On-the-Fly Algorithms and Sequential Machines

Christiane Frougny
L.I.A.F.A (L.I.T.P.)
4 place Jussieu, 75252 Paris Cedex 05, France.
Christiane.Frougny@litp.ibp.fr

Abstract

It is shown that a function is computable by an on-the-fly algorithm processing data in the most significant digit first fashion with a finite number of registers if and only if it is computable by a right subsequential finite state machine processing data in the less significant digit first fashion. Applications to conversion of redundant into conventional representations and to the canonical Booth recoding are given. We also indicate some applications to negative or complex radix number systems.

1. Introduction

Conversion from a redundant into a conventional representation is important in Computer Arithmetic because on-line algorithms require redundant representations with signed digits [19]. Similarly, in some arithmetic algorithms such as the SRT division algorithm, the result is generated most significant digit first (MSDF) in a redundant format. It is well known that such conversions cannot be realized on-line, that is to say MSDF digitwise, one output digit being produced by one input digit after a certain delay, because there is a carry which propagates from right to left. On-the-fly algorithms to solve this problem have been proposed by Ercegovac and Lang [7] and generalized by Kornerup [13] (see also [15] for application to multiplication). In an on-the-fly algorithm, data are processed in a serial manner from most significant to least significant, but the algorithm uses several registers, each of them representing a correct prefix of the result, corresponding to an assumed value of the carry. Using parallel prefix computation, on-the-fly algorithms can be implemented in time $\mathcal{O}(\log n)$ (see [13]).

The purpose of this work is not to improve on existing algorithms, but rather to present a theoretical framework allowing to easily obtain on-the-fly algorithms whenever it is possible. We show that a function is computable by an on-the-fly algorithm if and only if it is computable by a right subsequential finite state machine. Such a machine is a 2-tape finite state automaton of a certain kind: inputs are deterministically and serially processed from right to left, *i.e.* LSDF digitwise, and the output is generated LSDF (see [6] and [3]). In radix r with non redundant digit set $\{0, \cdots, r-1\}$, addition, subtraction, multiplication by a fixed integer are right subsequential functions. Division by a fixed integer is a left subsequential function (data are processed MSDF). More generally, functions which are computable by a 2-tape finite state automaton are those which need only a finite auxiliary storage memory, independently of the size of the data. Note that squaring, multiplication and division are functions which cannot be computed by any kind of a 2-tape finite state automaton but they are on-line computable.

On the other hand, with a redundant digit set of the form $\{-a, \cdots, a\}$ with $r/2 \leq a \leq r-1$, and following the Avizienis algorithm [2], addition is computable by an on-line finite state automaton, which is a particular case of a left subsequential machine (see [16], [9]).

The paper is organized as follows. First we set some definitions of computability. We prove that a function is on-the-fly computable if and only if it is right subsequential. Every radix r conversion into conventional representations is right subsequential. We illustrate our method on radix 2 conversion of redundant into conventional representations, showing how the on-the-fly algorithm of [7] can be derived from the right subsequential machine. We give applications to the conversion from radix 4 redundant into conventional 2's complement, to the Booth canonical recoding and to conversion in number systems where the base is a negative integer, or some complex number.

2. Definitions and results

In the sequel, we consider integers as finite strings or *words* on a finite digit set X, called *alphabet*, and real numbers as infinite strings on this digit set. Let r be the radix, r integer ≥ 2. An integer p is represented by a word $p_1 \cdots p_m$ of the free monoid X^* generated by X, with for $1 \leq k \leq m$, $p_k \in X$, and such that $p = \sum_{k=1}^{k=m} p_k r^{m-k}$. A real number

x of $[0,1]$ is represented by an infinite word $(x_k)_{k\geq 1}$ such that $x_k \in X$ and $x = \sum_{k\geq 1} x_k r^{-k}$. The leftmost digit is the most significant one. The empty word is denoted by ε. We expose the results on integers (finite words), but they are valid for real numbers (infinite words) as well.

