The characterization of NP within interval-valued computing

Benedek Nagy

(University of Debrecen, Faculty of Informatics)

Sándor Vályi

(College of Nyíregyháza,
Faculty of Natural Sciences and Informatics)

LR12

Outline

- Motivation
- Introduction to interval-valued computations
- Examples:
  - deciding SAT and QSAT
  - characterization of PSPACE
  - prime factorization
- Characterization of NP
- Questions

Motivation

- Fuzzy logic (interval-values form an infinite Boolean algebra)
- Computing with intervals (optical computing)
- Nice visualization

History

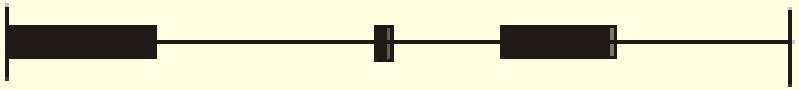
- New computing paradigms -- last 20 years:
  - DNA computing
  - Membrane computing
  - Quantum computing
  - Interval-valued computing
    coined by Benedek Nagy (2005)
  - Development: BN + SV, joining other people

Interval-values

Each interval $[a,b)$ with $0 \le a \le b \le 1$ is an atomic interval which contains all the points x between a and b ($a \le x \le b$). Interval-valued byte: bits indexed by [0,1)

Interval-value: a subset of [0,1), finite union of atomic intervals. It can be represented by its characteristic function: $[0,1) \to \{0,1\}$

Graphical representation:



Classical Computers

- **Based on two-valued Boolean Logic**

Truth-table of basic logical operators

Name	1st variable	2nd variable	Negation	Conjunction	Disjunction	Implication
Sign	A	B	$\neg A$	$A \wedge B$	$A \vee B$	$A \rightarrow B$
values	0	0	1	0	0	1
	0	1	1	0	1	1
	1	0	0	0	1	0
	1	1	0	1	1	1

Other operations are derived:

equivalence: $A \equiv B := (A \rightarrow B) \wedge (B \rightarrow A)$, **'xor'**: $A \oplus B := A \equiv \neg B$

'nor': $A|B := \neg A \wedge \neg B$, 'nand': $A \& B := \neg A \vee \neg B$.

Classical Computers

- Bits and bytes
  - Bit: unit of information (answer for a YES/NO question)
  - Byte: unit of data operations
  - More bits in a byte $\rightarrow$ higher level the CPU.
  - Logical operators - bitwise:

value of A	negation of A	value of B	conjunction of A and B	disjunction of A and B
$x_1\ x_2\ \ldots\ x_n$	$\neg x_1 \neg x_2 \ldots \neg x_n$	$y_1\ y_2\ \ldots\ y_n$	$x_1\ x_2\ \ldots\ x_n$ $\wedge\ \wedge\ \ldots \wedge$ $y_1\ y_2\ \ldots\ y_n$	$x_1\ x_2\ \ldots\ x_n$ $\vee\ \vee\ \ldots \vee$ $y_1\ y_2 \ldots\ y_n$

Classical Computers

Non-Boolean operators

example: SHIFT (Shifting bits in a byte)

value of A in binary code	left-shift A	right-shift A
$x_1\ x_2\ \ldots\ x_n$	$x_2\ x_3\ \ldots\ x_n\ 0$	$0\ x_1\ x_2\ \ldots\ x_{n-1}$

Generalizing the classical computations

- (Turing) the tape alphabet is fixed (in size): the information can be stored in a fixed size cell is limited
- (Neumann) the number of bits in a byte is also fixed
- **Idealization** in the classical paradigm: the size of the tape can grow arbitrarily if needed
- **Interval-values:** the information can be stored in a data processing unit is not limited
- **Idealization** of interval-valued computing:. the density of the memory can be raised unlimitedly
- Classical logic → fuzzy logic (interval-valued)

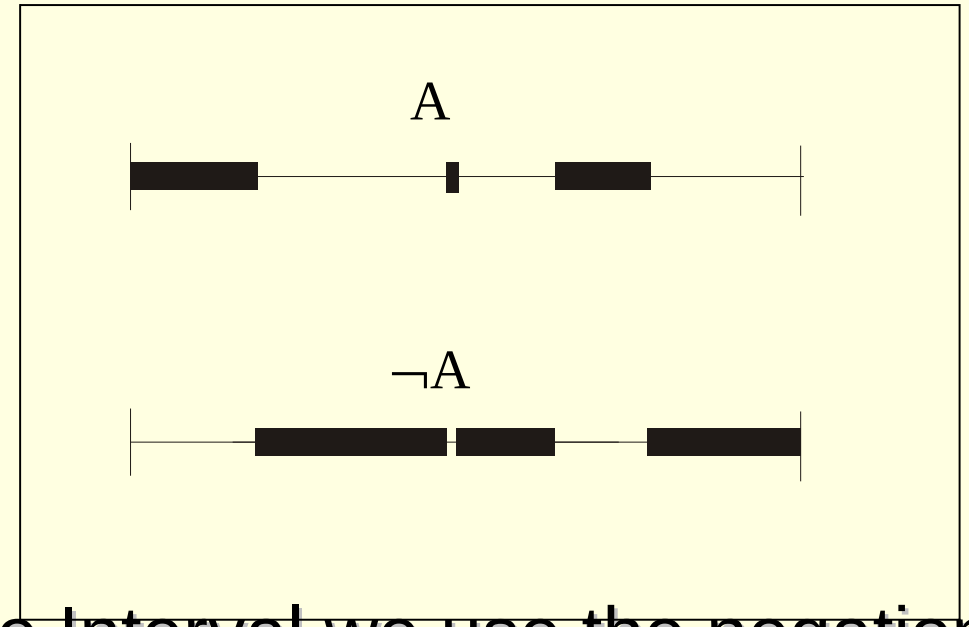
Interval-computation

- Logical gates, circuits:
- Classical computation is based on classical two-valued logic.
- Interval-computation is based on interval-valued logic (the number of bits in a unit can be increased during the computation).

