Publ. Math. Debrecen
79/3-4 (2011), 539–551
DOI: 10.5486/PMD.2011.5134

Prime factorization by interval-valued computing

By BENEDEK NAGY (Debrecen) and SÁNDOR VÁLYI (Nyíregyháza)

This paper is dedicated to Attila Pethő on his 60th birthday

Abstract. Interval-valued computing is a new theoretical computing paradigm. Hard problems, e.g. satisfiability of quantified Boolean formulae, can be solved in an efficient way deploying the massive parallelism of this paradigm. In this paper, we consider the prime factorization problem. We show an interval-valued algorithm that computes a proper divisor of the input number (or 1 in case the input is a prime). This interval-valued algorithm works in polynomial number of steps within this paradigm.

1. Introduction

There are intractable problems that traditional, i.e., Neumann-type architecture, computers cannot solve efficiently. For some classes of problems, such as NP-complete, PSPACE etc. it seems that there will not be any method to solve every instance in deterministic polynomial time depending on the length of the input. In number theory, there are also some hard problems, such as prime factorization. Although prime testing is in P, i.e., the decision whether a given number is a prime, or not, can be solved in traditional computers in polynomial time [1]; the factorization problem, i.e., to give a proper divisor if the number is not prime, cannot be solved in such an effective way (at least, recently we does not have such an algorithm). These number theoretic problems are effectively

Mathematics Subject Classification: 68Q05 (Models of computation).
Key words and phrases: unconventional computing, interval-valued computing, factorization.

used in applications such as cryptography. There are various new computing paradigms that attack these hard problems with success, at least in theory. There are methods based on inspiration from Biology (e.g., DNA-computing [2], membrane computing [12]), from Physics (e.g., Quantum computing [4]) and from other phenomena of the Nature. It is known that quantum computing, in theory, is able to solve prime factorization in polynomial time. In [13] a result is published: 15 is factorized in practice. Unfortunately there is not known any newer results about this topic. There are algorithms for P-systems (i.e., membrane computing) also, that solve factorization effectively, in theory [11], [3]. The efficiency of most of these new paradigms come from a massive parallelism built in the system. In this paper we use the Interval-valued paradigm which was introduced in [5]. This paradigm is close to the classical paradigm, but dropping the bound of the number of bits in a byte. The basic data unit is the interval-value that is a finite union of subintervals (components) over the unit interval $[0, 1)$. Logical operations are straightforward generalizations of the classical operations, moreover shift operations is also used to carry some pieces of information to other parts of the unit interval. The product operation allows to raise the density of information. This operation is analogous with the zoom operation of the optical computing [15]. With this paradigm NP-complete problems can be solved by polynomial number of steps (SAT in [5] in linear number of steps, the tripartite matching in [14]), the PSPACE-complete quantified SAT is also solved in a linear number of steps in [6]. The connection between interval-valued computing system and the class PSPACE is established in [9]. The system can also be used to visualize computations or can be used as a visual computing system [7], [8]. [10] provides on overview of this paradigm.

By this paper we also demonstrate that elegant, easily comprehensible solution can be given for other difficult problems. Moreover, the proof of correctness of that interval-valued computations is not really harder than the proof of correctness of corresponding classical algorithms.

In the next section we recall the interval-valued paradigm in a formal way and we extend it to allow computations of discrete functions, not only decision problems. In Section 3 we present our algorithm to solve the prime factorization problem, finally in Section 4 there are some concluding remarks.

2. Preliminaries: interval-valued computations

To have a self-contained paper, in this section, we recall the definitions of [9] needed to read the following sections of this paper. First we define what an interval-value means. Then we present the allowed operations which can be used to build and evaluate computation sequences. We also give the notions concerning decidability and computational complexity. Moreover we extend the notion to compute discrete functions.

2.1. Interval-values. We note in advance that we do not distinguish interval-values (specific functions from $[0,1)$ into $\{0,1\}$) from their subset representations (subsets of $[0,1)$) and we always use the more convenient notation.

Definition 1. The set $\mathbb{V}$ of *interval-values* coincides with the set of finite unions of [)-type subintervals of $[0,1)$.