Let X and Y be two finite digit sets, and let μ be a function from X^* to Y^* (for simplicity we consider only one-variable functions, but it is not a restriction). X is the *input* alphabet, and Y is the *output* alphabet. Following [19], we say that μ is *on-line computable* (with delay δ) if there exists a positive constant δ such that, for $y_1 \cdots y_n = \mu(x_1 \cdots x_m)$, to generate y_k, for $1 \leq k \leq n$, it is necessary and sufficient to have $x_1, \ldots, x_{k+\delta}$ available. After the delay, one digit of the result is produced upon receiving one digit of X. For us, *on-line* always refers to MSDF serial computations of that kind. It is well known that some functions are not on-line computable, like addition in the binary system with non redundant digit set $\{0,1\}$. Another example is the conversion function χ between radix 2 redundant representations and 2-complement representations

$$\chi : \{\bar{1}, 0, 1\}^* \longrightarrow \{0,1\}^*.$$

The conversion is equivalent to a subtraction. This conversion is not on-line computable: consider two redundant representations $p = 1\overbrace{0\cdots 0}^{n}1$, denoted by $p = 10^n 1$ and $q = 10^n\bar{1}$, for $n \geq 1$. Then $\chi(p) = 010^n 1$ and $\chi(q) = 001^{n+1}$; this shows that the least significant digit as to be known to be able to output the most significant digits of the result.

We now introduce an other definition [19]. A function is said to be *on-the-fly computable* if the digits of the result are obtained in a serial fashion in MSDF mode, using a finite number of registers corresponding to different conditional forms of the current result. More precisely, let $\mu : X^* \longrightarrow Y^*$ and let N registers $S_0, \cdots, S_{N-1} \subset Y^*$. Denote by $S_i[k]$ the content of register S_i at step k after having read MSDF k digits. We say that μ is on-the-fly computable with N registers if for $p = p_1 \cdots p_m$, $p_k \in X$ for $1 \leq k \leq m$, $S_0[k] = \mu(p_1 \cdots p_k)$, and if for $0 \leq i \leq N-1$, $S_i[k]$ is the result of a computation with input $p_1 \cdots p_k$ "assuming a certain condition s_i on the beginning of the computation". Conditions s_i are supposed to be all different for every $0 \leq i \leq N-1$. For an example where a condition is not always a carry, see the Booth canonical recoding below section 3.3.

Then we need a definition from Automata Theory (see [6], [3]). A *right subsequential machine* with input alphabet X and output alphabet Y, $\mathcal{M} = (S, X \times Y^*, E, s_0, \omega)$, is a directed graph labelled by elements of $X \times Y^*$:

- S is the set of vertices, called *states*, which is finite
- $E \subset S \times (X \times Y^*) \times S$ is the set of labelled edges
- s_0 is the *initial state*
- ω is the *terminal function* from S to Y^*. When $\omega(s) = \varepsilon$ for any $s \in S$, the machine is said to be *right sequential.*

The machine must satisfy the following property: it is input deterministic, that is to say if $s \xrightarrow{x/u} t$ and $s \xrightarrow{x/u'} t'$ are two edges of $\mathcal{M}$, then necessarily $t = t'$ and $u = u'$.
A word $p = p_1 \cdots p_m$, with $p_k \in X$ for $1 \leq k \leq m$, has $q \in Y^*$ for image by $\mathcal{M}$ if there exists a path in $\mathcal{M}$ starting in the initial state s_0

$$s_0 \xrightarrow{p_m/u_m} s_1 \xrightarrow{p_{m-1}/u_{m-1}} \cdots s_{m-1} \xrightarrow{p_1/u_1} s_m$$

with $u_j \in Y^*$, and $q = \omega(s_m)u_1 \cdots u_m$. A function $\mu : X^* \longrightarrow Y^*$ is *right subsequential* if there exists a right subsequential machine $\mathcal{M}$ such that if $p \in X^*$ and $q \in Y^*$, $q = \mu(p)$ if and only if q is the image of p by $\mathcal{M}$. For an example see section 3.1.

The machine is called *right* subsequential to stress on the fact that data are processed from *right* to left (LSDF). The dual notion, where data are processed from *left* to right (MSDF) and where the terminal function comes as a suffix of the result, is called a *left* subsequential machine.