BOOLEAN OPERATIONS:
Logic of Interval-values

- Negation

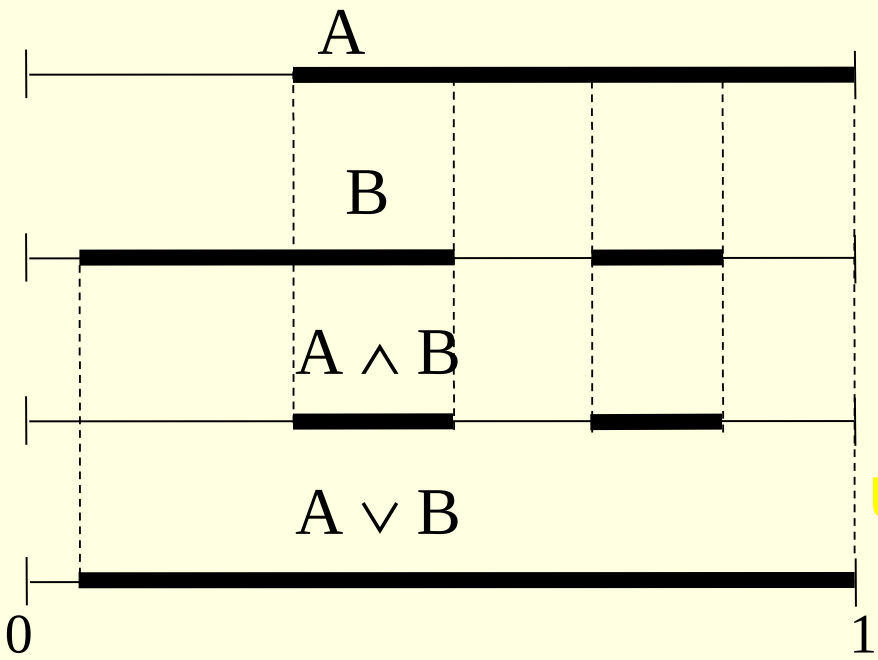
set theoretical complement



- for each point of the Interval we use the negation of the original characteristic value

Logic of interval-values

- Conjunction and Disjunction



set theoretical intersection

union

we use the logical operation pointwise for the characteristic values of the arguments

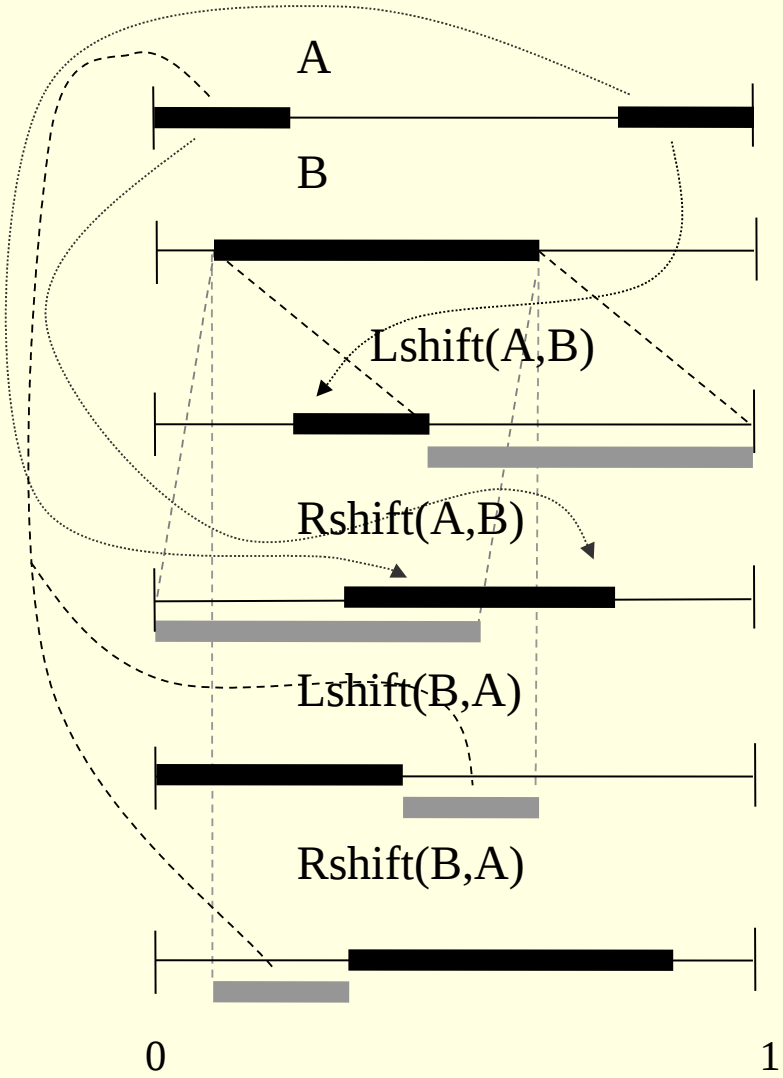
Non-Boolean operations for Interval-values

- Tool:
  - *Flength*(A) = $b - a$, if A contains the interval $[a,b)$ and A does not contain any interval (a,c) with $c \geq b$, moreover the difference of A and $[a,b)$ does not contain any point x with $x < a$.
    Flength(A) = 0, in other cases.
  - It is the length of the first component of the interval-value A.

