Solving a PSPACE-complete problem by a linear interval-valued computation

Benedek Nagy[1,2] and Sándor Vályi[1]

1: Department of Computer Science, Faculty of Informatics, University of Debrecen
Hungary H-4010 Debrecen, P.O. Box 12
{nbenedek|valyis}@inf.unideb.hu
2: Research Group on Mathematical Linguistics, Rovira i Virgili University
Tarragona, Spain

Abstract. We continue investigation of interval-valued computing based on [Nagy 2005b] where computation is executed on interval-valued bytes which are bits indexed by the interval [0,1] rather than by a finite set. This device was presented there as a new possible model of analogue computers. Also the question was posed in [Nagy 2005b], which complexity is needed to solve PSPACE-complete problems in this paradigm. We answer this question. Namely, we show that the problem of validity of quantified propositional formulae is decidable by a linear interval-valued computation.

Keywords: Interval-valued Logic, Computing with Interval-values, Non-standard Computing, Massively Parallel Computing

1 Introduction

More or less, all theoretical models for general computing devices involve some abstraction of the concrete possibilities of the available technical devices. For example, a Turing machine can access an unlimited number of tape units. Or, it is generally accepted to examine complexity-theoretical aspects of Turing machines with oracles of high complexity (see in [Rogers 1987]). There is an extensive literature on infinite size and infinite time computations on Turing machines (e.g. [Lenzi and Monteleone 2004], [Hamkins and Seabold 2001], [Welch 2000]). In [Blum et all. 1989] (see also in [Blum et all. 1998]), a complexity theory of calculations with real numbers is introduced. It is a possible approach to model analogue (in the sense of non-digital) computations. Another model appeared in [Nagy 2005b], where a new generalization of classical digital computers was proposed. Here the unit of information processing remains a byte which is in this case not a finite amount of bits but a full sequence of bits indexed by the interval [0,1]. The model is based on a version of interval-valued logic in [Nagy 2005a], which is a natural extension of Boolean logic to interval-values.

The motivation of this model is given in [Nagy 2005b]. The physical possibility of full implementation of these kind of computation is of course a question of

future investigations. Some limited approximations were mentioned already in [Nagy 2005b]. Our result applies for a restricted model which is closer to an implementable version by today available hardware. It may be of mathematical interest, what class of functions can be computed by this paradigm and how its complexity hierarchy relates to other hierarchies of complexity. In [Nagy 2005b] this research program was started; it was shown, that the problem SAT – which is a basic example for NP-complete problems – can be solved by a linear interval-valued computation.

In this paper we contribute to the development of complexity issues of this computing paradigm. [Nagy 2005b] posed the question whether there exists a PSPACE-complete problem solvable by an interval-valued computation of similar simple complexity. We show that the problem $QSAT$ – whether a quantified propositional formula is true, which problem is PSPACE-complete (see e.g. [Papadimitriou 1994]) – can be solved by a linear interval-valued computation, as well.

2 Interval-valued computations

In this section we recall the interval-valued computing system of [Nagy 2005b]. First we define what the interval-values mean. After this, we present the allowed operations which can be used to build expressions and also present how to give the values of this expressions. The selection of this operators are natural but we do not claim that it is the only natural possibility. It is clear that the complexity hierarchy depends on this selection. It is a possible future research to do investigations into this direction.

2.1 The allowed interval-values

In our model of computation, the computing device is similar to the processing unit of a traditional computer, but operates on interval-values instead of bytes. We give an alternative definition compared to that of [Nagy 2005b] but it is equivalent and more concise.

Any of the following objects is called a *subinterval*: $[a, b], [a, b), (a, b], (a, b), [a, a]$ where $0 \leqslant a < b \leqslant 1$. The *allowed class of interval values* (denoted by $\mathbb{V}$) coincides with the set of finite unions of subintervals.

We note that in this paper a restricted set of interval-values is used. We allow only finite unions of subintervals as values. This selection limits the computational power but put the system more close to the implementable by technical devices available today.

The set of finite unions includes the empty one, that is, $\emptyset$ is an allowed interval-value, too. $\mathbb{V}$ forms a Boolean set algebra with the usual set-theoretical operations of intersection (as conjunction), union (as disjunction) and complementation (as negation). Other logical operators (implication, nand etc.) are expressible using these basic operators in the usual way.

