7th International Conference on Applied Informatics
Eger, Hungary, 2007.

Interval-valued computations without the product operator

Benedek Nagy[a], Sándor Vályi[b]

[a] Faculty of Informatics, University of Debrecen
e-mail: nbenedek@inf.unideb.hu

[b] Faculty of Health College, University of Debrecen, Nyíregyháza
e-mail:valyis@de-efk.hu

Abstract

In [1] B. Nagy introduced a new model for analog computations, namely the interval-valued computations, where computation is executed on so-called interval-valued bytes, which are special subsets of interval [0,1) rather than a finite sequence of bits. The allowed set of computational operators on these values were motivated by the operators usually applied to finite sequences of bits, namely, Boolean operators and shifts, furthermore, a rather specific kind of "magnification" operator, named there fractalian product.

In [4] S. Vályi and B. Nagy solved a $PSPACE$-complete problem by a linear interval-valued computation. This solution depends on the possibility of construction of interval-values with arbitrarily small components and this step needs heavy application of products. In this article we show that omitting this operator still results in a computational device with a high computation power. Namely, we will demonstrate this by establishing that the finite variable satisfaction problem of quantified propositional formulae is still decidable by a fast (quadratic) interval-valued computation without any application of the product operator.

Keywords: Continuous space machine, Analog computation, Computing with interval-values, Massively parallel computing, New computational paradigms

MSC: 68Q10, 68Q15

1. Introduction

A new discrete time / continuous space computational model was proposed in [1], the so-called interval-valued computing. It involves another type of idealization than Turing machines – the density of the memory can be raised unlimitedly instead

of its length. This new paradigm keeps some of the features of traditional Neumann-Turing type computations.

It works on specific subsets of the interval $[0, 1)$, more specifically, on finite unions of [)-type subintervals. In a nutshell, interval-valued computations start with a given interval-value $\left(\left[0, \frac{1}{2}\right) \right)$ and continue with a finite sequence of operator applications. Work on interval-values happens sequentially in a deterministic manner.

The allowed operations are motivated by the operations of the traditional computers: Boolean operations and shift operations. There is only an extra operator, the product. The role of the introduced product is to connect interval-values on different 'resolution levels'. Essentially, it shrinks interval-values.

So, in interval-valued computing systems, an important restriction is eliminated, i.e. there is no limit on the number of bits of a cell in the system; we have to suppose only that we always have a finite number of bits. Of course, in the case of a given computation an upper bound (the bit height of the computation sequence) always exists, and it gives the maximum number of bits the system needs for that computation process. Hence our model still fits into the framework of the Church-Turing paradigm, but it faces different limitations than the classical Turing model.

Although the computation in this model is sequential, the inner parallelism is extended. One can consider the system without restriction on the size of the information coded in an information unit (interval-value). It allows to increase the size of the alphabet unlimitedly in a computation. A $PSPACE$-complete problem is solved in [4] by a linear interval-valued computation. The solution depends on the possibility of construction of interval-values with arbitrarily small components and this step needs the application of the product operator. In this article we show that omitting this operator still results in a computational device with a high computation power. Namely, we will demonstrate this by establishing that the finite variable satisfaction problem of quantified propositional formulae is still decidable by a fast (quadratic) interval-valued computation without any application of the product operator.

2. The interval-valued computation system

2.1. Interval-valued computations

In this section we formalize the interval-valued computing system given in [1], following [4]. First we define what an interval-value means. Then we present the allowed operations which can be used to build and evaluate computation sequences. Finally, we give the notions concerning complexity and decidability.

2.2. 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

use always the more convenient notation.

Definition 2.1. The set $\mathbf{V}$ of *interval-values* coincides with the set of finite unions of [)-type subintervals of $[0,1)$.

Definition 2.2. The set $\mathbf{V}_0$ of *specific interval-values* coincides with $\left\{ \bigcup_{i=1}^{k} \left[\frac{l_i}{2^m}, \frac{1+l_i}{2^m}\right) \ : \ m \in \mathbf{N}, k \leqslant 2^m, 0 \leqslant l_1 < \ldots < l_k < 2^m \right\}$.