Definition 2. The set $\mathbb{V}_0$ of *specific interval-values* coincides with

$$\left\{ \bigcup_{i=1}^{k} \left[\frac{l_i}{2^m}, \frac{1+l_i}{2^m} \right) : m \in \mathbb{N},\ k \leq 2^m,\ 0 \leq l_1 < \ldots < l_k < 2^m \right\}. \qquad (*)$$

Similarly, let $\mathbb{V}_n$ be the set of interval-values that can be represented by $(*)$ in a way that for the used maximal value m the condition $m \leq n$ holds.

We note that the set of finite unions includes the empty set ($k = 0$), that is, $\emptyset$ is also an allowed interval-value.

2.2. Operators on interval-values. If we consider interval-values as subsets of $[0,1)$, then the set-theoretical operations such as complementation ($\overline{A}$), union ($A \cup B$) and intersection ($A \cap B$) on $\mathbb{V}$ are included. $(\mathbb{V}, \bar{}, \cup, \cap)$ forms an infinite Boolean set algebra with these operations, $\mathbb{V}_0$ is one of its infinite subalgebra, while the systems based on $\mathbb{V}_n$ ($n > 0$) are finite subalgebras.

Definition 3. The first component of an interval value $A \in \mathbb{V}$, $A \neq \emptyset$, is defined as interval value $[t,s)$ where t and $s \in [0,1]$ satisfy that $[0,t) \cap A = \emptyset$, $[t,s) \subset A$ and $\forall s' > s : [t,s') \not\subset A$. Now the function $Flength : \mathbb{V} \to \mathbb{R}$ is defined as follows. If $A = \emptyset$, then $Flength(A) = 0$. Otherwise $Flength(A) = s - t$, where $[t,s)$ is the first component of A.

Intuitively, this function provides the length of the left-most component (included maximal subinterval) of an interval-value A. *Flength* helps us to define the binary shift operators on $\mathbb{V}$. The *left-shift* operator will shift the first interval-value to the left by the first-length of the second operand and remove the part

which is shifted out of the interval $[0,1)$. As opposed to this, the *right-shift* operator is defined in a circular way, i.e. the parts shifted above 1 will appear at the lower end of $[0,1)$. In this definition we write interval-values in their "characteristic function" notation.

Definition 4. The binary operators $Lshift$ and $Rshift$ on $\mathbb{V}$ are defined in the following way. If $x \in [0,1)$ and $A, B \in \mathbb{V}$ then

$$Lshift(A,B)(x) = \begin{cases} A(x + Flength(B)) & \text{if } \ 0 \le x + Flength(B) < 1, \\ 0 & \text{in other cases;} \end{cases}$$

and $Rshift(A,B)(x) = A(frac(x - Flength(B)))$.

Here the function *frac* gives the fractional part of a real number, i.e., $frac(x) = x - \lfloor x \rfloor$, where $\lfloor x \rfloor$ is the greatest integer which is not greater than x.

Definition 5. Let A and B be interval-values and $x \in [0,1)$. Then the (fractalian) product $B * A$ includes x if and only if $B(x) = 1$ and $A(\frac{x - x_B}{x^B - x_B}) = 1$, where x_B denotes the lower end-point of the B-component including x, and x^B denotes the upper end-point of this component, that is, $[x_B, x^B)$ is the maximal subinterval of B containing x.

We can give this operation in a more descriptive manner. If A contains k interval components with ends $a_{i,1}$, $a_{i,2}$ $(1 \le i \le k)$ and B contains l components with ends $b_{i,1}$, $b_{i,2}$ $(1 \le i \le l)$, then we determine the value of $C = A * B$ as follows: we set the number of components of C to be $k \cdot l$. For this process we can use double indices for the components of C. The lower and higher end-points of the ij-th component are $a_{i1} + b_{j1}(a_{i2} - a_{i1})$ and $a_{i1} + b_{j2}(a_{i2} - a_{i1})$, respectively.

The idea and the role of this operation is similar to the unlimited shrinking of 2-dimensional images in [15]. It will be used to connect interval-values of different resolution (i.e, increase n in the actually used $\mathbb{V}_n$).