2.2 Non-logical operators on interval-values

Similarly to a traditional computer which has not only logical (above called set-theoretical) operations, we need to allow some other operations, two shift operators and the "fractalian" product. This makes our system a general computing device, as was shown already in [Nagy 2005b].

For preparation, we introduce the first-length function $Flength$ which will be useful in the forthcoming definitions. This function assigns a real number to each interval-value, namely the length of its first "component". More precisely if we denote loosely both the interval-value and its characteristic function by the same letters, say $A, B, \ldots$, (we will do this identification permanently in the rest of this article) then $\mathit{Flength}(A) = b - a$, if A contains the open interval (a, b) and A does not contain any interval (a, c) with $c \geqslant b$, moreover the difference of A and $[a, b]$ does not contain any point x with $x < a$. In other cases $\mathit{Flength}(A) = 0$.

By the help of $Flength$ we recall now the shift operators. The *left-shift* operator will shift the first interval-value by the first-length of the second operand to the left, and remove the part which is shifted out of the interval $[0, 1]$. Opposite 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 $[0, 1]$. If A, B are two interval-values then interval-values $Lshift(A, B)$ and $Rshift(A, B)$ are defined through their characteristic functions.

$Lshift(A, B)(x) = A(x + \mathit{Flength}(B))$ if $0 \leqslant x + \mathit{Flength}(B) \leqslant 1$, and
$Lshift(A, B)(x) = 0$ in other cases.
$Rshift(A, B)(x) = A(\mathit{frac}(x - \mathit{Flength}(B)))$ if $x < 1$, and,
$Rshift(A, B)(1) = Rshift(A, B)(0)$.

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

In Figure 1 some examples appear for both operations $Rshift$ and $Lshift$. The second (ancillary-) operands are shown with grey colour to help understanding, but they are not real parts of the resulted interval-values.

Now we explain the fractalian product on intervals. If A contains k interval components with ends $a_{i,1}, a_{i,2}$ $(1 \leqslant i \leqslant k)$ and B contains l components with ends $b_{i,1}, b_{i,2}$ $(1 \leqslant i \leqslant l)$, then we determine the value of $C = A * B$ as follows: we set the number of components of C as $k \cdot l$. For this process we can use double indexes for the components of C. The starting and ending point of the ij-th component is $a_{i1} + b_{j1}(a_{i2} - a_{i1})$ and $a_{i1} + b_{j2}(a_{i2} - a_{i1})$, respectively. An endpoint belongs to the interval if and only if the original endpoints belong to the interval-values of the original interval-values A and B.

In Figure 2 there are examples for the fractalian product of interval-values.

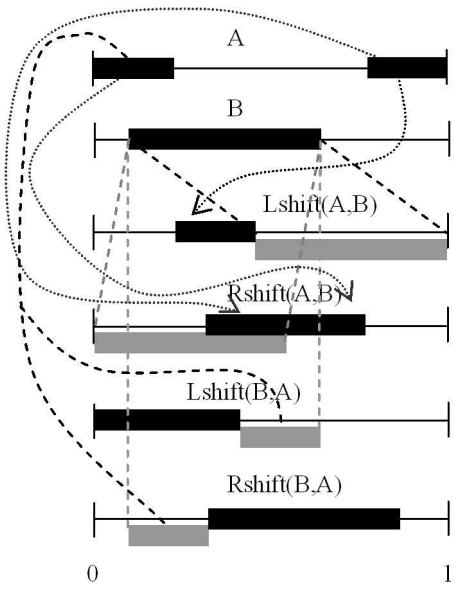

Fig. 1. Examples of shift operators with interval-values

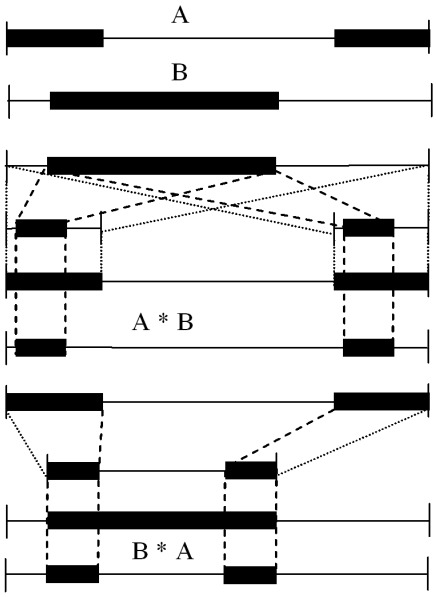

Fig. 2. Examples for product of interval-values

2.3 Computations as evaluation of expressions

Several variations can be considered based on the allowed constants and on the method checking the result. In this paper we use only one interval-valued constant and we allow to check emptiness and non-emptiness of the resulted interval-value. It is obvious that the concepts of computability and complexity depend on the used variation. Now we will describe the version used in this paper.