We can now state the principal result. The proof is illustrated on an example in section 3.1.

Proposition 1 . *A function $\mu : X^* \longrightarrow Y^*$ with Domain(μ) $= X^*$ is on-the-fly computable if and only if it is a right subsequential function.*

Proof. First suppose that $\mathcal{M} = (S, X \times Y^*, E, s_0, \omega)$ is a right subsequential machine realizing μ. Since the function is total, the machine can be chosen *complete*, that is to say, for each state $s \in S$ and for each input digit $x \in X$, there exists an edge of the form $s \xrightarrow{x/u} t$. In addition we choose a machine which is *minimal* in the number of states. Then we derive from $\mathcal{M}$ an on-the-fly algorithm computing MSDF the function μ as follows. Let us denote the states of $\mathcal{M}$ by $S = \{s_0, \cdots, s_{N-1}\}$. We thus need N registers denoted by $S_0, \cdots, S_{N-1}$, register S_i corresponding to state s_i, $0 \leq i \leq N-1$. Initialization is the following: for $0 \leq i \leq N-1$, $S_i[0] = \omega(s_i)$. Let $p = p_1 \cdots p_m$ be an input word and let $0 \leq k \leq m-1$; recurrence relations are determined by: for each $0 \leq i \leq N-1$, for each $p_{k+1} \in X$, if in $\mathcal{M}$ there is an edge of the form $s_i \xrightarrow{p_{k+1}/u} s_j$, $0 \leq j \leq N-1$, put $S_i[k+1] = (S_j[k], u)$ where $(S_j[k], u)$ denotes the result of the concatenation of $S_j[k]$ and of u (note that, for a given value of p_{k+1}, there is only one possible edge because $\mathcal{M}$ is input deterministic). We claim that for $0 \leq i \leq N-1$ and for $0 \leq k \leq m$, $S_i[k]$ is equal to the output label of the unique path in $\mathcal{M}$ starting in s_i and with

input label $p_k \cdots p_1$. The proof is by induction on k. When $k = 0$, the input is the empty word, and $S_i[0] = \omega(s_i)$. Let us consider the following path in $\mathcal{M}$

$$s_i \xrightarrow{p_k/u_k} s_{i_1} \xrightarrow{p_{k-1}/u_{k-1}} \cdots s_{i_{k-1}} \xrightarrow{p_1/u_1} s_{i_k} \xrightarrow{/\omega(s_{i_k})} .$$

By induction hypothesis, $S_{i_1}[k-1] = \omega(s_{i_k})\ u_1 \cdots u_{k-1}$. By construction, $S_i[k] = (S_{i_1}[k-1], u_k)$, thus $S_i[k] = \omega(s_{i_k})u_1 \cdots u_{k-1}u_k$, and we are done. Hence, for each $1 \leq k \leq m$, $S_0[k] = \mu(p_1 \cdots p_k)$.

Conversely, let us suppose that $\mu : X^* \longrightarrow Y^*$ is on-the-fly computable with N registers $S_0, \cdots, S_{N-1} \subset Y^*$. We define a right subsequential machine as follows: let $\mathcal{M} = (S, X \times Y^*, E, s_0, \omega)$ where $S = \{s_0, \cdots, s_{n-1}\}$, each s_i corresponding to S_i. When the recurrence relations for $p_{k+1} \in X$ are of the form $S_i[k+1] = (S_j[k], u)$, for i and j in $\{0, \cdots, N-1\}$ and $u \in Y^*$, we define in E an edge $s_i \xrightarrow{p_{k+1}/u} s_j$. The terminal function is defined by $\omega(s_i) = S_i[0]$. Clearly $\mathcal{M}$ is input deterministic, and as above, one verifies that $S_0[k] = \mu(p_1 \cdots p_k)$ if and only if there is a path in $\mathcal{M}$

$$s_0 \xrightarrow{p_k/u_k} s_{i_1} \xrightarrow{p_{k-1}/u_{k-1}} \cdots s_{i_{k-1}} \xrightarrow{p_1/u_1} s_{i_k} \xrightarrow{/\omega(s_{i_k})}$$

with $\mu(p_1 \cdots p_k) = \omega(s_{i_k})u_1 \cdots u_k$. ∎

Since the number of registers in the on-the-fly algorithm is equal to the number of states of the right subsequential machine, it is important to find a minimal right subsequential machine, and it is known how to achieve this task [5].

