Computing discrete logarithm by interval-valued paradigm

Benedek Nagy
Faculty of Informatics
University of Debrecen, Hungary
Department of Mathematics
Eastern Mediterranean University, Turkey
nbenedek@inf.unideb.hu

Sándor Vályi
Institute of Mathematics and Informatics
College of Nyíregyháza, Hungary
valyis@nyf.hu

Interval-valued computing is a relatively new computing paradigm. It uses finitely many interval segments over the unit interval in a computation as data structure. The satisfiability of Quantified Boolean formulae and other hard problems, like integer factorization, can be solved in an effective way by its massive parallelism. The discrete logarithm problem plays an important role in practice, there are cryptographical methods based on its computational hardness. In this paper we show that the discrete logarithm problem is computable by an interval-valued computing in a polynomial number of steps (within this paradigm).

1 Introduction

There are intractable problems that traditional computing devices (Turing machines, Neumann architecture computers) cannot solve efficiently. For some classes of hard problems, such as NP-complete, PSPACE etc., it is strongly believed that one cannot find any method to solve these in deterministic polynomial time. Such a hard problem is to find the discrete logarithm of a given positive integer. Several cryptographic methods are based on the assumption that the computation of discrete logarithm cannot be achieved in deterministic polynomial time [9].

There are various new theoretical computing paradigms [1] that attack these hard problems successfully at least in theory. There are various paradigms based on inspiration from Biology (e.g., DNA-computing, membrane computing), from Physics (e.g., Quantum computing) and from other phenomena of the Nature. The efficiency of most of these new paradigms come from a massive parallelism built in the system (the power of quantum computation also derives from entanglement). In this paper we dealt with another new paradigm that uses strong inner parallelism. The Interval-valued computing paradigm uses finitely many interval segments over the unit interval in a computation. Logical operations are straightforward generalizations of the classical bit operations (of usual computers), 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. The paradigm has been investigated in [3, 4] as a way of visual computations and proved to be very efficient, e.g., PSPACE is characterized by restricted polynomial interval-valued computations in [5]. Computationally hard problems are solved in efficient way in this paradigm, e.g., the PSPACE complete problem of satisfiability of Quantified Boolean formulae in [5], integer prime factorization in [6].

In this paper we solve another computationally hard problem, namely the discrete logarithm problem, that plays an important role both in mathematical theory and in practice, e.g., it is related to some cryptographical algorithms [2, 9]. We note here that other new computing paradigms also address this problem, see, e.g., [8], where a Quantum algorithm is presented to solve the discrete logarithm problem efficiently.

B. Löwe, G. Winskel (Eds.):
Developments in Computational Models 2012 (DCM 2012)
EPTCS 143, 2014, pp. 76–86, doi:10.4204/EPTCS.143.7

The structure of the paper is as follows. In the next section we recall the interval-valued paradigm in a formal way. In Section 3 our algorithm is presented that solves the discrete logarithm problem in a cubic complexity, finally in Section 4 some concluding remarks close the paper.

2 Preliminaries

For the sake of a self-contained paper, we repeat the needed definitions from [5] and [6]. 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.

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*

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

Similarly, let $\mathbb{V}_n$ *be the set of interval-values that can be represented by (1) using only values m with the condition* $m \leq n$.

We note that the set of finite unions includes the empty set ($k = 0$), that is, $\varnothing$ 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 definitely applicable. The algebra $(\mathbb{V}, \bar{}, \cup, \cap)$ 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* $A \in \mathbb{V}$, $A \neq \varnothing$, *is defined as the interval value* $[t,s)$ *where* t *and* $s \in [0,1]$ *satisfy that* $[0,t) \cap A = \varnothing$, $[t,s) \subset A$ *and* $\forall s' > s : [t,s') \not\subset A$. *Now the function* $\mathrm{Flength} : \mathbb{V} \to \mathbb{R}$ *is defined as follows. If* $A = \varnothing$, *then* $\mathrm{Flength}(A) = 0$. *Otherwise* $\mathrm{Flength}(A) = s - t$, *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. The function 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 $[0,1)$ to the negative direction. 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*

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

and

$$\mathrm{Rshift}(A,B)(x) = A(\mathrm{frac}(x-\mathrm{Flength}(B))),$$

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

By the combined application of the shift operators we can choose any 'important' part of the interval-values by erasing its complement.