An *interval-valued expression* means a term composed from the constant $FIRSTHALF$ and from expression labels, by a finite number of applications of the above defined two-argument operators (the two shifts and the fractalian product) and the set-theoretical ones. No recursion, even no implicit recursion by a reference cycle, is permitted when we use expression labels. The *operational complexity* of an expression E means the number of operation symbols used while building up E from the constant and the labels. We note that it remains an interesting question of future research to determine to which class of cyclic operator definitions one can provide adequate fixpoint semantics.

We define now the *interval-value* of an expression E, denoted by $\|E\|$. We fix $\|FIRSTHALF\|$ as $[0, \frac{1}{2}]$ while the value of the label of an already evaluated expression is just the value of the expression itself. Because lack of recursion, it is well-defined, simply the evaluation of a referred expression has to precede the evaluation of its referring expression. $\|Lshift(A, B)\|$, $\|Rshift(A, B)\|$ and $\|A * B\|$ are defined in the preceding subsection alike the logical operators, i.e. interval-values of union, intersection and complementation of intervals.

2.4 Decidability

Let Σ be a finite alphabet. We say that a language $L \subset \Sigma^*$ is *decidable by an interval-valued computation* if there is an algorithm A that for each input word $w \in \Sigma^*$ constructs an appropriate expression $A(w)$ such that $w \in L$ if and only if $\|A(w)\|$ is nonempty. Further, we consider in this case $\overline{L}$ also decidable. (This last remark makes possible to test emptiness and by set-theoretical operators also to test whether $\|A(w)\| = [0, 1]$.)

We say that a language $L \subset \Sigma^*$ is *decidable by a linear interval-valued computation* if and only if there is a positive constant c and an algorithm A with the following properties. For each input word $w \in \Sigma^*$ A constructs an appropriate list L of expressions with the last element $A(w)$ such that the length L is not greater than $c \cdot |w|$, and the operational complexity of each member of L is less than c, moreover, $w \in L$ if and only if $\|A(w)\|$ is nonempty. Again, we allow to decide $\overline{L}$ instead of L itself.

Our motivation to define linear interval-valued computations in this way was to make explicit in what sense [Nagy 2005b] stated that a linear computation exists to decide SAT.

3 A linear interval-valued computation to decide *QSAT*

3.1 The language of the true quantified propositional formulae

We recall now the definition for (a suitable variation of) the language $QSAT$ of the *true quantified propositional formulae*. It is a subset of the satisfiable propositional formulae, say built from the propositional variables $\{x_1, x_2, x_3, \ldots\}$. We do not put syntactically the quantifier prefix to the propositional formulae, only the definition of semantics is given by this way. The variables of odd index are meant to quantify universally while ones of even index to quantify existentially. It can be shown by variable renaming and using of the fictive quantifiers that this variation is equally PSPACE-complete as the original $QSAT$ ([Papadimitriou 1994]).

Before we continue the definition, we have to make same preparations. A *valuation* is a function with range $\{0,1\}$ on the domain $\{x_1, \ldots, x_n\}$ for some positive integer n. If $t_1, \ldots, t_n$ are truth values then we write $(t_1, \ldots, t_n)$ for the valuation v that $v(x_1) = t_1, \ldots, v(x_n) = t_n$ and $dom(v) = \{x_1, \ldots, x_n\}$. 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 $\exists$.

So, for any formula ϕ, ϕ belongs to our variation of $QSAT$ if and only if there exists a positive integer n such that the propositional variables occurring in ϕ are exactly $x_1, \ldots, x_n$, and $(\forall t_1 \in \{0,1\})(\exists t_2 \in \{0,1\}) \ldots (Q_n t_n \in \{0,1\}) : |\phi(t_1, \ldots, t_n)| = 1$.