2.3. Operators on interval-values

Similarly to traditional computers working on bytes, of course, we allow bitwise Boolean operations. If we consider interval-values as subsets of [0,1) then the corresponding operations coincide with the set-theoretical operations of complementation ($\overline{A}$), union ($A \cup B$) and intersection ($A \cap B$). $\mathbf{V}$ forms an infinite Boolean set algebra with these operations. $\mathbf{V}_0$ is an infinite subalgebra of the last algebra.

Before we add some other operators, we introduce a function assisting the formulation of the following definition. Intuitively, it provides the length of the left-most component (included maximal subinterval) of an interval-value A.

Definition 2.3. We define the function $Flength : \mathbf{V} \to \mathbf{R}$ as follows. If there exist $a, b \in [0,1]$ satisfying $[a,b) \subseteq A$, $[0,a) \cap A = \emptyset$ and $[a,b') \not\subseteq A$ for all $b' \in (b,1]$, then $Flength(A) = b - a$, otherwise $Flength(A) = 0$.

Flength helps us to define the binary shift operators on $\mathbf{V}$. 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 original, "characteristic function" notation.

Definition 2.4. The binary operators $Lshift$ and $Rshift$ on $\mathbf{V}$ are defined in the following way. If $x \in [0,1]$ and $A, B \in \mathbf{V}$ then

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

$$Rshift(A,B)(x) = \begin{cases} A(frac(x - Flength(B))), & \text{if } x < 1, \\ 0 & \text{if } x = 1. \end{cases}$$

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.

Now we explain the so-called *fractalian product* on intervals.

Definition 2.5. Let A and B be general interval-values and $x \in [0,1)$. Then the fractalian product $A * B$ 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.

The idea and the role of this operation is similar to that of unlimited shrinking of 2-dimensional images in [6]. It will be used to connect interval-values of different resolution.

2.4. Syntax and semantics of computation sequences

As usual, the length of a sequence S is denoted by $|S|$ and its i-th element by S_i. If $j \leqslant |S|$ then the j-length prefix of S is denoted by $S_{\to j}$.

Definition 2.6. 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) where $\{l, m\} \subseteq \{1, \ldots, i-1\}$. The *bit height* of a computation is the number of the applied $PRODUCT$ operators in it.

Let n denote a positive integer in the rest of this paper.

Definition 2.7. The notion of n-finitized interval-valued computation sequence is the modification of the above definition starting not from $FIRSTHALF$ but from another constant $FIRST_n$ and it does not use the operator $PRODUCT$.

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

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

One can notice, that in this formulation of interval-valued computations, only *specific* interval-values (cf. Definition 2.2) appear as values of computation sequences. We can also notice that an n-finitized computation sequence without product operators can have only values from the set $\mathbf{V}_0^n$ of n-finitized specific interval-values, that is, from $\left\{ \bigcup_{i=1}^{k} \left[\frac{l_i}{2^n}, \frac{1+l_i}{2^n}\right) \ : k \leqslant 2^n, 0 \leqslant l_1 < \ldots < l_k < 2^n \right\}$.

2.5. Complexity

In this subsection, we give the definitions concerning interval-valued computability and complexity.

Definition 2.9. Let Σ be a finite alphabet and let $L \subseteq \Sigma^*$ be a language. We say that L is *decidable by an interval-valued computation* if there is an algorithm A that for each input word $w \in \Sigma^*$ constructs an appropriate computation sequence $A(w)$ such that $w \in L$ if and only if $\|A(w)\|$ is nonempty. Furthermore, we consider $\overline{L}$ also decidable in this case.

This last remark makes it possible to test emptiness and, by applying set-theoretical operators, also to test whether $\|A(w)\| = [0, 1)$. In [1], SAT was solved by a linear interval-valued computation. In [4], a $PSPACE$-complete problem, $QSAT$ was solved by a linear interval-valued computation in the following meaning.

Definition 2.10. We say that a language $L \subseteq \Sigma^*$ is *decidable by a linear (quadratic) interval-valued computation* if and only if there is a positive constant c and a logarithmic space algorithm A with the following properties. For each input word $w \in \Sigma^*$, A constructs an appropriate interval-valued computation sequence $A(w)$ such that $|A(w)|$ is not greater than $c \cdot |w|$ ($c \cdot |w|^2$) and $w \in L$ if and only if $\|A(w)\|$ is nonempty. Again, deciding $\overline{L}$ instead of L itself is allowed.

