On-the-Fly Algorithms and Sequential Machines

Christiane Frougny

Université Paris 8 and L.I.A.F.A. *

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 deterministically data in the least significant digit first fashion. Some applications to complex radix number systems are given.

Keywords: On-the-fly algorithm, sequential machine, digit set conversion.

1 Introduction

Conversion from a redundant into a conventional representation is important in Computer Arithmetic because on-line algorithms require redundant representations [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 [16] 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

*L.I.A.F.A. - Case 7014, 2 place Jussieu, 75251 Paris Cedex 05 - France. Email: `Christiane.Frougny@liafa.jussieu.fr`

result, corresponding to an assumed value of the carry. Using the well known technique of parallel prefix computation, on-the-fly algorithms can be implemented in time $\mathcal{O}(\log n)$ (see [13]).

The purpose of this contribution 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 $\{0, \ldots, r-1\}$, 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 $\{-a, \ldots, a\}$ 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 [17], [10]).

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 Booth canonical recoding and to conversion in number systems where the base is a negative integer, or some complex number. In particular, we fully give the on-the-fly algorithm converting redundant representations in base $i\sqrt{2}$ on the digit set $\{-1, 0, 1\}$ onto non-redundant representations on the digit set $\{0, 1\}$.

This paper is the full version of [9].

2 Definitions and results

Let r be the radix, r integer ≥ 2, and let X be a finite digit set, called *alphabet*. An integer p is represented by a finite string or a word $x_1 \cdots x_m$ of the free monoid X^* generated by X, with for $1 \leq k \leq m$, $x_k \in X$, and such that $p = \sum_{k=1}^{m} x_k r^{m-k}$. 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 natural number δ such that, to compute $y_1 \cdots y_n = \mu(x_1 \cdots x_m)$, it is necessary and sufficient to have $x_1, \ldots, x_{k+\delta}$ available to generate y_k, for $1 \leq k \leq n$. 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

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

where $\bar{1}$ denotes the signed digit -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 another definition [7]. 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 $\mu : X^* \longrightarrow Y^*$. A *register* on Y is a sequence $R = (R[k])_{k \geq 0}$ with $R[k]$ in Y^*. The function μ is said to be on-the-fly computable with N registers if there exist N registers $S_0, \ldots, S_{N-1}$ on Y such that the following conditions are satisfied.

- Let $p = x_1 \cdots x_m$ with $x_k \in X$ for $1 \leq k \leq m$ be an input word. Then for each

$1 \le k \le m$, $\mu(x_1 \cdots x_k) = S_0[k]$ (prefix computation).
• Initial conditions $S_i[0]$ are all different for every $0 \le i \le N-1$.
• Registers are updated at each step and the update is restricted to the form

$$S_i[k+1] = (S_j[k], u)$$

where $(S_j[k], u)$ denotes the result of the concatenation of $S_j[k]$ and of $u \in Y^*$, and u and j depend on the current input digit x_{k+1}.

For $0 \le i \le N-1$, $S_i[k]$ can be seen as the result of a computation with input $x_1 \cdots x_k$ "assuming a certain condition on the beginning of the computation". For an example where a condition is not always a carry, see the Booth canonical recoding below Section 3.2.

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 $p = x_1 \cdots x_m$, with $x_k \in X$ for $1 \le k \le m$, has $q \in Y^*$ for image by $\mathcal{M}$ if there exists a path in $\mathcal{M}$ starting in the initial state s_0

$$s_0 \xrightarrow{x_m/u_m} s_1 \xrightarrow{x_{m-1}/u_{m-1}} \cdots s_{m-1} \xrightarrow{x_1/u_1} s_m$$

with $u_j \in Y^*$, and such that $q = \omega(s_m)u_1 \cdots u_m$. 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.

Theorem 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, 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 $S = \{s_0, \ldots, s_{N-1}\}$. We thus need N registers denoted by $S_0, \ldots, S_{N-1}$, 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 $p = x_1 \cdots x_m$ 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 $x_{k+1} \in X$, if in $\mathcal{M}$ there is an edge of the form $s_i \xrightarrow{x_{k+1}/u} s_j$, $0 \leq j \leq N-1$, $u \in Y^*$, put $S_i[k+1] = (S_j[k], u)$ (note that, for a given value of x_{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 reverse of the output label of the unique path in $\mathcal{M}$ starting in s_i and with input label $x_k \cdots x_1$. 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 $\mathcal{M}$

$$s_i \xrightarrow{x_k/u_k} s_{i_1} \xrightarrow{x_{k-1}/u_{k-1}} \cdots s_{i_{k-1}} \xrightarrow{x_1/u_1} s_{i_k} \xrightarrow{/\omega(s_{i_k})} .$$

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 $1 \leq k \leq m$, $S_0[k] = \mu(x_1 \cdots x_k)$.

