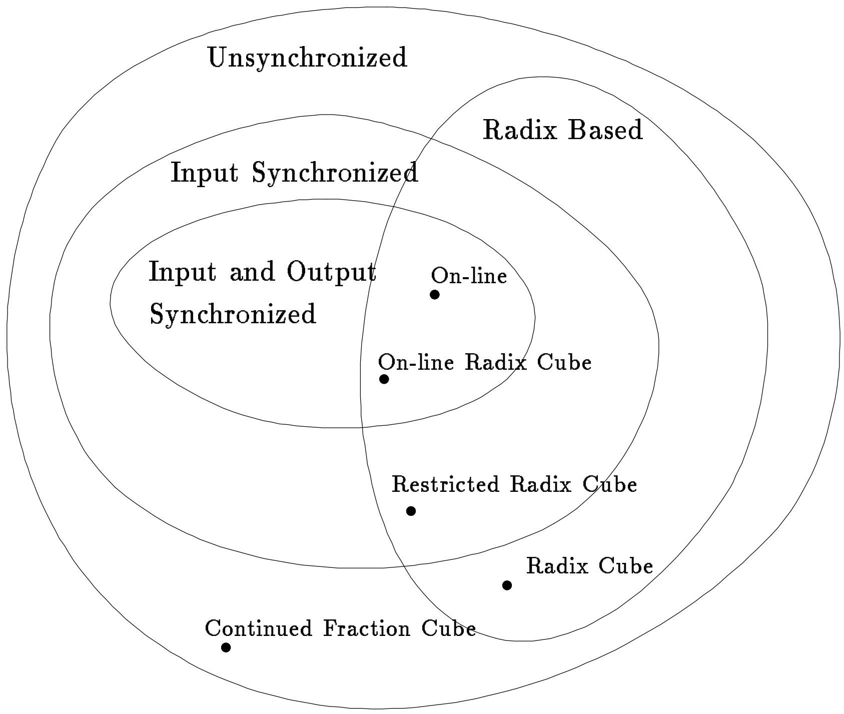


Figure 8: Topological classification of MSB-first digit serial algorithms.

A cube algorithm with no synchronization for continued fractions has been examined in [Gosper 72, Kornerup and Matula 88, Kornerup and Matula 89, Kornerup and Matula 90], it is an open problem whether synchronized algorithms exists for this number system. The continued fractions prove to be an elegant way of representing real valued operands with arbitrary precision, one problem with this number representation is, however, that the individual terms of a continued fractions can be any natural number, or integer number if negative numbers are allowed. This problem can be solved by expanding each term into simpler terms as demonstrated in [Kornerup and Matula 88, Kornerup and Matula 89, Kornerup and Matula 90]. One advantage of this representation is that the range of representable numbers is inherently unbounded. The floating point, digit serial representations introduced in the previous section provides the same unbounded range for radix representations. Digit serial computation on such operands can be performed using homographic or bi-homographic functions as described, however the state information in the matrix - and cube-representations is in simple cases (like addition) inherently large. It will be interesting to see if simpler, digit serial algorithms can be derived in special cases.

A combined way of handling infinite and finite precision arithmetic, can be constructed from the examples shown in this paper. By introducing an extra termination symbol, which signals that an operand was merely terminated due to its length exceeding some bound, operands can be kept as intervals, representing an imprecise operand. Operands terminated in the ordinary way can be taken to represent exact numbers.

References

[Vuillemin 90] Vuillemin, Jean: "Exact Real Computer Arithmetic with Continued Fractions"; IEEE Transactions on Computers, C-39, 8 (1990), 1087–1105.

[Duprat, Herreros and Muller 89] Duprat, J., Herreros, Y., Muller, J.M.: "Some Results about On-Line Computation of Functions"; Proc. 9th IEEE Symposium on Computer Arithmetic, IEEE Computer Society Press, Santa Monica (1989), 112-118.

[Kornerup and Matula 85] Kornerup, P., Matula, D. W.: "Finite Precision Lexicographic Continued Fraction Number Systems"; Proc. 7th IEEE Symposium on Computer Arithmetic, IEEE Computer Society Press, Urbana (1985), 207-214.

[Kornerup and Matula 88] Kornerup, P., Matula, D. W.: "An On-line Arithmetic Unit for Bit-Pipelined Rational Arithmetic"; Journal of Parallel and Distributed Computing, 5, 3 (1988), 310–330.

[Kornerup and Matula 89] Kornerup, P., Matula, D. W.: "Exploiting Redundancy in Bit-Pipelined Rational Arithmetic"; Proc. 9th IEEE Symposium on Computer Arithmetic, IEEE Computer Society Press, Santa Monica (1989), 119-126.

[Kornerup and Matula 90] Kornerup, P., Matula, D. W.: "An Algorithm for Redundant Binary Bit-Pipelined Rational Arithmetic"; IEEE Transactions on Computers, C-39, 8 (1990), 1106–1115.

[Gosper 72] Gosper, R. W.: "Item 101 in Hakmem"; AIM239, MIT, USA (1972), 37-44.

[Watanuki and Ercegovac 81] Watanuki, O., Ercegovac, M. D.: "Floating-Point On-Line Arithmetic: Algorithms"; Proc. 5th IEEE Symposium on Computer Arithmetic, IEEE Computer Society Press, Ann Arbor (1981) 81-86.

[Watanuki and Ercegovac 83] Watanuki, O., Ercegovac, M. D.: "Error Analysis of Certain Floating-Point Algorithms"; IEEE Transactions on Computers, C-32, 4 (1983), 352–358.

[Owens 83] Owens, R. M.: "Techniques to Reduce the Inherent Limitations of Fully Digit On-Line Arithmetic"; IEEE Transactions on Computers, C-32, 4 (1983), 406–411.