Interval-valued computing as a visual reasoning system

Benedek Nagy
University of Debrecen
Faculty of Informatics and Information Sciences
nbenedek@inf.unideb.hu

Sándor Vályi
University of Debrecen
Faculty of Health College
Department of Health Informatics
valyis@de-efk.hu

Abstract

We show the visual reasoning power of a recently developed unconventional computing model – the so-called interval-valued computing. It bears not only a high evidence of power in visual representation of Boolean algebraic calculations to show validity of propositional logical laws but also a natural way of faithful visual following of the process of the solution of a typical PSPACE-complete problem, namely, determining validity of quantified propositional formulæ.

1. Introduction

Diagrammatic and visual reasoning is a basic bridge between formal reasoning and human understanding. A basic method for visualizing Boolean algebraic calculations is the method of Venn diagramms. It is applicable for formulæ built from two or three propositional variables. There are good ideas to generalize Venn diagrams to a higher number of variables (see [3],[5],[6],[7],[10],[13]). We will employ the apparatus of interval-valued computing which is a recently developed computing paradigm ([9], [11], [12]) that works with finite sets of intervals (opposite to the interval-arithmetic which works with intervals taking care about uncertainty).

Moreover, we show that the interval-valued computation process is particularly suitable to visualize the process of evaluation of quantified propositional formulæ. This process is an unconventional but visually well interpretable decision process for a PSPACE-complete problem (whether the given quantified propositional formula is true). This problem is a basic example of a class of problems where the question is to determine whether there is a winning strategy for the first player in the given game and most of these problems are reducible to $QSAT$.

In Section 2, the notion of interval-values is introduced, also the interval-valued computations are described. In Section 3, we show a visualized proof of a propositional logical law with four variables. In Section 4, we demonstrate how visualize an interval-valued computation for the validation of a quantified propositional formula.

2. The interval-valued computation system

2.1. The idea

In [9], Nagy proposed a new discrete time / continuous space computational model, 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 $\left[0, \frac{1}{2}\right)$ and continue with a finite sequence of operator applications. It works 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. In this article we employ this inner parallelism to extend the visual expressiveness of calculations with interval-values. Long manipulations on the separate bits can be shown as a unit, acting uniformly to the whole stored data.

2.2. Interval-values

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

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

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

We note that the set of finite unions includes the empty set ($k = 0$), that is, $\emptyset$ is also an allowed interval-value.
Our notion of interval-value is related to the notion of generalized interval ([4]). Our paper outlines not only a rule-based reasoning system, but an unconventional computation system whose computations represent and follow also visually the steps of the solution of the given problem more faithful than other known unconventional computing paradigms as membrane or DNA-computing. In [2] diagrams are used to solve problems on specific graphs.

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 operations negation, disjunction, conjunction coincide with the set-theoretical operations of complement ($\overline{A}$), union ($A \cup B$) and intersection ($A \cap B$), respectively. $\mathbf{V}$ forms an infinite Boolean set algebra with these operations. $\mathbf{V}_0$ is an infinite subalgebra of this 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. 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 $Flength(A) = 0$.

The binary operators $Lshift$ and $Rshift$ on $\mathbf{V}$ are defined in the following way using the characteristic function notation of interval-values. If $x \in [0,1]$ and $A, B \in \mathbf{V}$, then

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

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

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.
Let A and B be interval-values and $x \in [0,1)$. Then the fractalian product $A * B$ includes x if and only if $B(x) = 1$ and $A\left(\frac{x-\underline{B}}{\overline{B}-\underline{B}}\right) = 1$, where $\underline{B}$ denotes the lower end-point of the B-component including x and $\overline{B}$ denotes the upper end-point of this component, that is, $[\underline{B}, \overline{B})$ 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 [14]. It will be used to connect interval-values of various resolution. The fractalian product of two interval-values is the result of shrinking the first operand to each component of the second one.

2.4. Syntax and semantics of computation sequences

In this subsection, we formalize the interval-valued computations of [9]. This formalisation is of Boolean network style, since equality or similar tests do not seem to be easily implementable for interval-values, just like in the case of optical computing (no tests for equalities on images)[14]. As usual, the length of a sequence S is denoted by $|S|$ and its i-th element by S_i. If $j \leq |S|$ then the subsequence containing the j first elements of S is denoted by $S_{\to j}$.

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.