Now we show a general result on conversion in radix r onto the canonical digit set $\{0, \cdots, r-1\}$.

Proposition 2 . *Let r be an integer ≥ 2, let X be any finite set of digits, and let $Y = \{0, \cdots, r-1\}$. The conversion $\psi : X^* \longrightarrow Y^*$ between representations with digits in X onto r's complement representations is a right subsequential function, and is thus on-the-fly computable.*

Proof. Without loss of generality, one can suppose that there exists an integer $d \geq 1$ such that $X = \{-d, \cdots, d\}$. One defines a right subsequential machine $\mathcal{R} = (S, X, Y, E, \omega)$ as follows. The set of states is $S = \{-d-1, -d, \cdots, d\}$. Let s be a state, and let $x \in X$ be an input digit.
If $s + x \geq 0$, let $s + x = rt + y$, with t being the quotient and y being the remainder of the Euclidean division of $s+x$ by r. Clearly $y \in Y$. We have $t = (s + x - y)/r$, and $t \leq d$ because $s \leq d$, $x \leq d$ and $r \geq 2$. Thus we define the edge $s \xrightarrow{x/y} t$. If $s + x < 0$, let $-(s + x) = rt + y$ be the Euclidean division as above. If $y = 0$, we put $s \xrightarrow{x/0} (-t)$. Since $s + x \geq -2d - 1$ we get that $(-t) \in S$.
If $y > 0$, we have $-y = (-r) + (r - y)$, thus $s + x = r(-t-1) + (r-y)$. Clearly $r - y \in Y$. Now, $-t - 1 = (s + x - r + y)/r \geq (-2d - r)/r \geq -d - 1$ and thus belongs to S. We let $s \xrightarrow{x/r-y} (-t-1)$.
For any state s, the terminal function $\omega(s)$ is equal to the r's complement representation of s. When $s \geq 0$, the input word represents a positive integer, if $s < 0$, then the input word represents a negative integer. It is then easy to see that ψ is realized by $\mathcal{R}$. Note that some of the states may be useless. ∎

In on-line arithmetic, the redundant digit set is usually of the form $X = \{\bar{a}, \cdots, a\}$, with $r/2 \leq a \leq r-1$. In that case, the right subsequential machine realizing the conversion onto r's complement notation has only two states, independently of the radix.

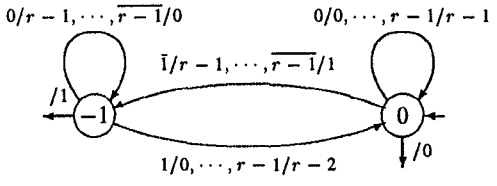

Figure 1. Right subsequential conversion from $\{\overline{r-1}, \cdots, r-1\}$ to r's complement

In Proposition 2, the canonical digit set $Y = \{0, \cdots, r-1\}$ can be replaced by a non redundant digit set $Z = \{z_0, \cdots, z_{r-1}\}$, where z_i is congruent to i modulo r, for each $0 \leq i \leq r-1$, and such that any number is representable (see [14]). For instance, when $r = 3$, we can choose $Z = \{\bar{1}, 0, 1\}$ (see [12]).

Notice that, in radix 2, addition is a conversion from $\{0, 1, 2\}$ onto $\{0, 1\}$, and the corresponding on-the-fly algorithm happens to be the well known conditional sum adding technique [18]. In that particular case, the right subsequential machine has only two states, and so only two registers are needed.

3. Examples and applications

Since there is a natural carry propagation from right to left in the most usual number systems, a lot of functions are right subsequential. We mention some of them.