2.3. Syntax and semantics of computation sequences. This formalism is of Boolean network style. As usual, the length of a sequence S is denoted by $|S|$ and its i-th element by S_i. If $j \le |S|$ then the j-length prefix of S is denoted by $S_{\to j}$.

Definition 6. An *interval-valued computation sequence* is a nonempty finite sequence S satisfying $S_1 = FIRSTHALF$ and further, for any $i \in \{2, \ldots, |S|\}$, S_i is (op, l, m) for some $op \in \{AND, OR, LSHIFT, RSHIFT, PRODUCT\}$ or S_i is (NOT, l) or $(OUTPUT, l)$ where $\{l, m\} \subseteq \{1, \ldots, i-1\}$.

One of the complexity measures of a given computation is the *bit height* of a computation. It is the minimal value n such that all the interval-values of the computation are in $\mathbb{V}_n$.

The semantics of interval-valued computation sequences is defined by induction on the length of the sequences. The *interval-value* of such a sequence S is denoted by $\|S\|$ and defined by induction on the length of the computation sequence, as follows.

Definition 7. First, we fix $\|(FIRSTHALF)\|$ as $\left[0, \frac{1}{2}\right)$. Second, if S is an interval-valued computation sequence and $|S|$ is its length, then

$$\|S\| = \begin{cases} \|S_{\to j}\| \cap \|S_{\to k}\|, & \text{if } S_{|S|} = (AND, j, k), \\ \|S_{\to j}\| \cup \|S_{\to k}\|, & \text{if } S_{|S|} = (OR, j, k), \\ \|S_{\to j}\| * \|S_{\to k}\|, & \text{if } S_{|S|} = (PRODUCT, j, k), \\ Rshift(\|S_{\to j}\|, \|S_{\to k}\|), & \text{if } S_{|S|} = (RSHIFT, j, k), \\ Lshift(\|S_{\to j}\|, \|S_{\to k}\|), & \text{if } S_{|S|} = (LSHIFT, j, k), \\ \|S_{\to j}\|, & \text{if } S_{|S|} = (OUTPUT, j), \\ \overline{\|S_{\to j}\|}, & \text{if } S_{|S|} = (NOT, j). \end{cases}$$

Here the system of [9] is extended with an instruction to write (i.e., generate) the output as we detail below.

2.4. Computing a discrete function by interval-values. We fix a possible formulation which is suitable for the present purpose. The semantics of writing the output is the following. The output sequence is an element of $\{0,1\}^*$, initially the empty sequence. Let $S_1 \dots S_n$ denote the computation sequence. If $S_j = (OUTPUT, i)$ where $i < j$ then $\|S_{\to j}\| = \|S_{\to i}\|$ and as a side effect, 1 is appended to the the output sequence if S_j is nonempty, otherwise 0 is appended to it. The answer of a computation sequence is its output sequence produced during the computation. Let $f : \{0,1\}^* \to \{0,1\}^*$. We say that f is computable by an interval-valued computation if and only if there exists a logspace algorithm $\mathcal{B}$ that for each possible input ($w \in \{0,1\}^*$) constructs a computation sequence that generates the output sequence $f(w)$.

3. Prime factorization by interval-values

In this section we solve the integer factorization problem, i.e., we give interval-valued computation sequence that give, as the output, a proper divisor of an input integer N or the result is 1 in case N is a prime number.

Theorem 1. *The prime factorization can be computed by an interval-valued computation of quadratic size.*

Further in this section we prove this theorem in a constructive way:

Let $b_1 \dots b_n$ be the binary representation of the input integer N. (One can assume that $N \geq 4$ and $n \geq 3$.) We give a logspace algorithm $\mathcal{B}$ that constructs an interval-valued computation sequence S from N with an output bit sequence $d_1 \dots d_n$ that is the binary representation of the greatest divisor of N different from N.

