University of Texas at El Paso
DigitalCommons@UTEP

Departmental Technical Reports (CS) | Department of Computer Science

1-1-2011

Designing, Understanding, and Analyzing Unconventional Computation: The Important Role Of Logic And Constructive Mathematics

Vladik Kreinovich
University of Texas at El Paso, vladik@utep.edu

Follow this and additional works at: http://digitalcommons.utep.edu/cs_techrep

Part of the Computer Engineering Commons

Comments:
Technical Report: UTEP-CS-11-02
Published in *Applied Mathematical Sciences*, 2012, Vol. 6, No. 13, pp. 629-644

Recommended Citation
Kreinovich, Vladik, "Designing, Understanding, and Analyzing Unconventional Computation: The Important Role Of Logic And Constructive Mathematics" (2011). *Departmental Technical Reports (CS)*. Paper 591.
http://digitalcommons.utep.edu/cs_techrep/591

This Article is brought to you for free and open access by the Department of Computer Science at DigitalCommons@UTEP. It has been accepted for inclusion in Departmental Technical Reports (CS) by an authorized administrator of DigitalCommons@UTEP. For more information, please contact lweber@utep.edu.

Applied Mathematical Sciences, Vol. x, 201x, no. xx, xxx - xxx

Designing, Understanding, and Analyzing Unconventional Computation: The Important Role of Logic and Constructive Mathematics

Vladik Kreinovich

Department of Computer Science
University of Texas at El Paso
500 W. University
El Paso, TX 79968, USA
email vladik@utep.edu

Abstract

In this paper, we explain why, in our opinion, logic and constructive mathematics are playing – and should play – an important role in the design, understanding, and analysis of unconventional computation.

Mathematics Subject Classification: 03A05, 03F60, 03F65, 68Q05, 65G20, 03F03

Keywords: unconventional computations; logic; constructive mathematics

1 Introduction

Challenge. There are many practical problems for which:

- the algorithms are, in principle, known, but
- computations requires such a long time that we have to stop them midway and get poor quality results (if we get any meaningful results at all).

Traditional approaches to this challenge and their limitations. Traditional approaches to this challenge are:

- to design faster super-computers (*hardware*), and/or
- to design faster *algorithms*.

Both approaches are reasonable and sound reasonable. However, when people implement these approaches, they often implement them in a limited way.

For example, when computer engineers talk about faster computers, they usually talk mostly about new computers:

- which are innovative on the engineering level – in the sense that they implement new engineering ideas – but
- which are not that innovative on the level of fundamental physics – in the sense that the new designs use the same physical processes as the existing computers.

Similarly, when computer scientists design faster algorithms, they usually mostly design new algorithms for solving the exact same problem as before – albeit slightly faster. For example, the existing software usually implement algorithms for finding accurate solutions to the corresponding partial differential equations. This accuracy makes sense in the ideal situations, when we know the initial conditions with high accuracy. In practice, often, due to sparsity and inaccuracy of sensor data, we only have approximate inputs. In such situations, when inputs are only known with high uncertainty, it makes no sense to compute the results with a high accuracy.

Thus, when the traditional approaches are not sufficient, it is reasonable to try alternative approaches that overcome these limitations. Specifically:

- re *hardware:* use unconventional physical (and biological) processes;
- re *algorithms:* perform computations only up to accuracy that matches the input accuracy.

Our main claim. We claim that for both alternative approaches to succeed, it is crucial to further develop the corresponding tools of mathematical *logic* – and related methods of *constructive mathematics*.

2 Why Logic?

Stages of solving a problem. A computational solution to a problem consists of the following stages:

- first, we *specify* a problem, i.e., describe the user's problem in precise terms;
- then, we design and implement an *algorithm* for solving this problem;
- finally, we *verify* the corresponding program.

Let us show that logic is useful on all these three stages.