3.2 The result

Theorem 1. *$QSAT$ is decidable by a linear interval-valued computation.*

We choose c as 9. We give an algorithm to construct expressions $A_1, \ldots, A_n$, $B_1, \ldots, B_m, C_0, \ldots, C_n, D$ for any input formula ϕ that contains exactly the variables $x_1, \ldots, x_n$ and the number of its subformulae is m. Clearly, the length of this list is less than $4 \cdot |\phi|$. Our algorithm constructs the mentioned list in such a way that $\|D\|$ will be empty if and only if $\phi \in QSAT$, moreover the operational complexity of its each member is less than 9.

In [Nagy 2005b], a proof was given for the decidability of SAT by a linear interval-valued computation. This proof essentially gives an algorithm that for any formula ϕ (having m subformulae) built from $x_1, \ldots, x_n$, constructs an expression list $A_1, \ldots, A_n, B_1, \ldots, B_m$ of length less than $2 * |\phi|$ and the requirement on the operational complexity is also satisfied. We can realize that these expressions satisfy the following conditions:

(1) $\|A_1\| = [0, 1/2]$,

(2) generally, for every $j \in \{1, \ldots, n\}$, $\|A_j\| = \bigcup_{k=0}^{2^j-2} \left[\frac{2k}{2^j}, \frac{2k+1}{2^j}\right]$, further,

(3) $\|B_m\| = \{r \in [0,1] : |\phi(r \in \|A_1\|, \ldots, r \in \|A_n\|)| = 1\}$.

Furthermore, for any $r \in [0,1], r \neq \frac{k}{2^n} (k \in \mathbb{Z})$ hold

(4) if $r \in \|A_j\|$ then $r + \frac{1}{2^j} \in \|A_i\|$ if and only if $r \in \|A_i\|$ for all $i < j$,
(5) if $r \notin \|A_j\|$ then $r - \frac{1}{2^j} \in \|A_i\|$ if and only if $r \in \|A_i\|$ for all $i < j$,
(6) $r + \frac{1}{2^j} \in \|A_j\|$ if and only if $r \notin \|A_j\|$.

Our algorithm continues the work of the original algorithm and constructs a new expression list $C_0, C_1, \ldots, C_n, D$ to append to $A_1, \ldots, A_n, B_1, \ldots, B_m$. The older expressions are cited as labels in the new ones. The expressions $C_0, \ldots, C_n$ are determined by an inductive way. This results no cyclic recursion. C_0 is just a copy of B_m. For $j \in \{0, \ldots, n-1\}$, let C_{j+1} be

$((Lshift(C_j, A_{n-j}) \wedge A_{n-j}) \vee C_j) \vee ((Rshift(C_j, A_{n-j}) \wedge \neg A_{n-j}) \vee C_j)$, if $n - j$ is even,
$(Lshift(C_j, A_{n-j}) \wedge A_{n-j} \wedge C_j) \vee (Rshift(C_j, A_{n-j}) \wedge \neg A_{n-j} \wedge C_j)$ in the other case.

The following lemma is essential to finish our proof. To make it more readable, we assume that the variables from $\{t_1, t_2, \ldots\}$ range over the truth values without any further mention. We recall the definition of quantifier sequence $Q_1, Q_2, Q_3, \ldots$ as $\forall, \exists, \forall, \ldots$, respectively.

Lemma 1. *For all integer j between 0 and n and for all $r \in \left([0,1] \setminus \bigcup_{k=0}^{2^n} \{\frac{k}{2^n}\}\right)$:*
$r \in \|C_j\|$ if and only if
$Q_{n-j+1} t_{n-j+1} \ldots Q_n t_n \ |\phi(r \in \|A_1\|, \ldots, r \in \|A_{n-j}\|, t_{n-j+1}, \ldots, t_n)| = 1$.

Proof. It is an induction on j from 0 up to n. For $j = 0$, the condition (3) implies the needed equivalence, which is
$r \in \|C_0\|$ if and only if $|\phi(r \in \|A_1\|, \ldots, r \in \|A_n\|)| = 1$.

Assume that for any r that is in $\left([0,1] \setminus \bigcup_{k=0}^{2^n} \{\frac{k}{2^n}\}\right)$, $r \in \|C_j\|$ if and only if $Q_{n-j+1} t_{n-j+1} \ldots Q_n t_n \ |\phi(r \in \|A_1\|, \ldots, r \in \|A_{n-j}\|, t_{n-j+1}, \ldots, t_n)| = 1$. It is the induction hypotheses.