Conversely, let us suppose that $\mu : X^* \longrightarrow Y^*$ is on-the-fly computable with N registers $S_0, \cdots, S_{N-1}$ on 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 $x_{k+1} \in X$ are of the form $S_i[k+1] = (S_j[k], u)$, for i and j in $\{0, \ldots, N-1\}$ and $u \in Y^*$, we define in E an edge $s_i \xrightarrow{x_{k+1}/u} s_j$. The terminal function is defined by $\omega(s_i) = S_i[0]$. Clearly $\mathcal{M}$ is input deterministic, and as above, one verifies that $S_0[k] = \mu(x_1 \cdots x_k)$ if and only if there is a path in $\mathcal{M}$

$$s_0 \xrightarrow{x_k/u_k} s_{i_1} \xrightarrow{x_{k-1}/u_{k-1}} \cdots s_{i_{k-1}} \xrightarrow{x_1/u_1} s_{i_k} \xrightarrow{/\omega(s_{i_k})}$$

with $\mu(x_1 \cdots x_k) = \omega(s_{i_k}) u_1 \cdots u_k$. ■

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, 15].

Now we show a general result on radix r conversion onto the canonical digit set $\{0, \ldots, r-1\}$. It is more or less folklore, but we give a proof of it for the convenience of the reader.

Proposition 1 . *Let r be an integer ≥ 2, let X be any finite set of digits, and let $Y = \{0, \ldots, r-1\}$. 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. Let $M = \max\{|x-y|;\ x \in X,\ y \in Y\}$, and let $\gamma = M/(r-1)$. One defines a right subsequential machine $\mathcal{R} = (S, X \times Y, E, s_0, \omega)$ as follows. The set of states is $S = \{s \in \mathbf{Z} \mid |s| < \gamma\}$. The initial state is $s_0 = 0$. Let s be a state, and let $x \in X$ be an input digit. By the Euclidean division of $s + x$ by r there exist unique $y \in Y$ and $t \in \mathbf{Z}$ such that $s + x = rt + y$. We have $t = (s + x - y)/r$, thus $|t| \leq (|s| + |x-y|)/r < (\gamma + M)/r = \gamma$, and so $t \in S$. Thus we define an edge $s \xrightarrow{x/y} t$ in E. For any state s, the terminal function $\omega(s)$ is taken as 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. Now let $p = x_1 \cdots x_m$ be a word of X^*. Starting in initial state $s_0 = 0$, and reading from right to left, we take the unique path

$$s_0 \xrightarrow{x_m/y_m} s_1 \xrightarrow{x_{m-1}/y_{m-1}} \cdots s_{m-1} \xrightarrow{x_1/y_1} s_m.$$

Since, for $1 \leq j \leq m$, $s_{m-j} + x_j = r s_{m-j+1} + y_j$, we get $\sum_{j=1}^{m} x_j r^{m-j} = \sum_{j=1}^{m} y_j r^{m-j} + s_m$, thus $\psi(p) = \omega(s_m) y_1 \cdots y_m \in Y^*$. Note that some of the states may be useless. ∎

In on-line arithmetic, the redundant digit set is usually of the form $X = \{\bar{a}, \ldots, a\}$, 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.

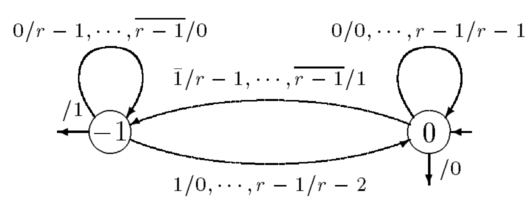

Figure 1. Right subsequential conversion from $\{\overline{r-1}, \ldots, r-1\}$ to r's complement

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

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 $\chi : \{\bar{1}, 0, 1\}^* \longrightarrow \{0, 1\}^*$ be the conversion function between radix 2 redundant representations and 2-complement representations. Below is the minimal right subsequential machine $\mathcal{C}$ realizing χ. The input alphabet is $X = \{\bar{1}, 0, 1\}$, the output alphabet is $Y = \{0, 1\}$, the set of states is $S = \{a, b\}$, the initial state is a, the terminal function is defined by $\omega(a) = 0$ and $\omega(b) = 1$.