Logic has been efficiently used to specify problems. Sometimes, the problem is presented in terms of explicit algebraic or differential equations. However, in general, the correct formulation of the practical problem requires logical terms (logical connectives, quantifiers, etc.).

For example, if the task is to design a stable control, this means that for all deviations which are not too large, the trajectory will eventually return to the standard one. In precise terms, this means that there exists a bound Δ such that, for every moment t, if the distance $\rho(x(t), x'(t))$ does not exceed Δ, then for every accuracy $\varepsilon > 0$, there exists a moment $t_0 > t$ such that for every $t' \geq t_0$, the new trajectory is ε-close to the original one. In precise terms, this means that

$$\exists\Delta\forall t\forall x'(\rho(x(t), x'(t)) \leq \Delta \Rightarrow \forall\varepsilon\exists t_0\forall t'(t' \geq t_0 \Rightarrow \rho(x(t'), x'(t')) \leq \varepsilon)).$$

This is a simple example of a specification requiring quite a few quantifiers to describe.

Logic has been efficiently used to design algorithms. A logical specification not only provides a formalized *description* of the problem, it can often lead to a *solution* to the problem. For example, several *logic programming* languages (widely used in AI applications) make it possible to automatically transform logical specifications into a code; see, e.g., [25, 26, 33].

Comment. It is worth mentioning that a related work was done at Microsoft Research on Spec Explorer and Abstract State Machine.

Logic has been efficiently used in program verification. Not only the *problem* itself – i.e., the connection between the input and the desired output – can often be naturally described in terms of logic: the desired behavior of each computational module can also be naturally described in logical terms.

A natural requirement for a computational module is that for all inputs that satisfy a certain *pre-condition*, the result must satisfy the corresponding *post-condition*. For example, for a sorting module, the resulting list must be

sorted and it must consist of exactly the same elements as the input (albeit maybe in a different order).

Similarly to the way logic is used in problem specification, this logical description not only helps to *describe* the computations, it also helps to *reason* about computations – since one of the main objectives of logic is reasoning. In particular, logical tools are extremely important in *program verification*: once we formulate both the specification and the computations in logical terms, the verification of a program can be reduced to proving a precise logical result – that the specification condition is satisfied for all the program outputs.

Logic can also help in developing proofs. For simple programs, correctness proofs are often simple. For more realistic and more complex programs, such proofs may be complex – and thus not easy to develop.

The general experience of proofs in different logics has recently led to the emergence of logic-based automatic *proof assistant* programs, programs that help users develop such proofs (HOL, Coq, etc.).

3 First Approach: Computations with Limited Accuracy

Analyzing the first approach: computations with limited accuracy. The algorithms use both the sensor data and the results of (often time-consuming) auxiliary computations (e.g., computation of special functions). To speed up computations, we must determine which accuracy of these auxiliary computations is sufficient to provide the desired accuracy of the final result.

In precise terms, our main task is to compute the value $f(x)$, based on the value x computed at some previous computation steps. If we only need to compute the value $f(x)$ with accuracy $\varepsilon > 0$, then it is sufficient to compute x only with an accuracy $\delta > 0$ for which $\rho(x, x') \le \delta$ implies $\rho(f(x), f(x')) \le \varepsilon$.

Comment. In many useful applications, by the way, it is extremely important that we *guarantee* that the actual values are within the given bounds of the computational results – e.g., we want to guarantee that the spaceship hits the Moon, that the nuclear reactor regime stays within the stable area, etc.

Enter constructive mathematics. In practice, whatever value x we compute, we always need to compute it with some accuracy. In other words, we are given some rational number $\varepsilon > 0$, and we need to produce a rational number $r(\varepsilon)$ for which $|r(\varepsilon) - x| \le \varepsilon$. A real number that can be computed with an arbitrary accuracy is called *computable*. In precise terms, a computable real

number x is a number for which there exists an algorithm that, given a rational number ε, produces a rational number $r(\varepsilon)$ for which $|r(\varepsilon) - x| \le \varepsilon$.