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* $A\left(\frac{x-x_B}{x^B-x_B}\right) = 1$*, 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 l interval components with end points $a_{i,1}, a_{i,2}$ $(1 \le i \le l)$ and B contains k components with end points $b_{i,1}, b_{i,2}$ $(1 \le i \le k)$, then we determine the value of $C = B*A$ as follows: we set the number of components of C to be $l \cdot k$. For this process we can use double indices for the components of C. The lower and higher end-points of the (i,j)th component are $b_{i1} + a_{j1}(b_{i2} - b_{i1})$ and $b_{i1} + a_{j2}(b_{i2} - b_{i1})$, respectively. In visual way, the interval-value A is zoomed to the components of the interval-value B. The idea and the role of this operation is similar to the unlimited shrinking of 2-dimensional images in optical computing. 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 ith 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*

$$\mathrm{op} \in \{\mathrm{AND}, \mathrm{OR}, \mathrm{LSHIFT}, \mathrm{RSHIFT}, \mathrm{PRODUCT}\}$$

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

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

Here the system of [5] is extended with an instruction to write (i.e., generate) the output as we detail below based on [6].

2.4 Computing a discrete function by interval-values

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 concatenated to the output sequence if S_j is nonempty, otherwise 0 is concatenated 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 $\mathscr{B}$ that for each possible input ($w \in \{0,1\}^*$) constructs a computation sequence that generates the output sequence $f(w)$.

The *size of a computation* is measured by the length of the computation sequence. We recall from [5] that the class of polynomial size interval-valued computations in which one of the arguments of every product operation is FIRSTHALF characterizes the classical complexity class PSPACE.

3 Computing the discrete logarithm by interval-values

In this section we solve the discrete logarithm problem within the interval-valued paradigm. Let the input $a, b, p \in \mathbb{Z}$. We give an interval-valued computation sequence that give the result as output: an exponent x of input integer a such that $a^x = b \bmod p$ holds. We can assume without loss of generality that a, b and x are non-negative integers less than p.

Theorem 8. *Discrete logarithm can be computed by an interval-valued computation of size* $O(n^3)$.

We prove this theorem in a constructive way through several Lemmas in this section.

The computation of discrete logarithm usually means the following computing task: For any input triplet(a,b,p), where p is a prime, a and b are non-negative integers less than p, find a non-negative integer x such that $a^x = b \bmod p$ holds. There is a value x such that $x < p$, therefore our search will check only integers that can be represented at most as many bits as p can be. Throughout in this paper we denote the upper integer part of the usual logarithm of p by n. That is, a, b, x and p all can be written by n binary digits. Let $a_1 \dots a_n$ be the binary representation of the input integer a. (One can assume that $n \geq 3$.) Similarly, $b_1 \dots b_n$ is the binary form of b and $p_1 \dots p_n$ is of p. We give a logspace algorithm $\mathscr{B}$ that constructs an interval-valued computation sequence S from a, b and p with an output bit sequence $d_1 \dots d_n$ that is the binary representation of the target x.

The algorithm $\mathscr{B}$ starts its work by representing the input bit sequences (a, b and p) by interval-values. First, fix S_1 as FIRSTHALF and S_2 as $(\text{RIGHT}, 1, 1)$. Then, for each $i \in \{1, \dots, n\}$, if $a_i =$

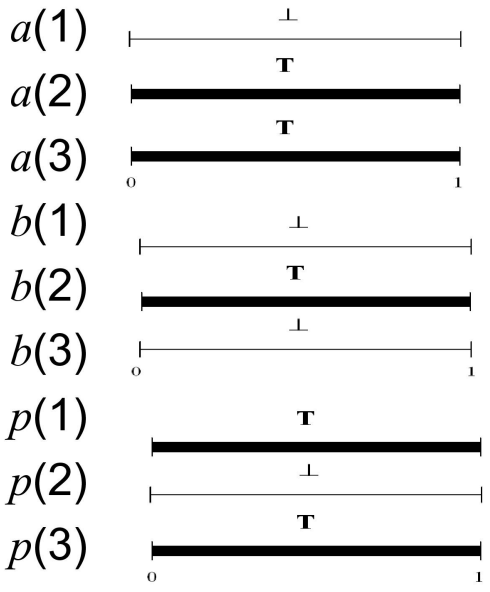

Figure 1: Representation of the input by interval-values.