$\mathcal{B}$ starts its work by writing the following. First, fix S_1 as $FIRSTHALF$ and S_2 as $(RIGHT, 1, 1)$. Then, for each $i \in \{1, \dots, n\}$, if $b_i = 1$ then put $S_{2+i} := (OR, 1, 2)$ else $S_{2+i} := (AND, 1, 2)$. We denote the indices of the subsequence $S_3, S_4, \dots, S_{2+n}$ by $i(1), i(2), \dots, i(n)$. In this way we have represented the input:

Lemma 1. *For each $k \in \{1, \dots, n\}$:*

$$\|S_{\to i(k)}\| = \begin{cases} [0,1) & \text{if } \ b_k = 1 \text{ and} \\ \emptyset & \text{if } \ b_k = 0. \end{cases}$$

Proof. It is straightforward: $\|S_{\to 1}\| = \left[0, \frac{1}{2}\right)$, $\|S_{\to 2}\| = \left[\frac{1}{2}, 1\right)$ and the union of the last two interval-value is $[0, 1)$ while their intersection is $\emptyset$. □

$\mathcal{B}$ continues its job. It defines $S_{2+n+1}, \dots, S_{2+n+(3n-2)}$ as follows. $S_{2+n+1} = (AND, 1, 1)$. For all positive integers $k < 2n$,

$$S_{2+n+3k-1} = (PRODUCT, 2+n+3k-2, 1),$$

$$S_{2+n+3k} = (RSHIFT, 2+n+3k-2, 2+n+3k-1) \quad \text{and}$$

$$S_{2+n+3k+1} = (OR, 2+n+3k, 2+n+3k-1).$$

The index sequence $2+n+1, 2+n+4, \dots, 2+n+(6n-2)$ will be denoted by $a(1), a(2), \dots, a(2n)$. By induction on k one can establish the following statement.

Lemma 2. *For all integer $k \in \{1, \dots, 2n\}$:*

$$\|S_{\to a(k)}\| = \|S_{\to 2+n+(3k-2)}\| = \bigcup_{l=0}^{2^{k-1}-1} \left[\frac{2l}{2^k}, \frac{2l+1}{2^k}\right).$$

PROOF. The proof goes by induction. Base of the induction: $k = 1$:

$$\|S_{\to 2+n+1}\| = \|FIRSTHALF\| \cap \|FIRSTHALF\|$$
$$= \left[0, \frac{1}{2}\right) = \bigcup_{l=0}^{0} \left[\frac{2l}{2}, \frac{2l+1}{2}\right) = \bigcup_{l=0}^{2^0-1} \left[\frac{2l}{2^1}, \frac{2l+1}{2^1}\right).$$

Now, for the induction we assume that the statement holds for all positive integers up to k ($k < 2n$). Let us prove it for $k+1$:

$$\|S_{\to a(k)+1}\| = \bigcup_{l=0}^{2^{k-1}-1} \left[\frac{2l}{2^k}, \frac{2l+1}{2^k}\right) * \left[0, \frac{1}{2}\right) = \bigcup_{l=0}^{2^{k-1}-1} \left[\frac{2l}{2^k}, \frac{2l}{2^k} + \frac{1}{2^{k+1}}\right);$$

$$\|S_{\to a(k)+2}\| = Rshift\left(\|S_{\to a(k)+1}\|, \|S_{\to a(k)}\|\right)$$
$$= Rshift\left(\bigcup_{l=0}^{2^{k-1}-1} \left[\frac{2l}{2^k}, \frac{2l}{2^k} + \frac{1}{2^{k+1}}\right), \left[0, \frac{1}{2^k}\right)\right)$$
$$= \bigcup_{l=0}^{2^{k-1}-1} \left[\frac{2l+1}{2^k}, \frac{2l+1}{2^k} + \frac{1}{2^{k+1}}\right); \quad \text{finally}$$

$$\|S_{\to a(k+1)}\| = \|S_{\to a(k)+3}\|$$
$$= \bigcup_{l=0}^{2^{k-1}-1} \left[\frac{2l}{2^k}, \frac{2l}{2^k} + \frac{1}{2^{k+1}}\right) \cup \bigcup_{l=0}^{2^{k-1}-1} \left[\frac{2l+1}{2^k}, \frac{2l+1}{2^k} + \frac{1}{2^{k+1}}\right)$$
$$= \bigcup_{j=0}^{2^k-1} \left[\frac{j}{2^k}, \frac{j}{2^k} + \frac{1}{2^{k+1}}\right) = \bigcup_{i=0}^{2^k-1} \left[\frac{2i}{2^{k+1}}, \frac{2i+1}{2^{k+1}}\right). \qquad \square$$