Non-Boolean operations for Interval-values

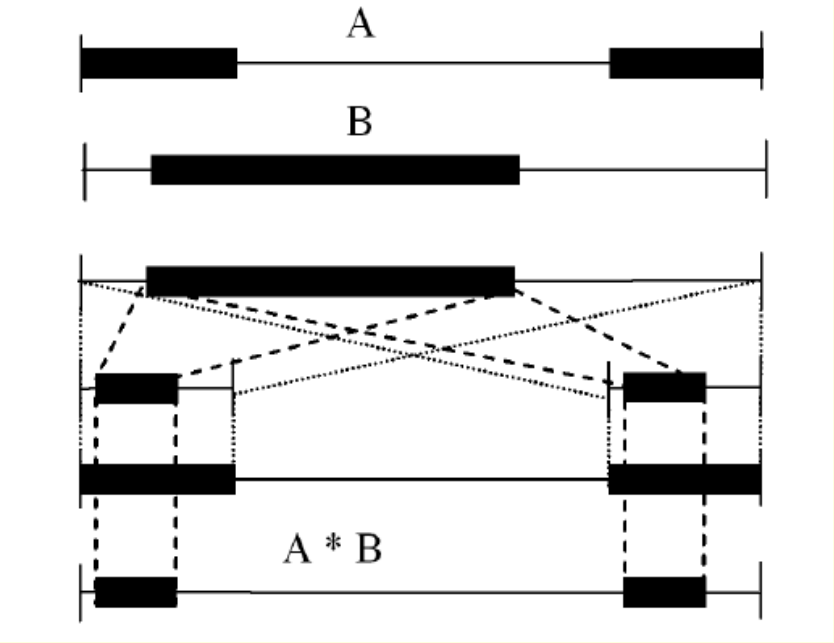
- SHIFT: (we define the two directions in a different way to get more effective device)

let Lshift(A,B) and Rshift(A,B) be the interval-values given after A was shifted to the left and to the right by *Flength*(B).



Product of interval-values

- Zooming an interval-value onto the complements of the second.



Nagy: *An interval-valued computing device* (2005)
, CiE 2005, "Computability in Europe": New Computational Paradigms, Amsterdam, Netherlands, 166-177.

Interval-valued computations

- Computation sequence: a sequence of applications of operators to already computed interval-values.
  (like Boolean networks)
- The sequence starts with the only predefined constant of the system FIRSTHALF : [0,1/2) .

- A language L is *decidable by an interval-valued computation* iff there is an algorithm *A* that for each input word *w* constructs an appropriate computation sequence with last element *A*(*w*) such that ***w* ∈ L if and only if the interval-value of *A*(*w*) is nonempty**.

Computation of functions

- Discrete functions can also be computed by this system.
- Special operation gives 0/1 bit on the output: (*OUTPUT, i)*, where *i is an index less than the index of the actual instruction.*
- $\perp \Rightarrow$ 0 is written out, otherwise 1.
- these output bits are concatenated during the computation process.

Decidability by interval-valued computations vs. classical decidability

- A language L is *decidable by an interval-valued computation* iff there is an algorithm *A* that for each input word *w* constructs an appropriate computation sequence with last element *A*(*w*) such that ***w* ∈ L if and only if the interval-value of *A*(*w*) is nonempty**.
- So, decidability by interval-values trivially coincides with classical decidability.

Resource restrictions

- We say that a language L 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* , *A* constructs an appropriate interval-valued computation sequence A(w) such that |
  - |A(w)| is not greater than c · |w| and
  - w in L if and only if the value of the computation sequence A(w) is nonempty.

Polynomial restriction

- We say that a language L is **decidable by a polynomial interval-valued computation** if and only if
- there is a polynom *P* and a **logarithmic space algorithm** *A* with the following properties. For each input word *w* , *A* constructs an appropriate interval-valued computation sequence A(w) such that |
  - |A(w)| is not greater than P(|w|) and
  - w in L if and only if the value of the computation sequence A(w) is nonempty.

First examples

- Solution to SAT
  - Independent interval-values for the variables
- Solution to QSAT
  - Dealing with boolean quantifiers

Solving SAT by interval-values

- Let n be the number of the variables of the SAT.
- Let FIRSTHALF be a constant interval-value: [0,1/2).

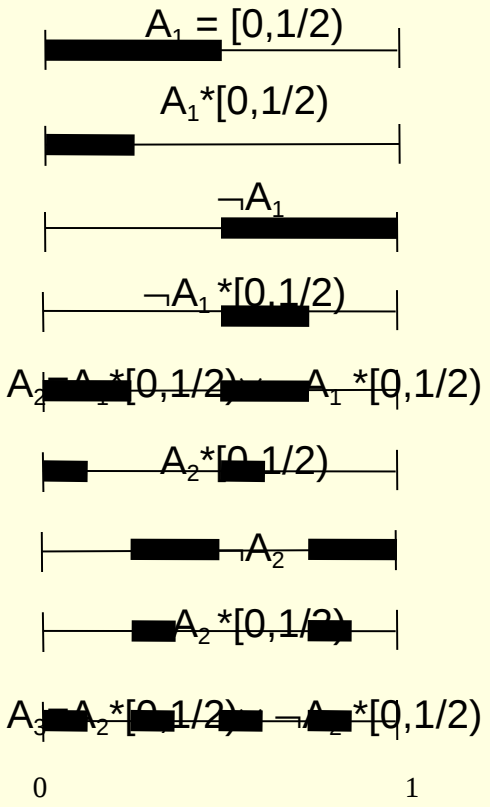
A computing sequence : starting from FIRSTHALF.