3. More notions

We recall now the definition of (a suitable variant of) the language $QSAT$ of *true quantified propositional formulae*. It is a subset of satisfiable propositional formulae, say, built from the propositional variables $\{x_1, x_2, \ldots\}$, by the logical operators $\neg, \wedge, \vee$. We *do not explicitly put the quantifier prefix to the propositional formulae*, only the definition of the language is given this way. Variables with odd indices are meant to quantify universally while those with even indices to quantify existentially. It can be shown by renaming of variables and using fictive quantifiers that this variant is equally PSPACE-complete as the original $QSAT$ ([5]). Before we define $QSAT$, we have to make some preparations.

Definition 3.1. A *valuation* is a function with range $\{0, 1\}$ on the domain $\{x_1, \ldots, x_n\}$. If $t_1, \ldots, t_n$ are truth values then we write $(t_1, \ldots, t_n)$ for the valuation v that $v(x_1) = t_1, \ldots, v(x_n) = t_n$. For a quantifier-free formula ϕ, $[[\phi v]]$ denotes the *truth value* of ϕ by the valuation v. For any positive integer i, *the quantifier* Q_i is $\forall$ if i is odd otherwise it is $\exists$.

Definition 3.2. For any propositional formula ϕ built from $x_1, \ldots, x_n$, ϕ belongs to $QSAT$ if and only if $(\forall t_1 \in \{0,1\})(\exists t_2 \in \{0,1\}) \ldots (Q_n t_n \in \{0,1\}) : [[\phi(t_1, \ldots, t_n)]] = 1$ holds. ϕ belongs to SAT if and only if $(\exists t_1 \in \{0,1\})(\exists t_2 \in \{0,1\}) \ldots (Q_n t_n \in \{0,1\}) : [[\phi(t_1, \ldots, t_n)]] = 1$ holds.

Definition 3.3. The restricted-to-n-variables subproblem of SAT ($QSAT$) means the fragment where the occuring propositional variables in the formulae are limited to $\{x_1, \ldots, x_n\}$.

4. Result and proof

Now we formalize our result that shows the correspondence between the restricted-to-n-variables subproblem of $QSAT$ and the n-finitized computations, without the product operator. We remind that n is an arbitrarily fixed positive integer.

Theorem 4.1. *Any restricted-to-n-variables subproblem of SAT and QSAT are decidable by a quadratic n-finitized interval-valued computation sequence without any usage of the product operator.*

Proof. We give an algorithm to construct the computation sequence $K_1, \ldots, K_{2n-1}, S_{1,0}, \ldots, S_{1,2n-2}, S_{2,0}, \ldots, S_{2,2n-4}, \ldots, S_{i,0}, \ldots, S_{i,2n-2i}, \ldots, S_{n-1,0}, \ldots, S_{n-1,1}, \ldots, S_{n,0}, B_1, \ldots, B_m, C_1, \ldots, C_{8n}, D$ for any input formula ϕ that contains exactly the variables $x_1, \ldots, x_n$ and the number of its subformulae is m. This algorithm produces the needed quadratic interval-valued computation. The length of the produced sequence is in $\mathcal{O}(n^2+|\phi|)$, where $|\phi|$ is the length of ϕ. The algorithm provides the above computation sequence in such a way that its interval-value will be empty if and only if $\phi \in QSAT$. Instead of writing members of computation sequence, for example, as $K_8 = (RSHIFT, 7, 2)$ we write this definition in form $K_i = Rshift(K_7, K_2)$.

First, we define K_1 as $(FIRST_n)$, further, for $j \in \{1, \ldots, n-1\}$, $K_{2j} = Rshift(K_{2j-1}, K_{2j-1})$, $K_{2j+1} = \bigcup(K_{2j-1}, K_{2j})$.

By induction on j one can establish the following.

Lemma 4.2. *For each* $j \in \{1, \ldots, n-1\}$, $\|K_{2j}\| = \left[\frac{1}{2^{n-j+1}}, \frac{1}{2^{n-j}}\right)$, *and for each* $j \in \{0, \ldots, n-1\}$, $\|K_{2j+1}\| = \left[0, \frac{1}{2^{n-j}}\right)$.

The algorithm proceeds in the following way.