1, then put $S_{2+i} := (\mathrm{OR}, 1, 2)$ else $S_{2+i} := (\mathrm{AND}, 1, 2)$; if $b_i = 1$, then put $S_{2+n+i} := (\mathrm{OR}, 1, 2)$ else $S_{2+n+i} := (\mathrm{AND}, 1, 2)$; and if $p_i = 1$, then put $S_{2+2n+i} := (\mathrm{OR}, 1, 2)$ else $S_{2+2n+i} := (\mathrm{AND}, 1, 2)$. We denote the indices of the subsequence $S_3, S_4, \ldots, S_{2+n}$ by $a(1), a(2), \ldots, a(n)$. Indices $b(1), b(2), \ldots, b(n)$ and $p(1), p(2), \ldots, p(n)$ can be defined similarly. In this way we have represented the input:

Lemma 9. *For each $k \in \{1, \ldots, n\}$:*

$$\|S_{\to a(k)}\| = \begin{cases} [0,1) & \text{if } a_k = 1 \text{ and} \\ \varnothing & \text{if } a_k = 0; \end{cases}$$
$$\|S_{\to b(k)}\| = \begin{cases} [0,1) & \text{if } b_k = 1 \text{ and} \\ \varnothing & \text{if } b_k = 0; \end{cases}$$
$$\|S_{\to p(k)}\| = \begin{cases} [0,1) & \text{if } p_k = 1 \text{ and} \\ \varnothing & \text{if } p_k = 0. \end{cases}$$

Proof. This is straightforward. □

We illustrate our algorithm by an example: $a = 3$, $b = 2$ and $p = 5$. In Figure 1 an illustration is given for the initialization part of the algorithm.

All possible candidates for x are going to be represented in a parallel way in different slices of the interval values. $\mathscr{B}$ continues its job by computing $S_{p(n)+1}, \ldots, S_{p(n)+(3n-2)}$ as follows: $S_{p(n)+1} = (\mathrm{AND}, 1, 1)$. For all positive integers $k < n$,

$$S_{p(n)+3k-1} = (\mathrm{PRODUCT}, p(n)+3k-2, 1),$$
$$S_{p(n)+3k} = (\mathrm{RSHIFT}, p(n)+3k-2, p(n)+3k-1) \text{ and}$$
$$S_{p(n)+3k+1} = (\mathrm{OR}, p(n)+3k, p(n)+3k-1).$$

Figure 2: Representation of the possible solutions.

The index sequence $p(n)+1, p(n)+4, \ldots, p(n)+(3n-2)$ will be denoted by $x(1), x(2), \ldots, x(n)$. By induction on k one can establish the following statement.

Lemma 10. *For all integer $k \in \{1,\ldots,n\}$:*

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

Similar representations were used, for instance, in [5] for a concise representation of all the possible evaluations of a Boolean formula. In this way all variations of n independent bits can be represented simultaneously by the interval-values $\|S_{\to x(1)}\|, \|S_{\to x(2)}\|, \ldots, \|S_{\to x(n)}\|$ in the following sense:

Lemma 11. *For each bit sequence $t_1 \ldots t_n$ there exists $r \in [0,1)$ that for any $k \in \{1,\ldots,n\}$: $r \in \|S_{\to x(k)}\|$ if and only if $t_k = 1$.*

The choice $r = \sum_{i=1}^{2n} \frac{1-t_i}{2^i}$ proves the lemma. In our example the largest number is 5 and it can be represented on 3 bits, therefore the case $n = 3$ is visualized in Figure 2. Now a further definition is needed to find the coded (represented) values.

Definition 12. *For $k \in \{1,\ldots,n\}$ and $r \in [0,1)$, let $x_k(r) := (r \in \|S_{\to x(k)}\|)$. Further, let the bit sequence $x_1(r)\ldots x_n(r)$ be denoted by $X(r)$. For any bit sequence $BS = b_1 \ldots b_n$, let #BS denote the integer whose binary representation is BS.*

Let us construct a Boolean circuit of size $m+n$ ($m > 0$) that multiplies two n-length input bit sequences (interpreted as integers in binary form) modulo a third n-length input bit sequence outputting the ith output bit in step $m+i$ ($i \in \{1,\ldots,n\}$). The circuit can be chosen so that circuit size m will depend on n quadratically.