- That is: $S_0,\ldots, S_n$, ($n \in$ **N**),
- where S_0 = FIRSTHALF,
- for each $i<n$: S_i = **op**(S_k, S_j), where $k,j<i$
- where **op** can be a Boolean operator, a shift or product (in negation only the first operand is used)

Solving SAT

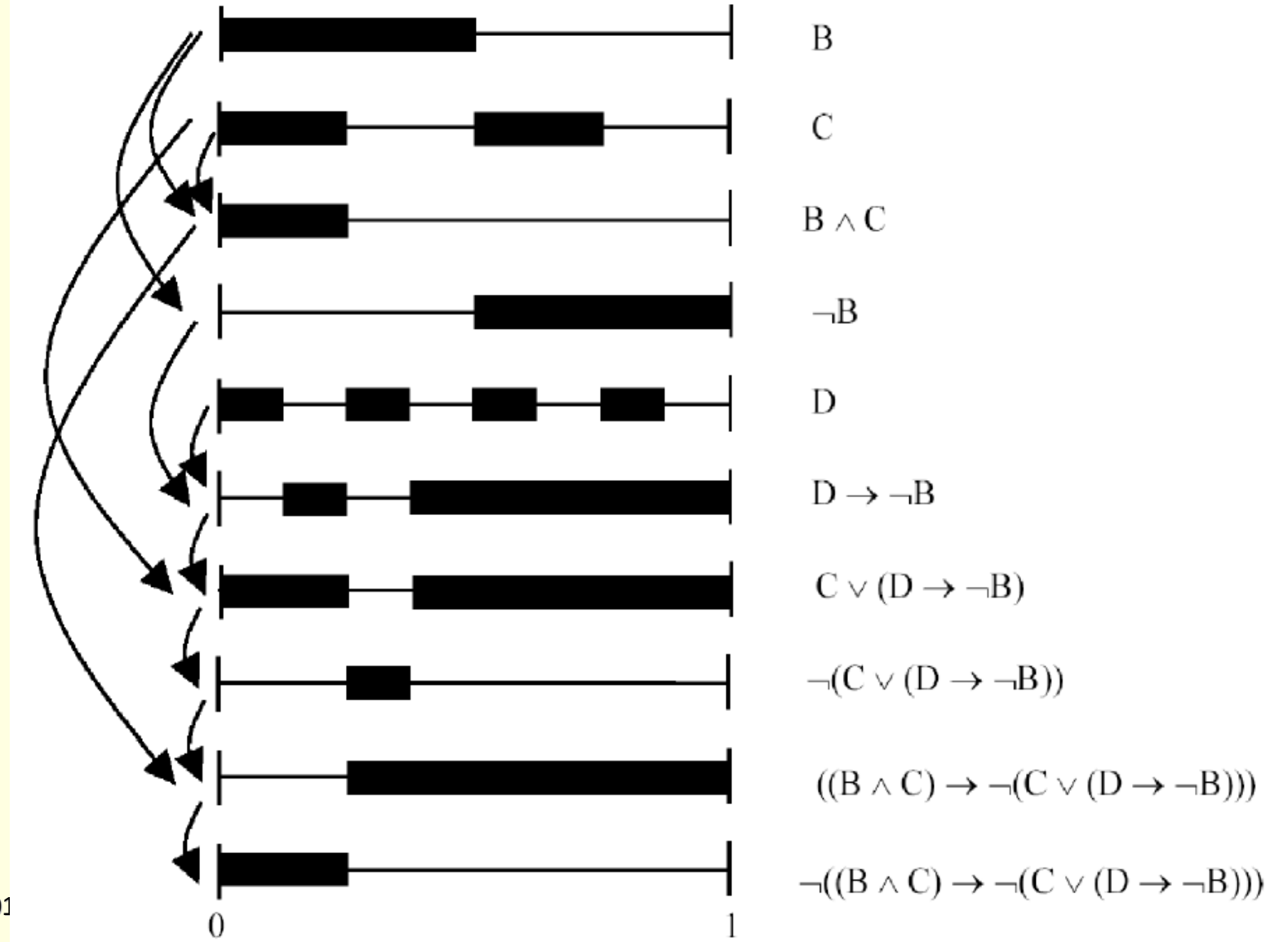
- We need to construct all possible combinations:
- It is possible by product, negation and union. Their number is linear on the number of variables:
- $A_{i+1}=A_i*[0,1/2)\vee\neg A_i*[0,1/2)$
- So, in this way the *i*th variable is:

$$A_i = \bigcup_{j=0}^{2^{i-1}-1} \left[\frac{j}{2^{i-1}}, \frac{2j+1}{2^i}\right)$$