Definition 4.3. For $i \in \{1, \ldots, n\}$, let $S_{i,0}$ be K_{2i-1} and for each $j \in \{0, \ldots, n-i-1\}$ let us define $S_{i,2j+1}$ as $Rshift(S_{i,2j}, K_{2i-1+2j})$ and $S_{i,2j+2}$ as $\bigcup(S_{i,2j}, S_{i,2j+1})$.

Lemma 4.4. *For each* $i \in \{1, \ldots, n\}$ *and* $j \in \{0, \ldots, n-i\}$,
$\|S_{i,2j}\| = \bigcup_{k=0}^{2^j-1} \left[\frac{2k}{2^{n-i+1}}, \frac{2k+1}{2^{n-i+1}}\right)$,
specially, for each $i \in \{1, \ldots, n\}$, $\|S_{i,2n-2i}\| = \bigcup_{k=0}^{2^{n-i-1}} \left[\frac{2k}{2^{n-i+1}}, \frac{2k+1}{2^{n-i+1}}\right)$.

From now on, we continue by the method of the proof of linear interval-valued decidability of $QSAT$.

The n independent truth values of $x_1, \ldots, x_n$ will be represented by the interval-values $\|S_{1,2n-2}\|, \|S_{2,2n-4}\|, \ldots, \|S_{n,0}\|$. Now we establish some further properties of the subsequence $\|S_{1,2n-2}\|, \|S_{2,2n-4}\|, \ldots, \|S_{n,0}\|$.

Lemma 4.5. *For every* $r \in [0, 1)$ *and positive integer* $j \leqslant n$ *hold the following conditions.*

(1) if $r \in \|S_{j,2n-2j}\|$ *then for all* $i < j$, $r + \frac{1}{2^j} \in \|S_{i,2n-2i}\|$ *if and only if* $r \in \|S_{i,2n-2i}\|$,

(2) if $r \notin \|S_{j,2n-2j}\|$ *then for all* $i < j$, $r - \frac{1}{2^j} \in \|S_{i,2n-2i}\|$ *if and only if* $r \in \|S_{i,2n-2i}\|$,

(3) $r + \frac{1}{2^j} \in \|S_{j,2n-2j}\|$ *if and only if* $r \notin \|S_{j,2n-2j}\|$.

Let $\phi_1, \ldots, \phi_m$ be an enumeration of all the subformulae of ϕ such that any formula is preceded by its subformulae (consequently, $\phi_m = \phi$). The algorithm gives the next part of the computation sequence, $(B_1, \ldots, B_m)$, in the following way. For each $i \in \{1, \ldots, m\}$,

$$B_i = \begin{cases} B_j \wedge B_k \text{ if } \phi_i = \phi_j \wedge \phi_k, \\ B_j \vee B_k \text{ if } \phi_i = \phi_j \vee \phi_k, \\ \neg B_j \text{ if } \phi_i = \neg \phi_j, \\ S_{j,2n-2j} \text{ if } \phi_i = x_j. \end{cases}$$

By induction on j the following statement can be verified.

Lemma 4.6. *For each* $j \in \{1, \ldots, m\}$,
$\|B_j\| = \{r \in [0,1) : [[\phi_j(r \in \|S_{1,2n-2}\|, r \in \|S_{2,2n-4}\|, \ldots, r \in \|S_{n,0}\|)]] = 1\}$ *holds.*

So far, we have obtained a quadratic size computation sequence to decide the satisfiability of $\phi (= \phi_m)$ by the validity of the following equivalence.

Lemma 4.7. *ϕ is satisfiable if and only if $\|B_m\|$ is nonempty.*

Proof. This can be concluded from the fact, that for each n-tuple $(t_1, \ldots, t_n)$ of truth values there is an $r \in [0,1)$ such that $(\forall i \in \{1, \ldots, n\})$: $t_i = r \in \|S_{i,2n-2i}\|$. □