It can be simulated by an interval-valued computation sequence using only the corresponding Boolean operators. Let $e(i,j)$ abbreviate $x(n)+(i-1)\cdot n + i\cdot m + j$, for any $(i,j) \in \{1,\ldots,n\}^2$. Applying the chosen multiplier computation sequence n times to the appropriate operands, $\mathscr{B}$ can construct an interval-valued computation sequence that satisfies the formula

$$\|S_{\to e(i,j)}\| = \begin{cases} [0,1), & \text{if the } j\text{th bit of } (a^{2^i} \bmod p) \text{ is } 1, \\ \varnothing, & \text{otherwise;} \end{cases}$$

for any $(i,j) \in \{1,\ldots,n\}^2$.

The length of the actual part of the constructed computation sequence is in $O(n^3)$. Figure 3 shows the interval-values $\{e(i,j) | 1 \le i,j \le 3\}$ in our example.

The algorithm $\mathscr{B}$ continues to build the computation sequence. This part is of length $3n^2$ and ensures the following: For any positive integer $i \le n$ and $j < n$,

$$\|S_{\to e(n,n)+3(i-1)n+3j}\| = \|S_{\to e(i,j)}\| \cap \|S_{\to x(i)}\|$$

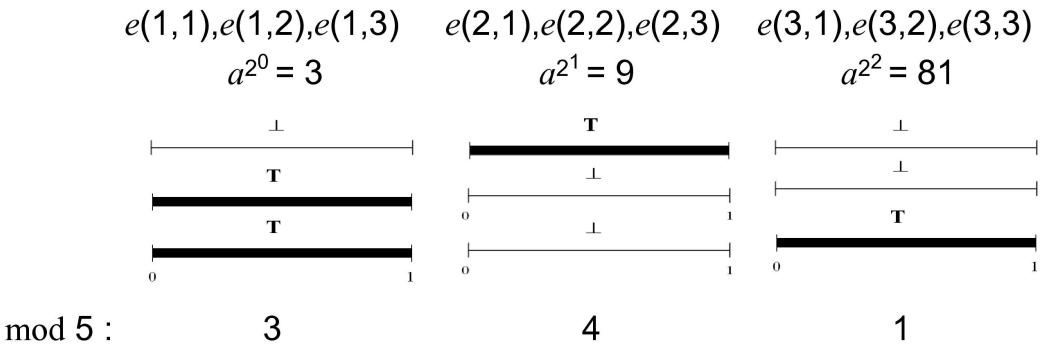

Figure 3: The values $e(i,j)$ with $a=3$, $b=2$ and $p=5$.

and

$$\|S_{\to e(n,n)+3in}\| = (\|S_{\to e(i,n)}\| \cap \|S_{\to x(i)}\|) \cup \overline{\|S_{\to x(i)}\|}.$$

With the notation $c(i,j) = e(n,n)+3(i-1)n+3j$, the last two properties lead to the following statement.

Lemma 13. *If $r \in \|S_{\to x(i)}\|$ then ($r \in \|S_{\to c(i,j)}\| \Leftrightarrow$ the jth bit of $(a^{2^i} \bmod p)$ is 1), otherwise $r \in \|S_{\to c(i,j)}\| \Leftrightarrow j = n$.*

By reusing the circuit for multiplication, $\mathscr{B}$ continues the computation in such a way that the following requirement fulfills, with the notation $f(i,j) = e(n,n)+(i-1)n+im+j \quad (1 \le i,j \le n)$,

$$r \in \|S_{\to f(i,j)}\| \Leftrightarrow \text{ the } j\text{th bit of } \left(\prod_{k=1}^{i} [x_k(r)(a^{2^k} \bmod p)] \bmod p \right).$$

The interval-values $c(i,j)$ and $f(i,j)$ of our example are shown in Figure 4. The next lemma is a direct corollary of Lemma 13.