Satisfiability of

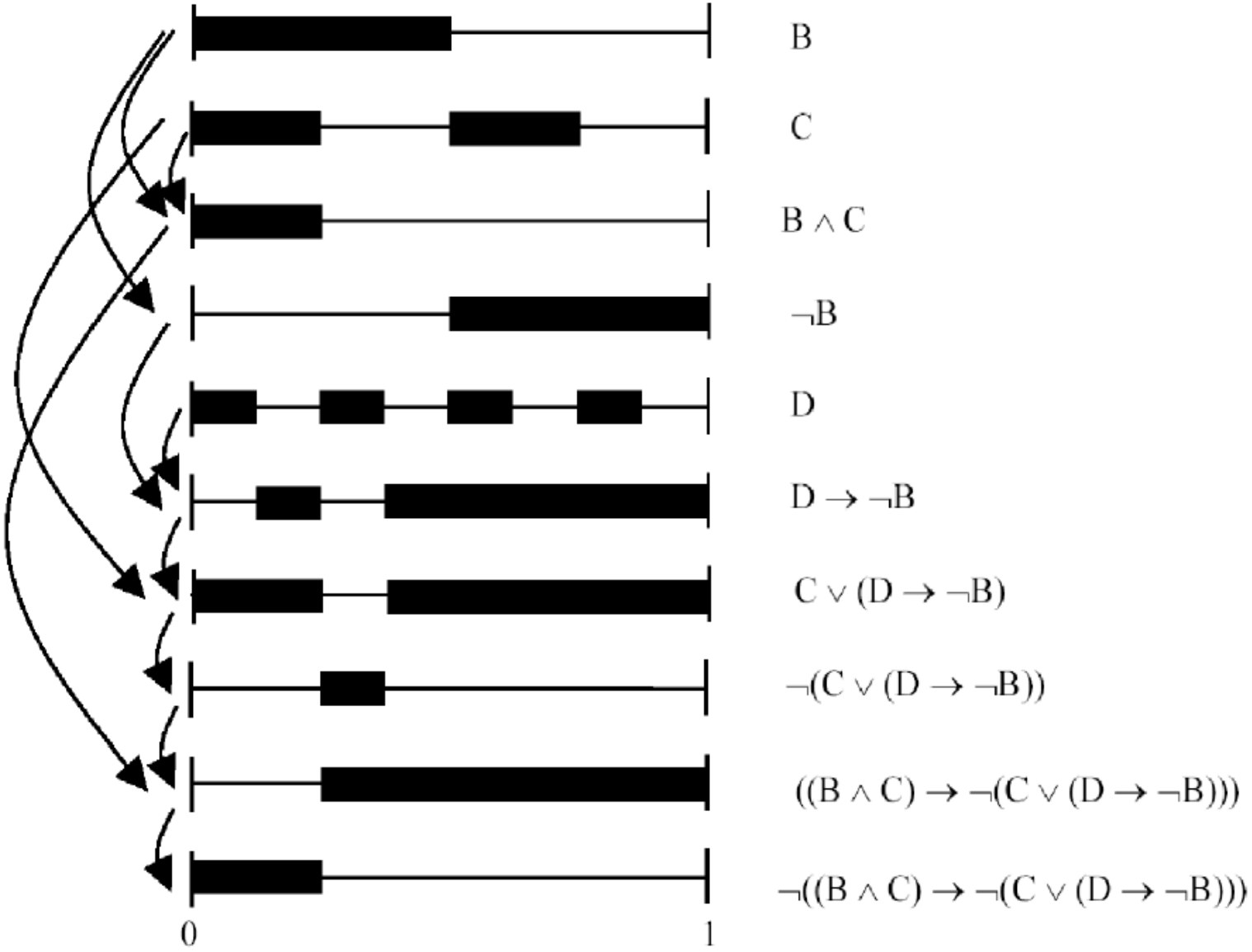
$\neg((B \wedge C) \rightarrow \neg(C \vee (D \rightarrow \neg B)))$



- Similar to the evaluation by a truth-table, but
- all possible combinations of the truth-values are evaluated parallely.

- Linear number of interval-valued operators to solve SAT
- The computation is visualised by intervals:
  - Interval-values of independent variables.
  - Logical operations (as in truth table).

Analogous truth-table - knowing the length of the smallest component



11110000
11001100
11000000
00001111
10101010
01011111
11011111
00100000
00111111
11000000

QSAT

- Formula where all the Boolean variables are quantified in the following way:

$$\forall t_1 \exists t_2 \ldots Q_n \varphi \text{ where } Q_i \text{ is } \forall \text{ for odd } i \text{ and } \exists \text{ for even } i.$$