3.1. Radix 2 conversion of redundant into conventional representations

Let χ be the conversion function between radix 2 redundant representations and 2-complement representations

$$\chi : \{\bar{1}, 0, 1\}^* \longrightarrow \{0, 1\}^*.$$

Below is a right subsequential machine $\mathcal{C}$ realizing χ. The input alphabet is $X = \{0, 1\}$, the output alphabet is $Y = \{\bar{1}, 0, 1\}$, the set of states is $S = \{a, b\}$, the initial state is a, the terminal function is defined by $\omega(a) = 0$ and $\omega(b) = 1$.

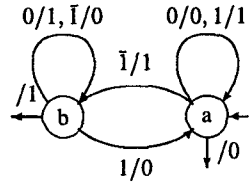

Figure 2. Right subsequential radix 2 conversion of redundant into conventional representation

State a means that there is no carry, and state b means that there is a negative carry -1. If the computation ends in state a, then the result must be prefixed by $\omega(a) = 0$ and the machine gives the conversion for a positive number. If the computation ends in state b, then we prefix the result by $\omega(b) = 1$, to get the conversion for a negative number.
Example. Let us consider 4-digit input integers. Let $v = 1\bar{1}0\bar{1}$. Then $(v)_2 = 3$. In the automaton $\mathcal{C}$, v is processed from right to left

$$\rightarrow a \xrightarrow{\bar{1}/1} b \xrightarrow{0/1} b \xrightarrow{\bar{1}/0} b \xrightarrow{1/0} a \xrightarrow{/0} .$$

Thus the conversion of v is 00011.
Let $w = \bar{1}0\bar{1}1$. Then $(w)_2 = -9$. We have in the automaton $\mathcal{C}$

$$\rightarrow a \xrightarrow{1/1} a \xrightarrow{\bar{1}/1} b \xrightarrow{0/1} b \xrightarrow{\bar{1}/0} b \xrightarrow{/1} .$$

Thus the conversion of w is 10111, which is the 2-complement representation of -9.

Following the method exposed in Proposition 1, we obtain exactly the on-the-fly algorithm of [7] for conversion. We need two registers A and B, corresponding to states a and b. The initial conditions of the recurrence are

$$A[0] = \omega(a) = 0, \quad B[0] = \omega(b) = 1.$$

We then define

p_{k+1}	$A[k+1]$	$B[k+1]$
1	$(A[k], 1)$	$(A[k], 0)$
0	$(A[k], 0)$	$(B[k], 1)$
$\bar{1}$	$(B[k], 1)$	$(B[k], 0)$

The result is contained in register A.
Example. The on-the-fly computation to convert $\bar{1}0\bar{1}1$ into 10111 is the following one.

k	p_k	A	B
0		0	1
1	$\bar{1}$	11	10
2	0	110	101
3	$\bar{1}$	1011	1010
4	1	10111	10110

Let's take the reverse automaton $\widetilde{\mathcal{C}}$, obtained in reversing the arrows of the right subsequential machine $\mathcal{C}$ (see Figure 3). In $\widetilde{\mathcal{C}}$, computations are done from left to right. It is then easy to see that, for $1 \leq k \leq m$, $A[k] \in Y^*$ contains the output label of the unique path with input $p_1 \cdots p_k$ arriving in state a in $\widetilde{\mathcal{C}}$ and similarly $B[k]$ for state b.

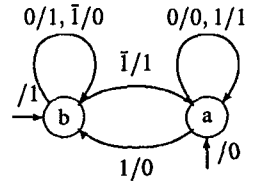

Figure 3. The reverse automaton $\widetilde{\mathcal{C}}$

3.2. Conversion from radix 4 redundant into conventional 2's complement

The conversion δ from radix 4 redundant to conventional 2's complement is shown to be on-the-fly computable in [7]. Let $X = \{\bar{3}, \cdots, 3\}$ and $Y = \{0, 1\}$. Then $\delta : X^* \longrightarrow Y^*$ is realized by the minimal right subsequential machine $\mathcal{D}$ as follows.