Lemma 14. *For any possible inputs a, p and for any $r \in [0,1)$, $\#(r \in \|S_{\to f(n,1)}\|, \ldots, r \in \|S_{\to f(n,n)}\|)$ is $a^{\#X(r)} \bmod p$.*

From this point $\mathscr{B}$ continues with the equality test and output generation. Equality test means pointwise checking for $\forall k \in \{1,\ldots,n\}: \; r \in \|S_{\to b(k)}\| \Leftrightarrow r \in \|S_{\to f(n,k)}\|$. $f := f(n,n)$. The following computation will do it. For any $k \in \{1,\ldots,n\}$, let

$$\begin{aligned}
S_{\to f+5k-4} &= (\text{AND}, b(k), f(n,k)),\\
S_{\to f+5k-3} &= (\text{NOT}, b(k)),\\
S_{\to f+5k-2} &= (\text{NOT}, f(n,k)),\\
S_{\to f+5k-1} &= (\text{AND}, f+5k-3, f+5k-2),\\
S_{\to f+5k} &= (\text{OR}, f+5k-4, f+5k-1), \text{ and}\\
S_{\to f+5n+1} &= (\text{AND}, f+5, f+5);
\end{aligned}$$

for any $k \in \{2,\ldots,n\}$, let

$$S_{\to f+5n+k} = (\text{AND}, f+5k, f+5(k-1)).$$

Now, let e denote $f+6n$.

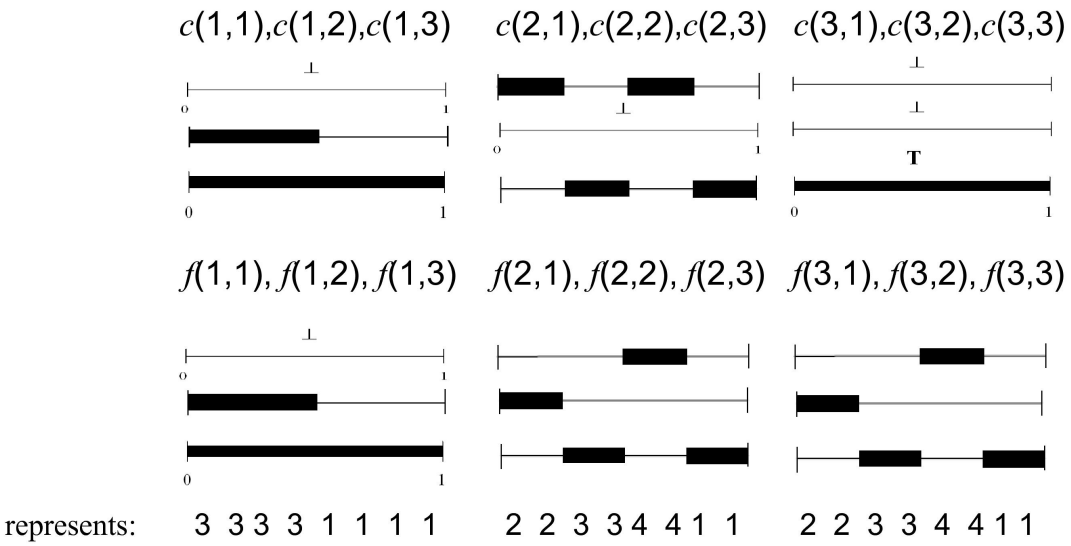

Figure 4: The values $c(i,j)$ and $f(i,j)$ with $a=3$, $b=2$ and $p=5$.

Lemma 15. *For any possible inputs a,p,b and for any $r\in[0,1)$, $r\in\|S_{\to e}\|\Leftrightarrow b=a^{\#X(r)} \bmod p$.*

Proof. We observe that

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

For $k=n$, the last statement means that $\forall r\in[0,1)$: $r\in\|S_{\to e}\|\Leftrightarrow\forall j\in\{1,\ldots,n\}:(r\in\|S_{\to b(j)}\|\Leftrightarrow r\in\|S_{\to f(n,j)}\|)$, that is, $b=a^{\#X(r)} \bmod p$ from Lemma 14 and the fact, that $b=\#(r\in\|S_{\to b(1)}\|,\ldots,r\in\|S_{\to b(n)}\|)$, independently from r. □

Now the proof of the main theorem is continued with separation of an $\frac{1}{2^n}$-size subinterval of $\|S_{\to e}\|$ that describes a solution. More definitely, we find a subinterval S of $\|S_{\to e}\|$ that $\forall r,t\in S:\#X(r)=\#X(t)$ holds. It is an important step because $\|S_{\to e}\|$ may describe more (at most two) solutions.