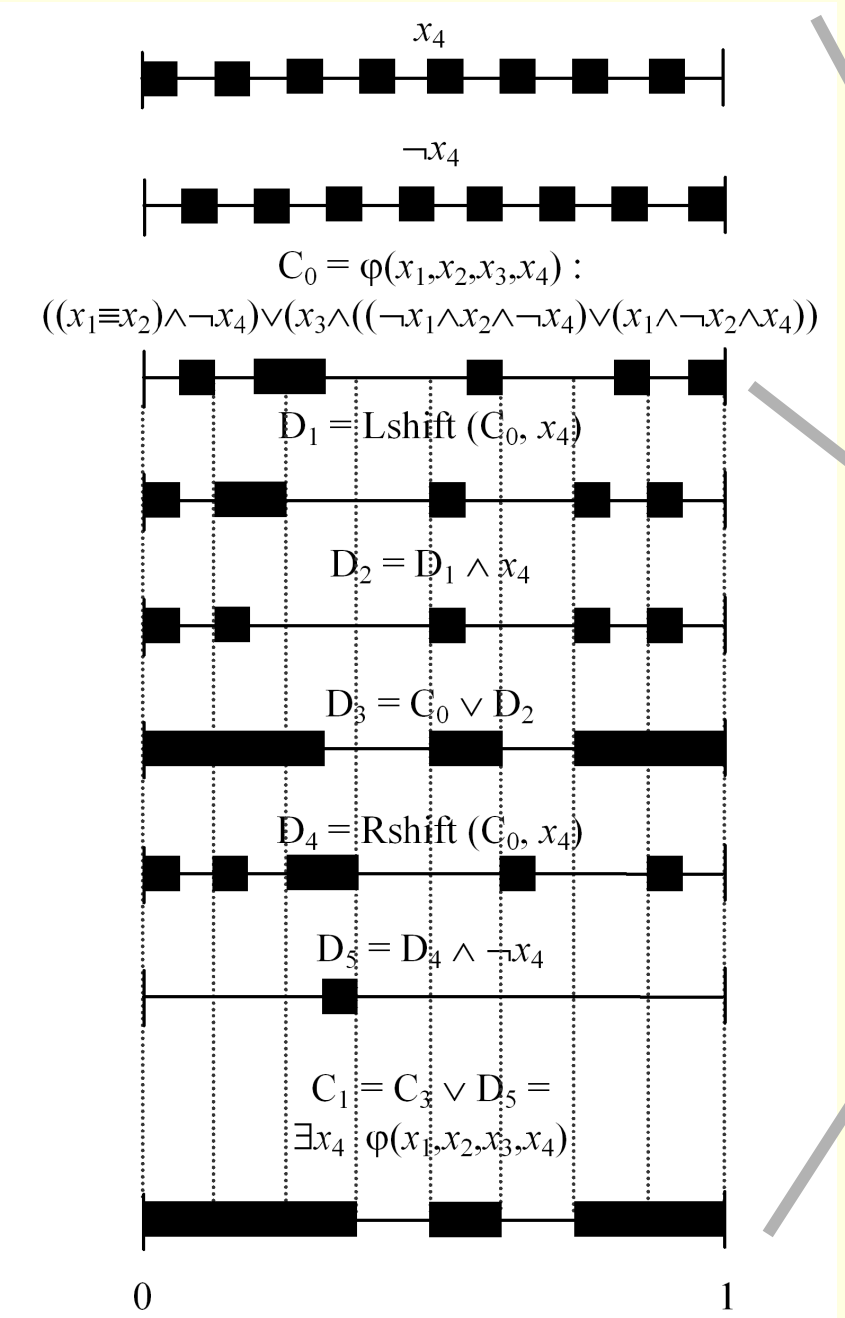
- It is a known PSPACE-complete problem to decide whether it is true or false.
- It is true if

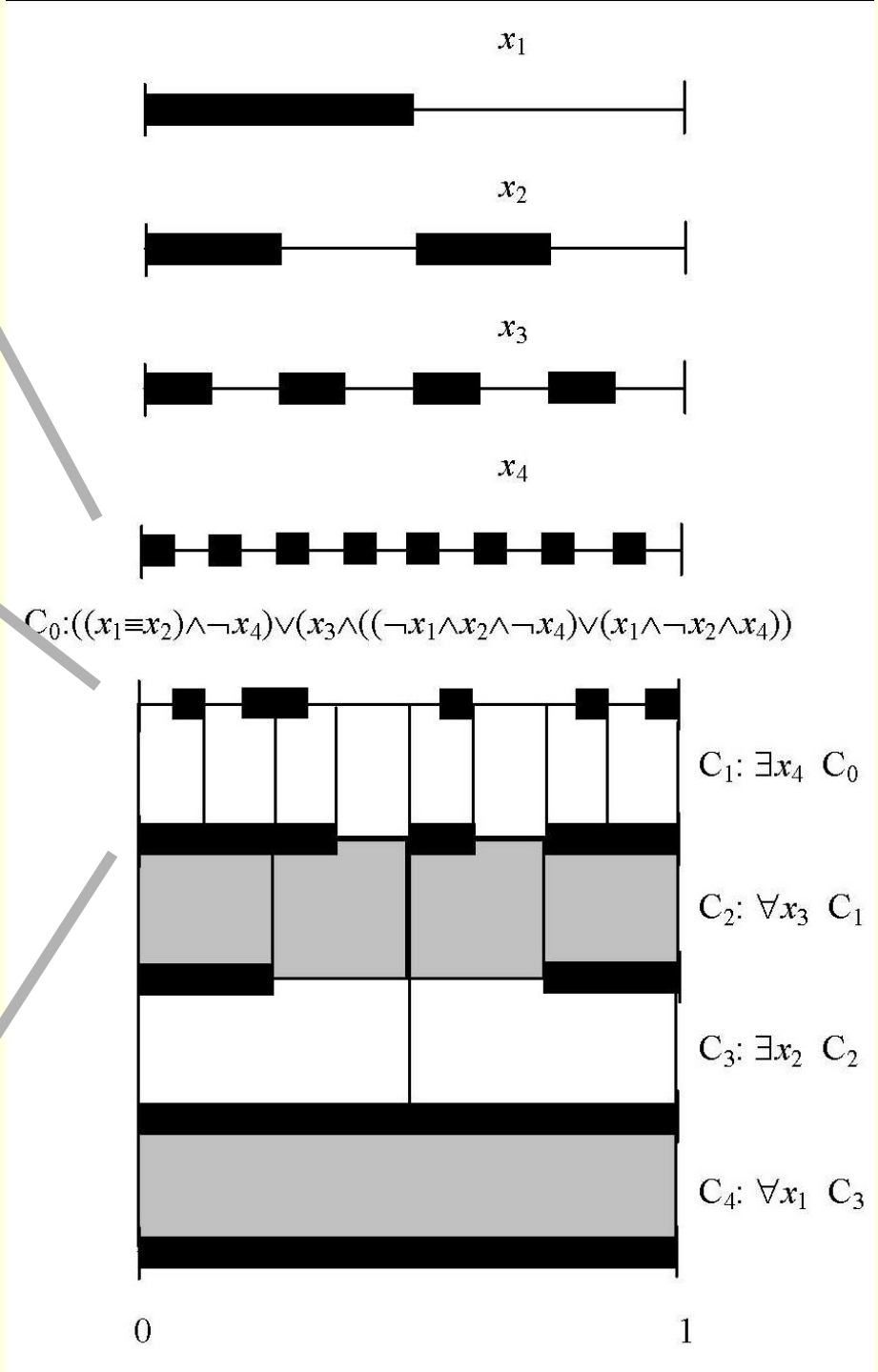
$$\forall t_1 \in \{0,1\} \exists t_2 \in \{0,1\} \ldots Q_n t_n \in \{0,1\} \varphi(x_1 : t_1, \ldots, x_n : t_n)$$