The semantics of interval-valued computation sequences can be defined by induction on the length of the sequences. The *interval-value* of such a sequence S is denoted by $\|S\|$. Let $\|(FIRSTHALF)\|$ be the interval-value $\left[0, \frac{1}{2}\right)$ and the value of longer sequences be composed by the corresponding operations on the interval-values.

2.5. Decidability

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

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 [9], SAT was solved by a linear interval-valued computation in the following meaning.

We say that a language $L \subseteq \Sigma^*$ is *decidable by a linear 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 $c \cdot (|w|)$

and $w \in L$ if and only if $\|A(w)\|$ is nonempty. Again, deciding $\overline{L}$ instead of L itself is allowed.

3. Visual expressiveness of interval-valued computations: Propositional logic

In [9] it was proved that the problem whether a propositional formula is tautology or not, is decidable by a linear interval-valued computation. We will show that the sequence of interval-values produced by this computation represents visually the information needed to follow the Boolean algebraic calculation proving or disproving that the input formula (ϕ) is a tautology ($\phi \in TAUT$). This representation is no less natural than the representation by Venn diagrams.

The solution was to give an algorithm for constructing a computation sequence $K_1, \ldots, K_{3n+m+1}$ for any input formula ϕ that contains exactly the variables $x_1, \ldots, x_n$ and the number of its subformulae is m. The algorithm provides the above computation sequence in such a way that its interval-value will be [0,1) if and only if $\phi \in TAUT$.

Let K_1 be $FIRSTHALF$. For all positive integers $k \leq n$, we define

$K_{3k-1} = (PRODUCT, 1, 3k-2)$,
$K_{3k} = (RSHIFT, 3k-1, 3k-2)$ and
$K_{3k+1} = (OR, 3k, 3k-1)$.

The following statement can be established:
For all positive integer k, if $k \leq n$ then

$$\|K_{\rightarrow 3k-2}\| = \bigcup_{l=0}^{2^{k-1}-1} \left[\frac{2l}{2^k}, \frac{2l+1}{2^k}\right).$$

The n independent truth values of $x_1, \ldots, x_n$ will be represented by the interval-values $\|K_{\rightarrow 1}\|, \|K_{\rightarrow 4}\|, \ldots, \|K_{\rightarrow 3n-2}\|$. See Figure 1, lines 1–4 for an example with $n = 4$, X_1 is $\|K_{\rightarrow 1}\|$, x_2 is $\|K_{\rightarrow 4}\|$, x_3 is $\|K_{\rightarrow 7}\|$ and x_4 is $\|K_{\rightarrow 10}\|$. We also can use connectives $\rightarrow, \equiv$, as they can be defined by $\neg, \vee$ and $\wedge$ by the usual way.

By this correspondence, we can build a mapping i from $\{0,1\}^n$ to $\mathbf{V}$ in the following way. If $(t_1, \ldots, t_n) \in \{0,1\}^n$ then $i(w)$ is $\left[\frac{v}{2^n}, \frac{v+1}{2^n}\right)$ where $v = \sum_{k=1}^{n} t_k 2^k$.

Let $\Phi_1, \ldots, \Phi_m$ be an enumeration of all the subformulae of Φ satisfying that any formula is preceded by its subformulae (consequently, $\Phi_m = \phi$). The algorithm gives the next part of the computation sequence $(K_{3n-2+1}, \ldots, K_{3n-2+m})$ in the following way. For each $i \in \{1, \ldots, m\}$,

$K_{3n-2+i} =$

$$\begin{cases} (AND, 3n-2+j, 3n-2+k) \text{ if } \phi_i = \phi_j \wedge \phi_k, \\ (OR, 3n-2+j, 3n-2+k) \text{ if } \phi_i = \phi_j \vee \phi_k, \\ (NOT, 3n-2+j) \text{ if } \phi_i = \neg\phi_j, \\ (AND, 3j-2, 3j-2) \text{ if } \phi_i = x_j. \end{cases}$$

By induction on j the following statement can be verified: *For each* $j \in \{1, \ldots, m\}$, $\|K_{\rightarrow 3n-2+j}\| =$

$\{r \in [0,1) :$
$\phi_j\,[[r \in \|K_{\rightarrow 1}\|, r \in \|K_{\rightarrow 4}\|, \ldots, r \in \|K_{\rightarrow 3n-2}\|]] = 1\}$.

Here $\psi[[t_1, \ldots, t_n]]$ denotes the truth value of ψ by truth valuation $(x_1 \rightarrow t_1, \ldots, x_n \rightarrow t_n)$. By this fact, once we have i, we represent visually all the subformulæ of the input formula by a homomorphism h from the Boolean algebra of the propositional formulæ into $\mathbf{V}$ as follows.