First a 7-step process is used to separate the first component of $\|S_{\to e}\|$.

$$\begin{aligned}
&e+1:\ (\mathrm{NOT},e),\\
&e+2:\ (\mathrm{LSHIFT},e,e+1),\\
&e+3:\ (\mathrm{LSHIFT},e+2,e+2),\\
&e+4:\ (\mathrm{RSHIFT},e+3,e+2),\\
&e+5:\ (\mathrm{RSHIFT},e+4,e+1),\\
&e+6:\ (\mathrm{NOT},e+5),\\
&e+7:\ (\mathrm{AND}e,e+6).
\end{aligned}$$

This computation guarantees that $\|S_{\to e+7}\|$ is the first component of $\|S_{\to e}\|$. Based on the fact that the solution $x=0$ implies another solution $x<p$ that is positive integer, we use right-shift to obtain an empty

subinterval $\left[0, \frac{1}{2^n}\right)$.

$$\begin{aligned} e+8: &\ (\text{RSHIFT}, e+7, x(n)),\\ e+9: &\ (\text{LSHIFT}, e+8, x(n)),\\ e+10: &\ (\text{RSHIFT}, e+9, x(n)). \end{aligned}$$

Then

$$\begin{aligned} e+11: &\ (\text{NOT}, e+10),\\ e+12: &\ (\text{LSHIFT}, e+10, e+11),\\ e+13: &\ (\text{AND}, e+12, x(n)),\\ e+14: &\ (\text{NOT}, e+13),\\ e+15: &\ (\text{LSHIFT}, e+13, e+14),\\ e+16: &\ (\text{LSHIFT}, e+15, e+15),\\ e+17: &\ (\text{RSHIFT}, e+16, e+15),\\ e+18: &\ (\text{RSHIFT}, e+17, e+14),\\ e+19: &\ (\text{NOT}, e+18),\\ e+20: &\ (\text{AND}, e+13, e+19),\\ e+21: &\ (\text{RSHIFT}, e+20, e+11). \end{aligned}$$

Finally, it is shifted back to the correct place

$$e+22: \ (\text{LSHIFT}, e+21, x(n)).$$

Lemma 16. *For all $r,t \in S_{e+22}$, we have $\#X(r) = \#X(t)$.*

Proof. There are two possible cases. The first component of $\|S_{\to e}\|$ is just an $\frac{1}{2^n}$-sized subinterval or longer. In both cases $\|S_{\to e+12}\|$ is a left-shifted version of $\|S_{\to e+7}\|$. Even if it is longer than $\frac{1}{2^n}$, the first component of $\|S_{\to e+13}\|$ has exactly length $\frac{1}{2^n}$. $\|S_{\to e+21}\|$ is computed by the same way from $\|S_{\to e+13}\|$ as $\|S_{\to e+7}\|$ from $\|S_{\to e}\|$, so $\|S_{\to e+21}\|$ is the first component of $\|S_{\to e+13}\|$. In $\|S_{\to e+22}\|$, this component is shifted back to its original place. So $\|S_{\to e+22}\|$ is the left $\frac{1}{2^n}$-length prefix of the first component of $\|S_{\to e}\|$. □

Let z denote $e+22$. Then $\mathscr{B}$ continues the computation sequence in the following way. Let S_{z+k} be $(\text{AND}, z, x(k))$ for all $k \in \{1,\ldots,n\}$ and let S_{z+n+k} be $(\text{OUTPUT}, z+k)$. By the above results, it is clear that for any possible input a, b, p, $\mathscr{B}$ will put out the bits of a solution x of $a^x = b \bmod p$. That is, computing the discrete logarithm is finished. Analyzing the length of the computation validates that it is really in $O(n^3)$. In this way our main result, Theorem 1 is proved.

Let us continue our example. In Figure 5 some details of the final part of the computation is shown. The result 111 is written to the output representing the number 7. One may easily check that $3^7 = 2187$ and $2187 \bmod 5 = 2$, the example computation is correct. With a small modification of the algorithm we may put the other value to the output (if more than one solutions can be represented on n bits).