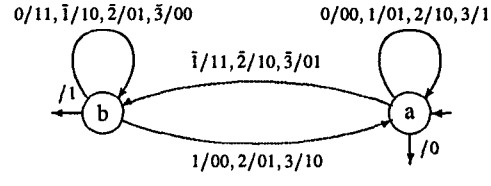

Figure 4. Right subsequential machine $\mathcal{D}$

Applying our method we find exactly the on-the-fly algorithm given in [7].

3.3. Booth canonical recoding

The Booth canonical recoding consists in, given a binary representation, finding an equivalent one with signed bits, and having the minimum number of non zero digits [4]. This has important application to multiplication. The Booth canonical recoding can be obtained by the simple LSDF

algorithm: each block of the form 01^n, with $n \geq 2$, is transformed into $10^{n-1}\bar{1}$, and other blocks are left unchanged. Let $X = \{0, 1\}$ be the input alphabet and let $Y = \{\bar{1}, 0, 1\}$ be the output alphabet. The Booth canonical recoding is a right subsequential function $\beta : X^* \longrightarrow Y^*$ realized by the following machine $\mathcal{B}$, which is minimal.

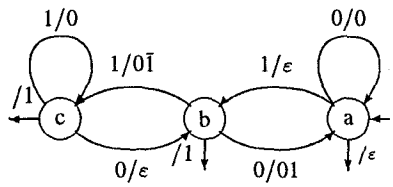

Figure 5. Right subsequential Booth canonical recoding

Example. Let $w = 11101101$. Then $\beta(w) = 1000\bar{1}0\bar{1}01$.

Note that in $\mathcal{B}$, the meaning of states b and c is not the same, although their terminal functions have the same value. State a means "no carry", state c means "there is a carry 1", and state b means "do as if a 1 has been read in advance and has not been output".

From the figure it is clear that, in the output, there are never two adjacent non zero digits, which implies that the coded representation can be seen as a radix 4 representation, with digit set $\{\bar{2}, \cdots, 2\}$.

The on-the-fly algorithm to compute the Booth canonical recoding is the following. Take three registers A, B, and C corresponding to states a, b, and c of $\mathcal{B}$. Initial conditions are

$$A[0] = \omega(a) = \varepsilon, \; B[0] = \omega(b) = 1,$$
$$C[0] = \omega(c) = 1.$$

Using the same notations as above, we define

p_{k+1}	$A[k+1]$	$B[k+1]$	$C[k+1]$
0	$(A[k], 0)$	$(A[k], 01)$	$(B[k], \varepsilon)$
1	$(B[k], \varepsilon)$	$(C[k], 0\bar{1})$	$(C[k], 0)$

The result of the computation is contained in register A.

3.4. Other number systems

We know consider less classical number systems, where the base is a negative integer or a quadratic complex number. For a discussion of the hardware implementation see [17].

Negative radix
Let r be an integer ≥ 2. It is known that any real number can be represented with radix $-r$ and digit set $\{0, \cdots, r-1\}$ without a sign (see [12]). It is also known that one can perform addition in parallel and on-line, using a redundant digit set $\{\bar{a}, \cdots, a\}$ where a is an integer such that $r/2 \leq a \leq r-1$ ([17], [8]). On the other hand, the conversion from $\{\bar{a}, \cdots, a\}$ to the canonical digit set $\{0, \cdots, r-1\}$ is right subsequential [8], and thus can be computed on-the-fly, with 3 registers. It cannot be computed on-line.

Base $i\sqrt{r}$
Let r be an integer ≥ 2. Every complex number is representable in base $i\sqrt{r}$ and digit set $\{0, \cdots, r-1\}$ (see [12], [11]). It is possible to perform addition in parallel and on-line with a redundant digit set $\{\bar{a}, \cdots, a\}$ where a is an integer such that $r/2 \leq a \leq r-1$ ([17], [8]). Conversion from $\{\bar{a}, \cdots, a\}$ to the canonical digit set $\{0, \cdots, r-1\}$ cannot be computed on-line, but is right subsequential [8], and thus can be computed on-the-fly.