Similarly, for each computational transformation $f : X \to Y$,

- we must *not only be able* to efficiently compute $f(x)$ given x,
- we must *also be able*, for any accuracy $\varepsilon > 0$, to efficiently produce $\delta > 0$ for which a δ-accurate approximation to x produces an ε-accurate approximation to $f(x)$:

$$\rho(x, x') \le \delta \Rightarrow \rho(f(x), f(x')) \le \varepsilon.$$

In other words, since we are interested in computations, we must focus on computable objects (i.e., objects computable with an arbitrary given accuracy), and on constructive mappings that enable us to transform computable objects into computable ones.

This need has been recognized for several decades already – actually, starting with the 1950s when the first computers appeared. There is a special branch of mathematics called *constructive mathematics* that deals with such definitions; see, e.g., [1, 2, 3, 4, 6, 18, 23].

At present, most research in constructive mathematics is devoted to specific problems in which specific algorithms are needed. These results are scattered around, are motivated mostly by specific problems, and are not easy to generalize and to use when a new specific problem appears. Thus, we arrive at:

Research Direction I.1. *Develop general constructive mathematics techniques, with a special emphasis on problems requiring intensive computations (such as large-scale partial differential equations).*

Need for constructive logic. For individual problems, we can creatively design appropriate algorithms – this is how many existing algorithms of constructive mathematics have been originally designed. However, designing a radically new algorithm is a very slow, time-consuming task. In most applications, we achieve good results by:

- decomposing a problem into subproblems with known algorithms, and
- combining these algorithms.

In the simplest situations, this decomposition can be described in algebraic or analytical terms. However, a general decomposition and combination requires full first order logic. For example:

- the solution of a certain equation can be reduced to the *existence* of some auxiliary polynomial, or

- in robust control, the stability under *all* possible values of the parameters within a certain domain is equivalent to certain inequalities.

Even a proper formulation of many problems, strictly speaking, requires first order logic: e.g., we want to make sure that a certain control strategy works *for all* possible perturbations that satisfy a certain property.

Thus, we need to find out when a logical combination of constructive results is also constructive. This analysis was started by Kolmogorov in 1920s. The resulting "constructive logic" is indeed actively used in constructive mathematics. Crudely speaking, in constructive logic:

- the formula $\exists x P(x)$ means that we can efficiently produce such an x for which the property $P(x)$ holds;
- the formula $\forall x \exists y P(x, y)$ means that there exists an algorithm $\varphi : X \to Y$ such that $P(x, \varphi(x))$ is always true.

In view of the need to develop a general constructive mathematics approach, we arrive at

Research Direction I.2. *Develop general constructive logic techniques, with a special emphasis on problems requiring intensive computations.*

Interval computations: general idea. In a general definition of constructive mathematics, we want to develop algorithms that work for *all* possible values of accuracy. In applications, the accuracy is usually *fixed*. In this case, it makes sense to develop simplified algorithms that work only for specific accuracy values. This is, in essence, the main idea of *interval computations* [10, 11, 12, 13, 21, 29] – what Yu. Matiyasevich has called *applied constructive mathematics*.

The name comes from the fact that for a single quantity, when we know the measurement result $\widetilde{x}$ with a known accuracy Δ, then all possible values of this quantity form an interval $[\widetilde{x} - \Delta, \widetilde{x} + \Delta]$.

Why intervals and not probability distributions. Traditional approach to situations with measurement inaccuracy is to assume that we know the probability distribution for the measurement error $\Delta x \stackrel{\text{def}}{=} \widetilde{x} - x$. Usually, it is assumed that this distribution is Gaussian; see, e.g., [38]. However, there are practical situations when we do not know this distribution.

Indeed, the distribution for Δx usually comes from the *calibration* of the corresponding measuring instrument (MI). To perform this calibration, we compare the results of measuring the same quantity by the available and by a "standard" MI which is several times more accurate than the given one.