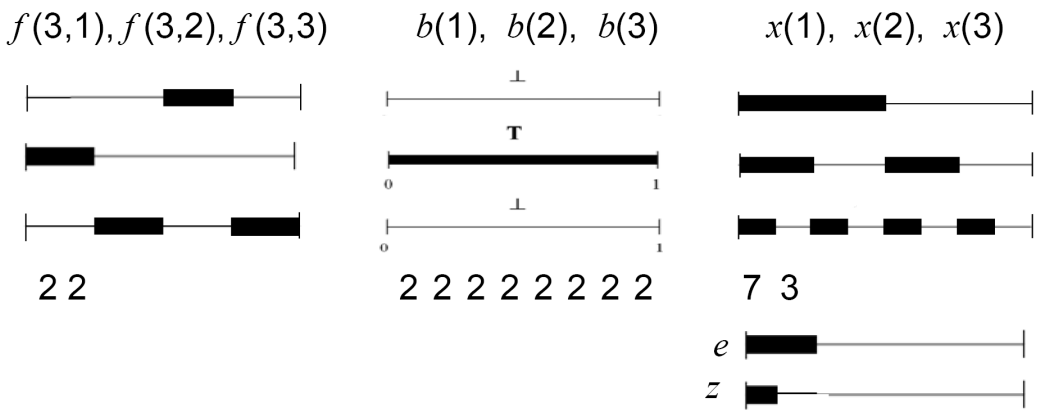

Figure 5: The final part of the computation on input $a = 3$, $b = 2$ and $p = 5$.

4 Concluding remarks

In this paper, the efficiency of the interval-valued paradigm is presented by solving the discrete logarithm problem by a cubic algorithm. Unfortunately our algorithm uses high inner parallelism (high number of small interval segments in the unit interval) and therefore it does not help to solve the problem in traditional computers.

Based on some similarities of our new result and the result presented in [6]), we could formulate a conjecture: any function computation problem that $\{(x,y)|f(x) = y\}$ is checkable on a Boolean circuit of polynomial size can be solved by a interval-valued computation of polynomial size, where the computation has a special form: the product operator is used only at the beginning, where 'all possible inputs' are generated, and later product is not used. Moreover, it seems that the reverse direction of the conjecture also holds.

As we already mentioned a class of restricted polynomial size interval-valued computations characterizes PSPACE. It is an interesting challenge to analyse the power of non-restricted case and the relations of various classes of interval-valued computations to other paradigms, e.g., to vector machines [7].

Acknowledgements

The work is supported by the TÁMOP 4.2.1/B-09/1/KONV-2010-0007 and by the TÁMOP 4.2.2/C-11/1/KONV-2012-0001 projects. The projects are implemented through the New Hungary Development Plan, co-financed by the European Social Fund and the European Regional Development Fund.

References

[1] C.S. Calude & Gh. Păun (2001): *Computing with cells and atoms: an introduction to quantum, DNA and membrane computing*. Taylor and Francis Publishers, London.

[2] R. Crandall & C. B. Pomerance (2005): *Prime numbers: a computational perspective*. Springer Verlag.

[3] B. Nagy (2005): *An interval-valued computing device*. In S.B. Cooper, B. Löwe & L. Torenvliet, editors: *CiE 2005: New Computational Paradigms, ILLC Publications* X-2005-01, Amsterdam, pp. 166–177.

[4] B. Nagy & S. Vályi (2007): *Visual reasoning by generalized interval-values and interval temporal logic*. In Philip T. Cox, Andrew Fish & John Howse, editors: *Proceedings of the VLL 2007 workshop on Visual Languages and Logic in Coeur d'Aléne, Idaho, USA, 23rd September 2007 as part of the 2007 IEEE Symposium on Visual Languages and Human Centric Computing VL/HCC 07. CEUR-WS.org 2007 CEUR Workshop Proceedings*, pp. 13–26.

[5] B. Nagy & S. Vályi (2008): *Interval-valued computations and their connection with PSPACE*. *Theoretical Computer Science* 394(3), pp. 208–222, doi:10.1016/j.tcs.2007.12.013.

[6] B. Nagy & S. Vályi (2011): *Prime factorization by interval-valued computing*. *Publicationes Mathematicae Debrecen* 79(3–4), pp. 539–551, doi:10.5486/PMD.2011.5134.

[7] V.R. Pratt & L.J. Stockmeyer (1976): *A characterization of the power of vector machines*. *Journal of Computer and System Sciences* 12(2), pp. 198–221, doi:10.1016/S0022-0000(76)80037-2.

[8] P.W. Shor (1997): *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*. *SIAM Journal on Computing* 26(5), pp. 1484–1509, doi:10.1137/S0097539795293172.

[9] D.R. Stinson (2006): *Cryptography: theory and practice*, 3rd edition. Discrete Mathematics and its Applications (Boca Raton), Chapman & Hall/CRC.