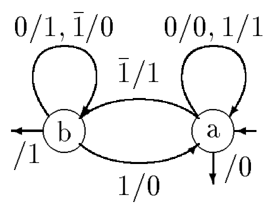

Figure 2. Right subsequential radix 2 conversion of redundant into conventional representation

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 the result is prefixed by $\omega(b) = 1$, to get the conversion for a negative number.

Example. Let us consider 4-digit input integers. Let $w = \bar{1}0\bar{1}1$. Then $(w)_2 = -9$. We have in the automaton $\mathcal{C}$

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

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

Following the method exposed in Theorem 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

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

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

3.2 Booth canonical recoding

Given a binary representation, the Booth canonical recoding consists of 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 $\varphi : X^* \longrightarrow Y^*$ realized by the following machine $\mathcal{B}$, which is minimal.

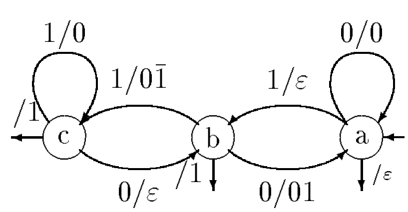

Figure 4. Right subsequential Booth canonical recoding

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

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 "a 1 has been read 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 $\{\bar{2}, \ldots, 2\}$.

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

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

The result of the computation is contained in register A.

3.3 Other number systems

We now consider less classical number systems, where the base is a negative integer or a quadratic complex number. Addition in base -2, $i\sqrt{2}$, $2i$, and $-1+i$ has been shown to be computable in constant time in parallel in [18].

Negative radix

Let r be an integer ≥ 2. It is known that any real number can be expressed with radix $-r$ and digit set $\{0, \ldots, r-1\}$ without a sign (see [12]). Using a redundant digit set $\{\bar{a}, \ldots, a\}$ where a is an integer such that $r/2 \leq a \leq r-1$ one can perform addition in parallel; under the same hypothesis addition is computable by an on-line finite state automaton [8]. On the other hand, the conversion from $\{\bar{a}, \ldots, a\}$ to the canonical digit set $\{0, \ldots, r-1\}$ is right subsequential [8], and thus can be computed on-the-fly, with 3 registers. It cannot be computed on-line.

Note also that the conversion between radix r and radix $-r$ expansions is right subsequential [1], and thus is on-the-fly computable.

Base $i\sqrt{r}$

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

Example. Let $r = 2$ and let $\beta = i\sqrt{2}$. The canonical digit set associated with β is $Y = \{0,1\}$, and the minimally redundant one is $X = \{\bar{1},0,1\}$. We give the construction of the right subsequential machine $\mathcal{M} = (S, X \times Y, E, s_0, \omega)$ realizing the conversion in base β from X to Y.

• The set of states is $S = \{s_{c,d} \mid c, d \in X\}$. State $s_{c,d}$ represents the number $\beta c + d$. The initial state is $s_0 = s_{0,0}$.

• Let $s_{c,d} \in S$ and let $x \in X$. By the Euclidean division of $d + x$ by -2 there exist unique $y \in Y$ and $e \in X$ such that $d + x = -2e + y$. Thus $\beta c + d + x = \beta(\beta e + c) + y$, since $\beta^2 = -2$. We then define in the machine $\mathcal{M}$ an edge $s_{c,d} \xrightarrow{x/y} s_{e,c}$.

• The terminal function is defined by $\omega(s_{c,d}) = cd$ if c and d are in Y, $\omega(s_{\bar{1},d}) = 101d$ if $d \in Y$, $\omega(s_{c,\bar{1}}) = 1c1$ if $c \in Y$, and $\omega(s_{\bar{1},\bar{1}}) = 1111$.