In this way all variations of $2n$ independent bits can be represented simultaneously by the interval-values $\|S_{\to a(1)}\|, \|S_{\to a(2)}\|, \ldots, \|S_{\to a(2n)}\|$ in the following sense:

Lemma 3. *For each bit sequence $x_1 \ldots x_{2n}$ there exists $r \in [0,1)$ that for any $k \in \{1, \ldots, 2n\}$: $r \in \|S_{\to a(k)}\|$ if and only if $x_k = 1$.*

PROOF. We show that $r = \sum_{i=1}^{2n} \frac{1-x_i}{2^i}$ is a right choice.

$$\sum_{i=1}^{2n} \frac{1-x_i}{2^i} \in \bigcup_{l=0}^{2^{k-1}-1} \left[\frac{2l}{2^k}, \frac{2l+1}{2^k}\right) \Leftrightarrow$$

$$\sum_{i=1}^{k-1} 2^{k-i} \cdot \frac{(1-x_i)}{2^k} + \frac{1-x_k}{2^k} + \sum_{i=k+1}^{2n} \frac{1-x_i}{2^i} \in \bigcup_{l=0}^{2^{k-1}-1} \left[\frac{2l}{2^k}, \frac{2l+1}{2^k}\right) \Leftrightarrow$$

$$(1-x_k) \quad \text{is even} \Leftrightarrow x_k = 1. \qquad \square$$

Definition 8. For $k \in \{1,\dots,n\}$, $x_k(r) := (r \in \|S_{\to a(k)}\|)$ and $y_k(r) := (r \in \|S_{\to a(n+k)}\|)$. Let the bit sequence $x_1(r)\dots x_n(r)$ be denoted by $X(r)$, similarly, $y_1(r)\dots y_n(r)$ be denoted by $Y(r)$. For any bit sequence $BS = b_1\dots b_n$, let $\#BS$ denote the integer whose binary representation is BS.

The algorithm $\mathcal{B}$ continues the construction of the computation sequence by pointwise simulating an n-bit sequence multiplication Boolean circuit. This part needs (at most) a quadratic amount of operations, moreover, it does not apply $PRODUCT$ or $SHIFT$: only pointwise Boolean operations are used. Let $m(1),\dots,m(2n)$ denote the sequence of the indices of the produced interval-values that represent the bits of the result of the multiplication (the most significant value $m(1)$ is on the left).

Lemma 4. *$\forall r \in [0,1)$ $\forall k \in \{1,\dots,2n\} : r \in \|S_{\to m(k)}\|$ if and only if the k-th bit in the $2n$-length binary representation of $\#X(r) \cdot \#Y(r)$ is 1.*

Proof. The standard schoolbook multiplication algorithm of binary numbers is appropriate to this computation. □

Thereafter, $\mathcal{B}$ appends to the constructed computation sequence the partial and the final results of a pointwise testing for $r \notin \|S_{\to m(2n)}\|$ and $\dots r \notin \|S_{\to m(n+1)}\|$, then also checking $r \in \|S_{\to m(n+k)}\| \Leftrightarrow r \in \|S_{i(k)}\|$, for all $k \in \{1,\dots,n\}$.

More definitively, we give the computation sequence.

For any $k \in \{1,\dots,n\}$, let

$$S_{m(n)+k} = (NOT, m(n+k)),$$
$$S_{m(n)+n+1} = (AND, m(n)+1, m(n)+2).$$

Further, for any $k \in \{2,\dots,n-1\}$,

$$S_{m(n)+n+k} = (AND, m(n)+k+1, m(n)+n+k-1),$$
$$S_{m(n)+2n} = (OR, 1, 2).$$

For any $k \in \{1,\dots,n\}$:

$$S_{m(n)+2n+(5k-4)} = (AND, i(k), m(k)),$$
$$S_{m(n)+2n+(5k-3)} = (NOT, i(k)),$$
$$S_{m(n)+2n+(5k-2)} = (NOT, m(k)),$$
$$S_{m(n)+2n+(5k-1)} = (AND, m(n)+2n+(5k-2), m(n)+2n+(5k-3)),$$
$$S_{m(n)+2n+5k} = (OR, m(n)+2n+(5k-1), m(n)+2n+(5k-4)).$$

Finally, $S_{m(n)+7n+1} = (AND, m(n) + 2n - 1, m(n) + 2n + 5)$ and for all $k \in \{2, \ldots, n\}$: $S_{m(n)+7n+k} = (AND, m(n) + 7n + k - 1, m(n) + 2n + 5k)$.