Base $-1+i$
It is known that every complex number has a representation in base $-1+i$ and digit set $\{0, 1\}$ ([11]) and that parallel and on-line addition are possible with digit set $\{\bar{2}, \cdots, 2\}$ or $\{\bar{3}, \cdots, 3\}$ ([17], [8]). It has also been shown in [10] that on-line addition with digit set $\{\bar{1}, 0, 1\}$ is theoretically possible, but practically difficult. Nevertheless, conversion in base $-1+i$ between digit set $\{\bar{a}, \cdots, a\}$, $1 \leq a \leq 3$, into canonical digit set $\{0, 1\}$ is not on-line computable, is right subsequential [8], and is thus on-the-fly computable. In [1] it is shown how to obtain the $(-1+i)$-representation of a Gaussian integer from the 2-representation of its real and imaginary part by a right sequential machine. Hence this process can be realized on-the-fly.

4. Conclusions

In Computer Arithmetic, on-the-fly algorithms have been used in cases where one requires that some process be computed MSDF, but where it is not possible to achieve this task by an on-line algorithm. Our purpose here is to give a theoretical point of view on this notion, allowing us to show that functions which are on-the-fly computable in the sense we have defined are very simple; in particular, they always stay within the domain of functions computable by finite state automaton. At the same time, subsequential functions are quite well studied in Automata Theory, and some of their properties could be useful for the efficiency of on-the-fly algorithms. Finally, we believe that our result provides an easy way to obtain such algorithms, since right subsequential functions are very natural.

References

[1] J.-P. Allouche, E. Cateland, W. Gilbert, and H.-O. Peitgen. Automatic maps in exotic numeration systems. *Math. Systems Theory*, to appear.

[2] A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, 10:389–400, 1961.

[3] J. Berstel. *Transductions and Context-free Languages*. Teubner, 1979.

[4] A. Booth. A signed binary multiplication technique. *Quart. J. Mech. Appl. Math.*, 4:236–240, 1951.

[5] C. Choffrut. A generalization of Ginsburg and Rose's characterization of gsm mappings. In *ICALP '79*, number 71 in L.N.C.S., pages 88–103, 1979.

[6] S. Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, 1974.

[7] M. Ercegovac and T. Lang. On-the-fly conversion of redundant into conventional representations. *I.E.E.E. Trans. on Computers*, C 36:895–897, 1987.

[8] Ch. Frougny. Parallel and on-line addition in negative base and some complex number systems. In *Euro-Par'96*, number 1124 in L.N.C.S., pages 175–182, 1996.

[9] Ch. Frougny and J. Sakarovitch. Synchronisation déterministe des automates à délai borné. *T.C.S.*, to appear, 1997.

[10] Y. Herreros. *Contribution à l'arithmétique des ordinateurs*. Ph.d. dissertation, I.N.P.G., Grenoble, France, 1991.

[11] I. Kátai and J. Szabó. Canonical number systems for complex integers. *Acta Sci. Math.*, 37:255–280, 1975.

[12] D. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 1988.

[13] P. Kornerup. Digit-set conversions: Generalizations and applications. *I.E.E.E. Trans. on Computers*, 43:622–629, 1994.

[14] D. Matula. Basic digit sets for radix representation. *J.A.C.M.*, 29:1131–1143, 1982.

[15] P. Montuschi and L. Ciminiera. $n \times n$ carry-save multipliers without final addition. In *ARITH 11*, pages 54–61. I.E.E.E. Computer Society Press, 1993.

[16] J.-M. Muller. Some characterizations of functions computable in on-line arithmetic. *I.E.E.E. Trans. on Computers*, 43:752–755, 1994.

[17] A. Nielsen and J.-M. Muller. Borrow-save adders for real and complex number systems. In *Conf. Real Numbers and Computers*, pages 121–137, 1996.

[18] J. Slansky. Conditional sum addition logic. *IRE Transactions on Electronic Computers*, EC-9:226–231, 1960.

[19] K. Trivedi and M. Ercegovac. On-line algorithms for division and multiplication. *I.E.E.E. Trans. on Computers*, C 26:681–687, 1977.