From that machine we derive an on-the-fly algorithm computing the conversion. There are 9 registers, of the form $S_{c,d}$, for c and d in Y. Initial conditions are $S_{c,d}[0] = \omega(s_{c,d})$.

x_{k+1}	0	1	$\bar{1}$
$S_{0,0}[k+1]$	$(S_{0,0}[k],0)$	$(S_{0,0}[k],1)$	$(S_{1,0}[k],1)$
$S_{0,1}[k+1]$	$(S_{0,0}[k],1)$	$(S_{\bar{1},0}[k],0)$	$(S_{0,0}[k],0)$
$S_{0,\bar{1}}[k+1]$	$(S_{1,0}[k],1)$	$(S_{0,0}[k],0)$	$(S_{1,0}[k],0)$
$S_{1,0}[k+1]$	$(S_{0,1}[k],0)$	$(S_{0,1}[k],1)$	$(S_{1,1}[k],1)$
$S_{1,1}[k+1]$	$(S_{0,1}[k],1)$	$(S_{\bar{1},1}[k],0)$	$(S_{0,1}[k],0)$
$S_{1,\bar{1}}[k+1]$	$(S_{1,1}[k],1)$	$(S_{0,1}[k],0)$	$(S_{1,1}[k],0)$
$S_{\bar{1},0}[k+1]$	$(S_{0,\bar{1}}[k],0)$	$(S_{0,\bar{1}}[k],1)$	$(S_{1,\bar{1}}[k],1)$
$S_{\bar{1},1}[k+1]$	$(S_{0,\bar{1}}[k],1)$	$(S_{\bar{1},\bar{1}}[k],0)$	$(S_{0,\bar{1}}[k],0)$
$S_{\bar{1},\bar{1}}[k+1]$	$(S_{1,\bar{1}}[k],1)$	$(S_{0,\bar{1}}[k],0)$	$(S_{1,\bar{1}}[k],0)$

The result of the computation is contained in register $S_{0,0}$.

For instance, let $(11\bar{1}\cdot)_{i\sqrt{2}}$ be a representation on $X = \{\bar{1},0,1\}$ of $z = -3 + i\sqrt{2}$. Below is the on-the-fly computation of the conversion into $(1010011\cdot)_{i\sqrt{2}}$.

k	0	1	2	3
x_k		1	1	$\bar{1}$
$S_{0,0}$	00	001	0011	1010011
$S_{0,1}$	01	10100	10110	00110
$S_{0,\bar{1}}$	101	000	0010	1010010
$S_{1,0}$	10	011	101001	1111001
$S_{1,1}$	11	10110	111100	101100
$S_{1,\bar{1}}$	111	010	101000	1111000
$S_{\bar{1},0}$	1010	1011	0001	1010001
$S_{\bar{1},1}$	1011	11110	10100	00100
$S_{\bar{1},\bar{1}}$	1111	1010	0000	1010000

Base $-1+i$

It is known that every complex number has a representation in base $-1+i$ and digit set $\{0,1\}$. In particular every Gaussian integer has a unique representation of the form $\sum_{k=0}^{n} d_k(-1+i)^k$ ([11]). Parallel and on-line addition are possible with digit set $\{\bar{2},\ldots,2\}$ or $\{\bar{3},\ldots,3\}$ ([18], [8]). Nevertheless, conversion in base $-1+i$ between digit set $\{\bar{a},\ldots,a\}$, $1 \leq a \leq 3$, 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. As a corollary of our result, this process can be realized by an on-the-fly algorithm.

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.

Acknowledgements. We want to thank the referees for suggestions which greatly improved the manuscript.

References

[1] J.-P. Allouche, E. Cateland, W.J. Gilbert, and H.-O. Peitgen. Automatic maps in exotic numeration systems. *Theory of Computing Systems*, 30:285–331, 1997.

[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.D. 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 Lecture Notes in Computer Science, pages 88–103, 1979.

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

[7] M.D. 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 Lecture Notes in Computer Science, pages 175–182, 1996.

[9] Ch. Frougny. On-the-fly algorithms and sequential machines. In *13th IEEE Symposium on Computer Arithmetic*, pages 260–265. I.E.E.E. Computer Society Press, 1997.

[10] Ch. Frougny and J. Sakarovitch. Synchronisation déterministe des automates à délai borné. *Theoretical Computer Science*, 191:61–77, 1998.

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

[12] D.E. 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] M. Mohri. Minimization of sequential transducers. In *STACS '94*, number 807 in Lecture Notes in Computer Science, pages 151–163, 1994.

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

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

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

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