$h(\psi) = \bigcup\{i(t_1, \ldots, t_n) : \psi[[t_1, \ldots, t_n]] = 1\}$.

In Fig. 1 line 5 shows the resulted interval-value of the propositional formula C_0. This representation is very natural and easily realizable even for 5-6 and more variables. In principle, this representation works with an arbitrary number of propositional variables, only the resolution of the visual representation give a limit. Having the interval-values of the propositional variables the solution of SAT or tautology problems is based on truth-tables: the interval-values represent bit-sequences. The following part of the representation is more interesting.

4. Visual solution of a PSPACE-complete problem

We show the visual reasoning power of the interval-valued computations by a PSPACE-complete problem, namely, the problem whether a quantified propositional formula is true. It is also decidable by a linear interval-valued computation ([11]). Also for this case, we show that the sequence of interval-values produced by this computation represents visually the full information needed to understand the solution of the given case of the problem.

We continue the computation sequence in the previous section with $K_{3n-2+m+1}, \ldots, K_{3n-2+m+8n}$ in such a way that for each integer $j < n$, the following holds:

$$\begin{aligned} &\|K_{\rightarrow 3n-2+m+8(j+1)}\| = \\ &((Lshift(\|K_{\rightarrow 3n-2+m+8j}\|, \|K_{\rightarrow 3(n-j)-2}\|) \cap \\ &\qquad\qquad \|K_{\rightarrow 3(n-j)-2}\|) \\ &\qquad \cup \\ &\qquad \|K_{\rightarrow 3n-2+m+8j}\|) \\ &\cup \\ &((Rshift(\|K_{\rightarrow 3n-2+m+8j}\|, \|K_{\rightarrow 3(n-j)-2}\|) \cap \\ &\qquad\qquad \overline{\|K_{\rightarrow 3(n-j)-2}\|}) \\ &\qquad \cup \\ &\qquad \|K_{\rightarrow 3n-2+m+8j}\|), \end{aligned}$$

if $n-j$ is even,

and

$$\begin{aligned} &(Lshift(\|K_{\rightarrow 3n-2+m+8j}\|, \|K_{\rightarrow 3(n-j)-2}\|) \cap \\ &\qquad\qquad \|K_{\rightarrow 3(n-j)-2}\| \cap \\ &\qquad\qquad \|K_{\rightarrow 3n-2+m+8j}\|) \\ &\cup \\ &(Rshift(\|K_{\rightarrow 3n-2+m+8j}\|, \|K_{\rightarrow 3(n-j)-2}\|) \cap \\ &\qquad \overline{\|K_{\rightarrow 3(n-j)-2}\|} \cap \|K_{\rightarrow 3n-2+m+8j}\|) \end{aligned}$$

in the other case.

In this definition, we did not specify all the intermediate expressions between $K_{3n-2+m+8j}$ and $K_{3n-2+m+8(j+1)}$, they are the subexpressions of $K_{3n-2+m+8(j+1)}$ needed

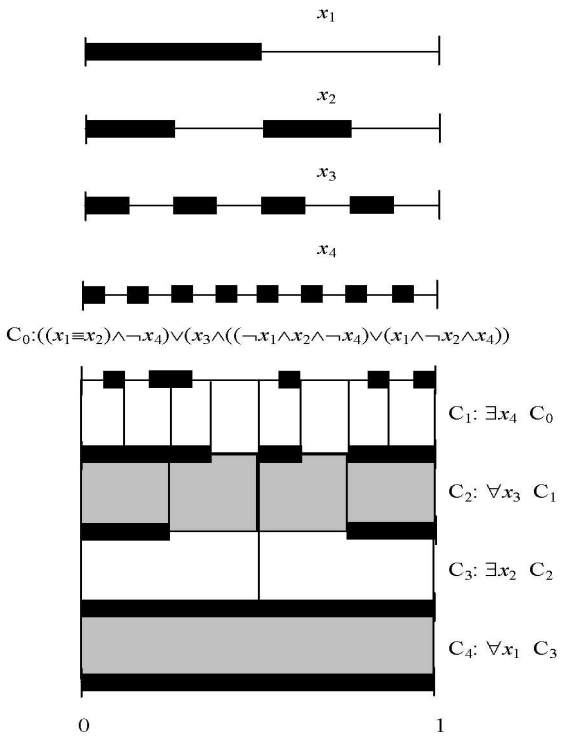

Figure 1. Example for testing validity of a quantified propositional formula

to express $K_{3n-2+m+8(j+1)}$ from $K_{3n-2+m+8j}$ and $K_{3(n-j)-2}$.