The final index $(m(n) + 8n)$ will be denoted by e – the e-th element of the computation sequence satisfies then the following property:

Lemma 5. *$\forall r \in [0, 1) : r \in \|S_{\to e}\|$ if and only if $\#X(r) \cdot \#Y(r) = N$.*

Proof.

$$\forall k \in \{1, \ldots, n\} \forall r \in [0, 1) : r \in \|S_{\to m(n)+k}\| \Leftrightarrow r \notin \|S_{\to m(k)}\|.$$

$$\forall k \in \{1, \ldots, n-1\} \forall r \in [0, 1) : r \in \|S_{\to m(n)+n+k}\| \Leftrightarrow r \notin \bigcup_{i=1}^{k+1} \|S_{\to m(i)}\|.$$

$$\|S_{\to m(n)+2n}\| = [0, 1).$$

$$\forall k \in \{1, \ldots, n\} \forall r \in [0, 1) : r \in \|S_{\to m(n)+2n+5k}\| \Leftrightarrow (r \in \|S_{\to m(k)}\| \Leftrightarrow r \in \|S_{\to i(k)}\|).$$

$$\forall k \in \{1, \ldots, n\} \forall r \in [0, 1) : r \in \|S_{\to m(n)+7n+k}\| \Leftrightarrow \forall j \in \{1, \ldots, k\}$$
$$(r \in \|S_{\to m(j)}\| \Leftrightarrow r \in \|S_{\to i(j)}\|) \wedge \forall u \in \{1, \ldots, n\} r \notin \|S_{\to m(u+j)}\|.$$

As a special case of the last statement,

$$\forall r \in [0, 1)\ r \in \|S_{\to m(n)+8n}\| \Leftrightarrow \forall j \in \{1, \ldots, n\}$$
$$(r \in \|S_{\to m(j)}\| \Leftrightarrow r \in \|S_{\to i(j)}\|) \wedge \forall u \in \{1, \ldots, n\} r \notin \|S_{\to m(u+j)}\|.$$

This fact combined with Lemma 4 provides the result of Lemma 5. □

Lemma 6. *$\forall r_1, r_2 \in [0, 1) : (r_1 \leq r_2 \Leftrightarrow \#X(r_1) \geq \#X(r_2))$. Further, if $|r_1 - r_2| > \frac{1}{2^n}$ then $\#X(r_1) \neq \#X(r_2)$.*

Proof. It follows from the fact that for any bit sequence $x_1 \ldots x_n$ and all $r \in [0, 1)$, $X(r) = x_1 \ldots x_n$ if and only if $r \in \left[\sum_{i=1}^{n} \frac{1-x_i}{2^i}, \sum_{i=1}^{n} \frac{1-x_i}{2^i} + \frac{1}{2^n}\right)$. □

Now the concept of first component is used from Definition 3, actually we compute it for some interval-values:

Lemma 7. *For arbitrary nonempty starting interval value $A \in \mathbb{V}$ of index k, the following computation ends with the first component of A if $0 \notin A$ but A is nonempty. That is, if $\|S_{\to k}\| = A$ then $\|S_{\to k+7}\|$ is the first component of A if*

$A \neq \emptyset$.

$k+1:\ (NOT, k)$,

$k+2:\ (LSHIFT, k, k+1)$,

$k+3:\ (LSHIFT, k+2, k+2)$,

$k+4:\ (RSHIFT, k+3, k+2)$,

$k+5:\ (RSHIFT, k+4, k+1)$,

$k+6:\ (NOT, k+5)$,

$k+7:\ (ANDk, k+6)$.