We have to see that $r \in \|C_{j+1}\|$ if and only if
$Q_{n-j} t_{n-j} \ldots Q_n t_n \ |\phi(r \in \|A_1\|, \ldots, r \in \|A_{n-(j+1)}\|, t_{n-j}, \ldots, t_n)| = 1$,
for arbitrary r from $\left([0,1] \setminus \bigcup_{k=0}^{2^n} \{\frac{k}{2^n}\}\right)$.

We write a sequence of equivalent conditions starting with $r \in C_{j+1}$ and closing with the right side of the equivalence to prove. We prove the case when $n - j$ is even and Q_{n-j} is $\exists$, the proof for the case of odd $n - j$ can be constructed analogously.

(i) $r \in \|C_{j+1}\|$,
(ii) $r \in \|C_j\|$ or $\left(r \in \|Lshift(C_j, A_{n-j})\| \wedge r \in \|A_{n-j}\|\right)$ or
$\left(r \in \|Rshift(C_j, A_{n-j})\| \wedge r \in \neg\|A_{n-j}\|\right)$,

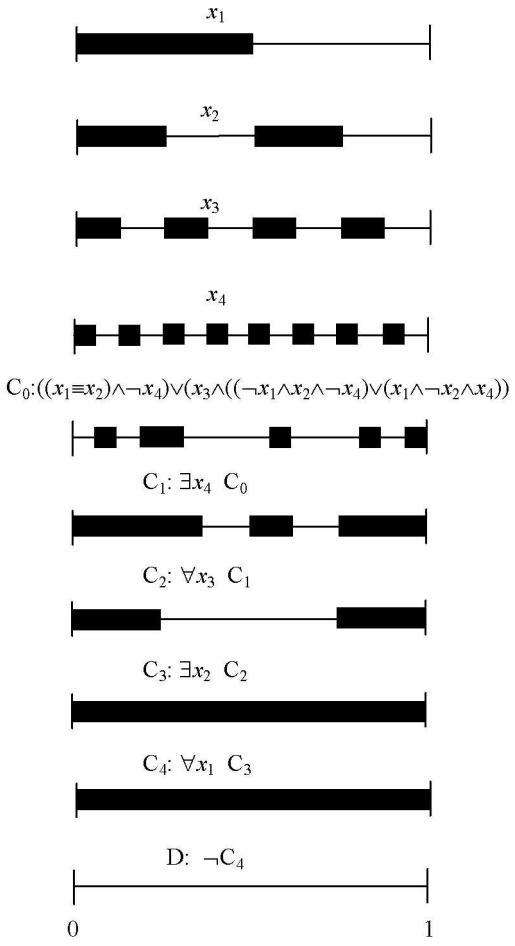

Fig. 3. $\forall x_1 \exists x_2 \forall x_3 \exists x_4 (((x_1 \equiv x_2) \wedge \neg x_4)) \vee (x_3 \wedge ((\neg x_1 \wedge x_2 \wedge \neg x_4) \vee (x_1 \wedge \neg x_2 \wedge x_4)))$ is true.

(iii) $\forall t_{n-j+1} \dots Q_n t_n \ |\phi(r \in \|A_1\|, \dots, r \in \|A_{n-j}\|, t_{n-j+1}, \dots, t_n)| = 1$ or
$r \in \|A_{n-j}\| \wedge$
$\forall t_{n-j+1} \dots Q_n t_n \ |\phi(r+\frac{1}{2^{n-j}} \in \|A_1\|, \dots, r+\frac{1}{2^{n-j}} \in \|A_{n-j}\|, t_{n-j+1}, \dots, t_n)| =$
1, or
$r \notin \|A_{n-j}\| \wedge$
$\forall t_{n-j+1} \dots Q_n t_n \ |\phi(r-\frac{1}{2^{n-j}} \in \|A_1\|, \dots, r-\frac{1}{2^{n-j}} \in \|A_{n-j}\|, t_{n-j+1}, \dots, t_n)| =$
1,

(iv) $\forall t_{n-j+1} \dots Q_n t_n \ |\phi(r \in \|A_1\|, \dots, r \in \|A_{n-j}\|, t_{n-j+1}, \dots, t_n)| = 1$ or
$\forall t_{n-j+1} \dots Q_n t_n \ |\phi(r \in \|A_1\|, \dots, r \in \|A_{n-j-1}\|, r \notin \|A_{n-j}\|, t_{n-j+1}, \dots, t_n)| =$
1,