The computation sequence continues with $C_1, \ldots, C_{8n}$ so that for each integer $j < n$, the following holds. $\|C_{8(j+1)}\| =$
$((Lshift(\|C_{8j}\|, \|S_{n-j,2n-2(n-j)}\|) \cap \|S_{n-j,2n-2(n-j)}\|) \cup \|C_{8j}\|) \cup$
$((Rshift(\|C_{8j}\|, \|S_{n-j,2n-2(n-j)}\|) \cap \overline{\|S_{n-j,2n-2(n-j)}\|}) \cup \|C_{8j}\|)$, if $n-j$ is even, and $(Lshift(\|C_{8j}\|, \|S_{n-j,2n-2(n-j)}\|) \cap \|S_{n-j,2n-2(n-j)}\| \cap \|C_{8j}\|) \cup$
$(Rshift(\|C_{8j}\|, \|S_{n-j,2n-2(n-j)}\|) \cap \overline{\|S_{n-j,2n-2(n-j)}\|} \cap \|C_{8j}\|)$ in the other case. In this definition, we do not specify all the intermediate expressions between C_{8j} and $C_{8(j+1)}$, they are the subexpressions of $C_{8(j+1)}$ needed to express $C_{8(j+1)}$ from C_{8j} and $S_{n-j,2n-2(n-j)}$.

To make the next lemma more readable, we assume without any further mention, that variables $t_1, t_2, \ldots, t_n$ range over the truth values. We recall that the quantifier sequence $Q_1, Q_2, Q_3, \ldots$ is defined as $\forall, \exists, \forall, \ldots$, respectively.

Lemma 4.8. *For each* $j \in \{0, \ldots, n\}$ *and for all* $r \in [0,1)$:
$r \in \|C_{8j}\|$ *if and only if*
$Q_{n-j+1} t_{n-j+1} \ldots Q_n t_n$
$\big[\big[\phi(r \in \|S_{1,2n-2}\|, \ldots, r \in \|S_{n-j,2n-2(n-j)}\|, t_{n-j+1}, \ldots, t_n)\big]\big] = 1.$

The proof of the lemma is technical, we omit it.

Letting $j = n$, the above lemma ensures that $r \in \|C_{8n}\|$ if and only if

$Q_1 t_1 \dots Q_n t_n : [[\phi(t_1, \dots, t_n)]] = 1$ holds for any $r \in [0, 1)$. Since the right side of the last equivalence is independent from r, we can state that $Q_1 t_1 \dots Q_n t_n : [[\phi(t_1, \dots, t_n)]] = 1$ if and only if $\|C_{8n}\|$ is equal to $[0, 1)$. Finally, by setting the last element of the computation sequence, D to $\neg C_{8n}$ the algorithm constructs a computation sequence whose interval-value is empty if and only if $\phi \in QSAT$. □

Acknowledgements. The work of the first author has been supported by grants of the Hungarian National Foundation for Scientific Research OTKA F043090, T049409 and by a grant of the Hungarian Ministry of Education and by the Öveges programme of the Agency for Research Fund Management and Research Exploitation (KPI) and National Office for Research and Technology

References

[1] Nagy, B. "An Interval-valued Computing Device", in: *"Computability in Europe 2005: New Computational Paradigms"*, (eds. S. B. Cooper, B. Löwe, L. Torenvliet), ILLC Publications X-2005-01, Amsterdam, pp. 166–177.

[2] Nagy, B. and Allwein, G. Diagrams and Non-monotonicity in Puzzles. Proceedings of Diagrams'2004, Third International Conference on the Theory and Application of Diagrams, Cambridge, England, *LNCS/LNAI* 2980, Diagrammatic Representation and Inference, eds.:A. Blackwell, K. Marriott, A. Shimojima, 2004.

[3] Nagy, B., Vályi, S. "Solving a PSPACE-complete problem by a linear interval-valued computation", in: *Proceedings of Conference "Computability in Europe 2006: logical approaches to computational barriers"* (eds. Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V Tucker) University of Wales Swansea, Computer Science Report Series 2006.

[4] Nagy, B., Vályi, S. "Interval-valued computations: introduction and connection to classical complexity theory", accepted for publication in *Theoretical Computer Science*.

[5] C. H. Papadimitriou *Computational Complexity*, Addison-Wesley, 1994.

[6] D. Woods, T. Naughton, An optical model of computation, *Theoretical Computer Science* 334(2005):227-258.

Benedek Nagy
University of Debrecen, Faculty of Informatics,
Hungary, H-4032, Debrecen, Egyetem tér 1.

Sándor Vályi
University of Debrecen, Faculty of Health College,
Hungary, H-4400, Nyíregyháza, Sóstói út 2-4.