PROOF. If $A = [a, 1)$ with an $a > 0$, then $\emptyset = \|S_{\to k+3}\| = \ldots = \|S_{\to k+5}\|$, $\|S_{\to k+6}\| = [0, 1)$ so $\|S_{\to k+7}\| = [a, 1)$. If the first component of A is $[a, b)$ and $b < 1$, then there is $c > b$ that $[b, c) \cap A = \emptyset$. In this case we denote $A \cap [c, 1)$ by X. So, $A = [a, b) \cup X$ and $b < c$ and $X \cap [0, c) = \emptyset$. In this case

$\|S_{\to k}\| = [a, b) \cup X$,

$\|S_{\to k+1}\| = [0, a) \cup [b, 1) \setminus X$, so its first component $[0, a)$,

$\|S_{\to k+2}\| = [0, b-a) \cup (X - a)$ where $(X - a)$ denotes $\{x - a | x \in X\}$

$\|S_{\to k+3}\| = X - b$,

$\|S_{\to k+4}\| = X - a$,

$\|S_{\to k+5}\| = X$,

$\|S_{\to k+7}\| = ([a, b) \cup X) \cap ([0, 1) \setminus X) = [a, b)$. □

Lemma 8. (1) *If A is the first component of $\|S_{\to e}\|$, then $A \neq \emptyset$ but $0 \notin A$. Further, if $r \in A$, then $\#X(r) = N$.*

(2) *$\|S_{\to e}\| \setminus A$ is also nonempty and not including 0. Further, for the least $r \in \|S_{\to e}\| \setminus A$ holds that $\#X(r)$ is the largest divisor of N different from N.*

PROOF. (1) A is nonempty since always $N * 1 = N$ but $0 \notin A$ because N cannot be the square of the number whose binary representation is of the same length than N, particularly if all of the bits of that number is 1. ($X(0) = 1^n$.) The third part follows from Lemma 4, 5, 6 and 7.

(2) $\|S_{\to e}\| \setminus A$ is nonempty by $1 * N = N$. Of course, it cannot include 0. $\#X(r)$ is the largest divisor of N different from N because $N - 1$ does not divide N. Thus, if $\#X(r)$ divides N and less than N, then by Lemma 6 there exists some $s < r$ such that s is not an element of the first component but greater than the endpoint of this component. □

$\mathcal{B}$ appends to the above constructed computation sequence of length e the 7 operations described in Lemma 7 with a starting index $k = e$. So, by this lemma, $\|S_{e+7}\|$ is the first component of $\|S_e\|$. Then $\|S_{e+16}\|$ will be the second component of $\|S_e\|$ if $\mathcal{B}$ sets the computation steps below:

$$e+8: \ (NOT, e+7)$$
$$e+9: \ (AND, e, e+8)$$

and then applies again the steps $e+10, \ldots, e+16$ of Lemma 7 with $k = e+9$. Finally, $\mathcal{B}$ appends the following steps:

$$e+16+k: \ (AND, e+16, a(k)), \qquad \text{for all } k \in \{1, \ldots, n\} \text{ and}$$
$$e+16+n+k: \ (OUTPUT, e+16+k), \qquad \text{for all } k \in \{1, \ldots, n\}.$$

By the definition of output, $\|S_{\to e+16+k}\| \neq \emptyset$ if and only if the k-th output bit is 1.

We remark, that neither e nor the values $a(k), i(k)$ and $m(k)$ depend in any way on the observation of the computation steps but only on N. By the last lemma, we can show that for all $k \in \{1, \ldots, n\}$ holds that $\|S_{e+16+k}\| \neq \emptyset$ if and only if the k-th bit of the binary representation of the greatest divisor of N (different from N) is 1. That is, the prime factorization, formulated as a function, is computed by the given interval-valued computation algorithm $\mathcal{B}$.

The proof of Theorem 1 is finished.

4. Concluding remarks

The algorithm used for prime factorization is a uniform algorithm in the following sense. It generally produces the factorization of all numbers that can be represented on (at most) n bits. Only the parts at the beginning (coding the input N) should be changed for a new test; and only the linear part in the end from the equality test till the output generation should be recomputed (the middle, quadratic size part is independent of the input number in this sense, only the size of the input: n does matter.)

In the other side, it is clear that the given interval-valued implementation is not the most effective one but already this construction gives the aimed quadratic size upper limit. For example, implementing more sophisticated multiplication algorithms (like Karatsuba's algorithm) will improve efficiency.