(v) $\exists t_{n-j} \forall t_{n-j+1} \dots Q_n |\phi(r \in \|A_1\|, \dots, r \in \|A_{n-j-1}\|, t_{n-j}, \dots, t_n)| = 1$

The equivalence of (i) and (ii) is validated by the definitions of C_{j+1} and the operators, while of (ii),(iii) and (iv) by the properties (3)–(6). finally, the equivalence of (iv) and (v) can be shown by the consideration that only two possible truth values exist.
Q.E.D.

Now, we are ready for

Proof of Theorem 1. The above lemma ensures by letting $j = n$ that $r \in \|C_n\|$ if and only if $Q_1 t_1 \dots Q_n t_n \ : |B_m(t_1, \dots, t_n)| = 1$ and this holds for any r in [0,1] maybe except some of the values $\frac{k}{2^n}$ where $0 \leqslant k \leqslant 2^n$. By taking $D = \neg(Lshift(C_n, A_n * A_1) \vee C_n \vee Rshift(C_n, A_n * A_1))$
we get an expression satisfying that $\|D\|$ is empty if and only if $\phi \in QSAT$.
Q.E.D.

Figure 3 shows an example of evaluation of a formula. The formula $(((x_1 \equiv x_2) \wedge \neg x_4)) \vee (x_3 \wedge ((\neg x_1 \wedge x_2 \wedge \neg x_4) \vee (x_1 \wedge \neg x_2 \wedge x_4)))$ is demonstrated to be in $QSAT$.

We can observe that due to the special choice of the semantics of the expressions, the parallelism of our computations is finite (but unbounded). In fact, for any of our computations there is an integer n that the occurring interval values are finite unions of intervals $[\frac{k}{2^n}, \frac{k+1}{2^n}](0 \leqslant k < n)$ and their open or half open counterparts. It is obvious that these limitations restrict the computational power of the model.

4 Conclusion and final remarks

We proved the solvability of a $PSPACE$-complete problem by a linear interval-valued computation. It seems to be trivial that a constant amount of operators is not enough to decide even SAT, at least if we does not allow other operators on interval-values. To find the upper side of limitations of this paradigm one should search for conditions describing linear or general computability of problems by interval-valued computation. These conditions may differ for different choices of initial interval values and of operators. For example, is the language

of true arithmetics decidable by an interval-valued computation, of course allowing infinite parallelism? We do not see any trivial argumentation against this possibility.

Our definition for computability has a Boolean-network style in some sense. One could consider a computational framework in a more imperative manner and relates the hierarchies arising in this way.

Acknowledgements

The authors are grateful to the referee for her or his valuable remarks. This research was partly supported by the Hungarian Research Foundation for Scientific Research (OTKA), grant no. F043090, T049409 and T043242.

References

[Blum et all. 1989] Blum, L., Shub, M. and Smale, S. "On a theory of computation and complexity over the real numbers", Bull. AMS (New Series) 21(1989), no. 1, pp. 1–46.

[Blum et all. 1998] Blum, L., Cucker, F., Shub, M. and Smale, S. "Complexitiy and real computation", Springer, 1998

[Hamkins and Seabold 2001] Hamkins, J. D. and Seabold, D. E., "Infinite time Turing machines with only one tape", Math. Logic Quarterly 47(2001), no. 2, pp. 271–287.

[Lenzi and Monteleone 2004] Lenzi, G. and Monteleone, E., "On fixpoint arithmetic and infinite time Turing machines", Inform. Process. Letters 91(2004), no. 3, pp. 121–128.

[Nagy 2005a] Nagy, B., "A general fuzzy logic using intervals", 6th International Symposium of Hungarian Researchers on Computational Intelligence, Budapest, Hungary, pp. 613–624. (earlier version: Interval-valued logic, University of Debrecen, 1998, thesis, in Hungarian).

[Nagy 2005b] 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.

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

[Rogers 1987] Rogers, H., "Theory of Recursive Functions and Effective Computability", 1987, MIT Press.

[Welch 2000] Welch, P. D., "Eventually infinite time Turing machine degrees: infinite time decidable reals", J. of Symb. Logic 65(2000), no. 3, pp. 1193–1203.