Visual solution by interval-values

- Variables
- Boolean formula (SAT)
- Dealing with quantifiers:
  look for the values of appropriate neighbours

EXAMPLE 1

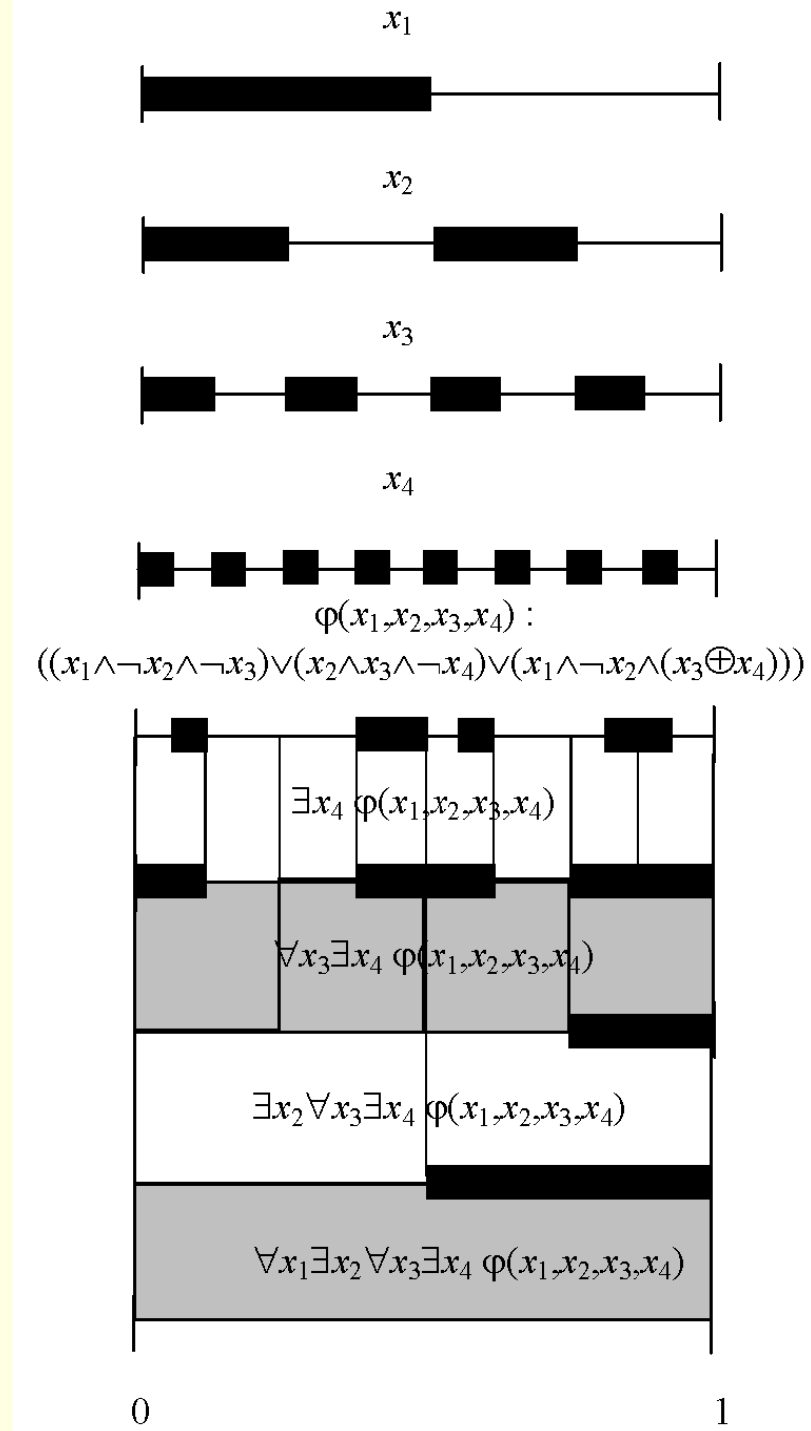

x_4
$\neg x_4$
$C_0 = \varphi(x_1,x_2,x_3,x_4)$:
$((x_1 \equiv x_2) \wedge \neg x_4) \vee (x_3 \wedge ((\neg x_1 \wedge x_2 \wedge \neg x_4) \vee (x_1 \wedge \neg x_2 \wedge x_4))$
$D_1 = \text{Lshift}(C_0, x_4)$
$D_2 = D_1 \wedge x_4$
$D_3 = C_0 \vee D_2$
$D_4 = \text{Rshift}(C_0, x_4)$
$D_5 = D_4 \wedge \neg x_4$
$C_1 = C_3 \vee D_5 =$
$\exists x_4 \ \varphi(x_1,x_2,x_3,x_4)$
0
1


x_1
x_2
x_3
x_4
C_0:$((x_1 \equiv x_2) \wedge \neg x_4) \vee (x_3 \wedge ((\neg x_1 \wedge x_2 \wedge \neg x_4) \vee (x_1 \wedge \neg x_2 \wedge x_4))$
C_1: $\exists x_4$ C_0
C_2: $\forall x_3$ C_1
C_3: $\exists x_2$ C_2
C_4: $\forall x_1$ C_3
0
1

EXAMPLE 2

This formula

is not satisfiable.