Acknowledgements. The main result of this paper was initially presented at the conference NTA 2010: Number Theory and its Applications: An International Conference Dedicated to Kálmán Győry, Attila Pethő, János Pintz and András Sárközy. The first author is supported by the TÁMOP 4.2.1/B-09/1/KONV-2010-0007 project. The project is implemented through the New Hungary Development Plan co-financed by the European Social Fund, and the European Regional Development Fund.

References

[1] Manindra Agrawal, Neeraj Kayal and Nitin Saxena, PRIMES is in P, *Ann. of Math.* **160** (2004), 781–793.

[2] Martyn Amos, Theoretical and Experimental DNA Computation, Natural Computing Series, *Springer*, 2005.

[3] Erzsébet Csuhaj-Varjú, On the factorization problem and membrane computing, presented in NTA 2010: Number Theory and its Applications, Debrecen, 2010.

[4] Mika Hirvensalo, Quantum Computing, Natural Computing Series, (2nd edition), *Springer*, 2003.

[5] Benedek Nagy, An interval-valued computing device, CiE 2005, Computability in Europe: New Computational Paradigms, Amsterdam, Netherlands, 2005, 166–177.

[6] Benedek Nagy and Sándor Vályi, Solving a PSPACE-complete problem by a linear interval-valued computation, CiE 2006, Computability in Europe: Logical Approaches to Computational Barriers, University of Wales Swansea, UK, 2006, 216–225.

[7] Benedek Nagy and Sándor Vályi, Interval-valued computing as a visual reasoning system, DMS 2007, The Thirteenth International Conference on Distributed Multimedia Systems – VLC 2007, International Workshop on Visual Languages and Computing, *San Francisco, CA, USA*, 2007, 247–250.

[8] Benedek Nagy and Sándor Vályi, Visual reasoning by generalized interval-values and interval temporal logic, VLL 2007, Workshop on Visual Languages and Logic – VL/HCC 07, IEEE Symposium on Visual Languages and Human Centric Computing, CEUR Workshop Proceedings Vol-274, *Coeur d'Aléne, Idaho, USA*, 2007, 13–26.

[9] Benedek Nagy and Sándor Vályi, Interval-valued computations and their connection with PSPACE, *Theoret. Comput. Sci.* **394** (2008), 208–222.

[10] Benedek Nagy, Effective Computing by Interval-values, INES 2010, 14th IEEE International Conference on Intelligent Engineering Systems, *Las Palmas of Gran Canaria, Spain*, 2010, 91–96.

[11] Adam Obtulowicz, On P Systems with Active Membranes Solving the Integer Factorization Problem in a Polynomial Time, Multiset Processing, Lecture Notes in Computer Science **2235**, 2001, 267–285.

[12] Gheorghe Paun, Membrane Computing, An Introduction, *Springer-Verlag*, *Berlin*, 2002.

[13] Peter Shor, Algorithms for quantum computation: discrete logarithms and factoring, 35th FOCS, Annual Symposium on Foundations of Computer Science, 1994, 124–134.

[14] Ákos Tajti and Benedek Nagy, Solving Tripartite Matching by Interval-valued Computation in Polynomial Time, CiE 2008, Fourth Conference on Computability in Europe: Logic and Theory of Algorithms, *Local Proceedings, Athens, Greece*, 2008, 435–444.

[15] Damien Woods and Thomas J. Naughton, An optical model of computation, *Theoret. Comput. Sci.* **334** (2005), 227–258.

BENEDEK NAGY
DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF INFORMATICS
UNIVERSITY OF DEBRECEN
H-4010 DEBRECEN, P.O. BOX 12
HUNGARY

E-mail: nbenedek@inf.unideb.hu
URL: http://www.inf.unideb.hu/~nbenedek

SÁNDOR VÁLYI
INSTITUTE OF MATHEMATICS AND INFORMATICS
COLLEGE OF NYÍREGYHÁZA
SÓSTÓI ÚT 31/B.
H-4410 NYÍREGYHÁZA, HUNGARY

E-mail: valyis@nyf.hu

(Received February 7, 2011; revised October 25, 2011)