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. With these notations, the following holds:

For each $j \in \{0, \ldots, n\}$ *and for all* $r \in [0, 1)$*:*

$r \in \|K_{\to 3n-2+m+8j}\|$ *if and only if*

$$Q_{n-j+1}t_{n-j+1} \ldots Q_n t_n$$
$$\Phi\big[\big[r \in \|K_{\to 3\cdot 1-2}\|, \ldots, r \in \|K_{\to 3(n-j)-2}\|,$$
$$t_{n-j+1}, \ldots, t_n\big]\big] = 1.$$

The last statement is the base of our visual representation. The sequence of interval-values $\|K_{\to 3n-2+m+8j}\|$, $(j \in \{0, \ldots, n\})$, shows visually the steps of computation corresponding to the application of propositional quantifiers.

A $\forall$-step means checking the interval AND its corresponding neighbour, while an $\exists$-step amounts to checking the interval OR its neighbour.

In this way the linear computation sequence of interval-values includes visually all the information needed to follow a proof for a quantified propositional formula.

In Figure 1 one can follow a detailed interval-valued computation deciding whether a given quantified propositional formula is true.

5. Conclusions

While interval temporal logic ([1]) deals with problems about interval-values representing non-contiguous events, our system efficiently computes classical decision problems with the help of computations on such interval-values. By the visual power of the system one can easily understand the reasoning itself. Our system does not use the interval-relations of [8], but some of these relations can be expressed by the logic of interval-values. To make explicit comparisons of these systems is a topic of future work.

6. Acknowledgements

The work has been supported by the grants OTKA T049409, by the Hungarian Ministry of Education and by the KPI and NKTH.

References

[1] J. Allen. Maintaing information about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[2] B. Nagy and G. Allwein. Diagrams and Non-monotonicity in Puzzles. *Proceedings of Diagrams'2004, Cambridge, England, LNCS/LNAI* 2980:82–96, Diagrammatic Representation and Inference, eds.:A. Blackwell, K. Marriott, A. Shimojima, 2004.

[3] D. E. Anderson and F. L. Cleaver. Venn-type diagrams for arguments of n terms. *J. Symbolic Logic*, 30:113–118, 1965.

[4] P. Balbiani, J.-F. Condotta, L. F. del Cerro and A. Osmani. Reasoning about generalized intervals. *in: F. Giunchiglia (ed), Artificial Intelligence: Methodology, Systems and Applications, LNCS-LNAI* 1480:50–61, 1998.

[5] K. B. Chilakamarri, P. Hamburger and R. E. Pippert. Venn diagrams and planar graphs. *Geometriae Dedicata*, 62:73–91, 1996.

[6] A. W. F. Edwars. *Cogwheels of the Mind: The Story of Venn Diagrams*. John Hopkins University Press, 2004.

[7] D. W. Henderson. Venn diagrams for more than four classes. *American Mathematical Monthly*, 70:424–426, 1963.

[8] Z. Kulpa. A diagrammatic approach to investigate interval relations. *J.Visual Lang.and Computing*, 17:466-502, 2006.

[9] B. Nagy. An interval-valued computing device. *Proceedings of CiE2005, Computability in Europe: New Computational Paradigms, Amsterdam eds: S. B. Cooper, B. Loewe, J. Tucker, 166-177*.

[10] B. Nagy. Reasoning by intervals. *4th International Conference on the Theory and Applications of Diagrams, Stanford(CA), USA, LNCS/LNAI* 4045:145–147, 2006.

[11] B. Nagy and S. Vályi. Solving a pspace-complete problem by a linear interval-valued computation. *Proceedings of CiE2006, Computability in Europe: Logical Approaches to Computational Barriers, Swansea, Report no. CSR-7-2006, eds. A. Beckmann, U. Berger, B. Loewe, 216–225*.

[12] B. Nagy and S. Vályi. Interval-valued computations and their connections with PSPACE. *Theoretical Computer Science*, accepted.

[13] F. Ruskey and M. Weston. A survey of Venn diagrams. *Electronic Journal of Combinatorics*, 12, 2005.

[14] D. Woods and T. Naughton. An optical model of computation. *Theoretical Computer Science*, 334:227–258, 2005.