- The PSPACE-complete problem QSAT is solvable by a **linear** interval-valued computation.

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

Restricted polynomial restriction

- We say that a language L is **decidable by a restricted interval-valued computation** if and only if
- there is a polynom *P* and a logarithmic space algorithm *A* with the following properties. For each input word *w* , *A* constructs an appropriate interval-valued computation sequence A(*w*) such that |
  - |A(w)| is not greater than P(|*w*|) and
  - *w* in L if and only if the value of the computation sequence A(*w*) is nonempty
  - A(w) containing product operators only of the form *(Product, 0, n) (product with FIRSTHALF)*

Languages decided by restricted polynomial interval-valued computations vs. PSPACE

- **Theorem:** The class of languages decidable by a restricted polynomial interval-valued computation is equal to PSPACE.

The reverse direction

One direction of this class equation is already achieved.

For the reverse:

a quadratic space algorithm decides whether the value of an input interval-valued computation sequence is

equal to the full [0, 1).

first: a recursive algorithm (exponential time is guaranteed)

second: a back-track like control in such a way that the needed memory is limited by a quadratic function of the length of the input computation sequence.

- Nagy, Vályi:, *Interval-valued computations and their connection with PSPACE*, Theoretical Computer Science - TCS 394/3 (2008), 208-222.)

Characterizing NP within interval-valued computing

'Trying_all' normal form of interval-valued computation sequences: there exists a polynom Q(.) that for any input word *w*

it starts with generating some number (less than Q(|*w*|)) of 'independent' interval-values, as in the case of SAT
and continues using only Boolean operators no more times than Q(|*w*|)

L is *decidable by a trying_all interval-valued computation* if there exists an logspace algorithm that generates for arbitrary input word a computation sequence of trying_all normal form that decides L

All member of NP is decidable by a trying_all normal form interval-valued computation

If L is in NP

– let P(.) be the witness size limiting polynom

– W is the witness language (in P => there exists a uniform Boolean network BN to decide it)

– Let w is an input word for the algorithm that should construct the computation sequence of trying_all n. f.

– an amount of P(|w|) independent interval-values is generated and then the operators of BN executed step-by-step on them

then it is decidable by a trying_all n.f . interval-valued computation

If L is decidable by a trying_all i-v computation

a simple witness language exists for L:

if a word w is in L then this fact can be witnessed by the description of the position in [0,1) where the final result of the computation sequence was nonempty

then it is in NP

this simple idea is the base for two result:

Nagy, Vályi: *Prime factorization by interval-valued computing*, Publicationes Mathematicae Debrecen 79/3-4 (2011), 539-551.

Nagy, Vályi: *Discrete logarithm by interval-valued computing,* DCM 2012, 8th International Workshop on Developments in Computational Models, Cambridge, England

What is the difference

Between the computations for PSPACE and NP?

While deciding QSAT we use heavily shifts computations with shifts cannot be witnessed in poly size (at least: now we cannot)

Conclusions

- The problems in NP are effectively solvable in this paradigm by its massive parallellism (not surprising)

Open questions

Not restricted polynomial sized interval-valued computations ? EXPSPACE

Other paradigm: computation starting with more sophisticated input D/A conversion and output A/D

Physical implementation (for example: interval-value is the positivity places of a $[0,1) \rightarrow IR$ function)

If **V** is the set of possible interval-values then $Th(\mathbf{V},\neg,\cap,\cup,Lshift,Rshift,Product)$ is decidable? Even the set of its equations is decidable?

References

- Nagy: *An interval-valued computing device* (2005)
  , CiE 2005, "Computability in Europe": New Computational Paradigms, Amsterdam, Netherlands, 166-177.
- Nagy, Vályi: *Solving a PSPACE-complete problem by a linear interval-valued computation*, CiE 2006, Computability in Europe 2006: Logical Approaches to Computational Barriers, University of Wales Swansea, UK, 216-225.
- Nagy, Vályi: *Interval-valued computations without the product operator* (2007), ICAI 2007, 7th International Conference on Applied Informatics, Eger, Hungary, volume I. 83-90.
- Nagy, Vályi: *Interval-valued computing as a visual reasoning system* (2007), DMS2007, The Thirteenth International Conference on Distributed Multimedia Systems - VLC'2007, International Workshop on Visual Languages and Computing, San Francisco, CA, USA, 247-250.
- Nagy, Vályi: *Visual reasoning by generalized interval-values and interval temporal logic* (2007), VLL 2007, Workshop on Visual Languages and Logic, - VL/HCC 07, IEEE Symposium on Visual Languages and Human Centric Computing, (CEUR Workshop Proceedings Vol-274), Coeur d'Aléne, Idaho, USA, 13-26.
- Nagy,Tajti: Solving Tripartite Matching by Interval-valued Computation in Polynomial Time (2008), CiE 2008, Fourth Conference on Computability in Europe: Logic and Theory of Algorithms (Local Proceedings), Athens, Greece, 435-444
- Nagy, Vályi:, Interval-valued computations and their connection with PSPACE, Theoretical Computer Science - TCS 394/3 (2008), 208-222. (co-author: S. Vályi)

Thank You for your attention !

Happy birthday, István!