Since the standard MI is much more accurate, the corresponding measurement errors can be safely ignored in comparison with the measurement errors of the original MI. In other words, we can safely assume that the results of the standard MI are error-free, i.e., that they practically coincide with the actual (unknown) values x of the corresponding quantity. Thus, the difference $\Delta x = \widetilde{x} - x$ can be approximately described as the difference between the results of measuring the same quantity by the original and the standard MI. After several measurements, we get a sample of values Δx, and from this sample, we reconstruct the desired probability distribution for Δx.

There are two types of practical situations when this procedure is not performed. The first is the case of state-of-the-art measurements, e.g., in fundamental science. In this case the measuring instrument that we use is the best, there is no better MI that can serve as a standard. Yes, it would be nice if near the Hubble telescope, there would be another telescope that would measure all the star coordinates with 5 times more accuracy – but the Hubble telescope is the best we have. In this situation, we cannot determine the probability distribution for Δx. At best, we have an upper bound Δ on the (absolute value of) this measurement error – i.e., we get an interval.

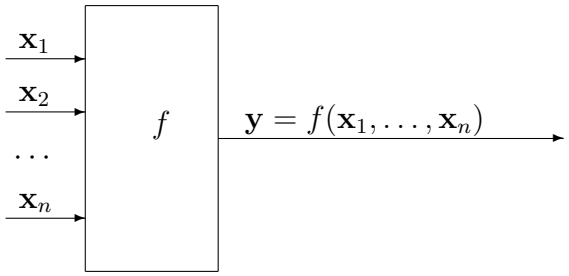
Another case when we do not know the probability distribution for Δx is the case of routine manufacturing on the factory floor. In this case, it is theoretically possible to calibrate every sensor, but calibration costs a lot of money – sensors are often cheap, but calibration means using the standard MI which is much more expensive. As a result, for most sensor used in manufacturing, it is too expensive to calibrate them. Instead of the probability distribution for Δx, we only use the upper bound Δ provided by the manufacturer of the corresponding MI (and the manufacturer must provide some bound, otherwise, if the manufacturer does not guarantee any accuracy, this is not a measuring instrument.)

Interval computations: precise formulation of the main problem. The main problem of interval computations is as follows:

- we have a data processing algorithm $f(x_1, \ldots, x_n)$ that transforms n real numbers $x_1, \ldots, x_n$ into a new value $y = f(x_1, \ldots, x_n)$;
- we do not know the exact values of the inputs x_i; instead, for every input i, we only know the interval $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$ of the corresponding value.

Different values $x_i \in \mathbf{x}_i$ lead, in general to different values $y = f(x_1, \ldots, x_n)$. Our objective is to find the range of possible value of y, i.e., the interval

$$\mathbf{y} = [\underline{y}, \overline{y}] = \{f(x_1, \ldots, x_n) : x_1 \in \mathbf{x}_1, \ldots, x_n \in \mathbf{x}_n\}.$$



Main problem of interval computations: computational complexity. It is known that the above main problem of interval computations is NP-hard even for quadratic f; see, e.g., [18]. In this case, when we cannot compute the exact range $\mathbf{y}$, a natural idea is to compute a good approximating *enclosure*, i.e., an interval $\mathbf{Y}$ that contains (encloses) the desired range: $\mathbf{Y} \supseteq \mathbf{y}$.

Comment. Of course, as with all other NP-hard problems, there are important cases for which efficient algorithm can compute the exact range.

Interval computations: inverse problem. So far, we have considered the *main* (*forward*) problem, when

- we know the range X_0 of possible values of x and
- we need to efficiently compute the corresponding range $f(X_0)$ of $y = f(x)$.

In some practical applications, we need to solve the *inverse problem*, when:

- we know the range Y_0 of y, and
- we must find the range X_0 which guarantees that $f(x) \in Y_0$.

Applications of interval computations. Interval computations have many applications; see detailed description in [10, 11, 12, 13, 21, 29].

Historically the first applications were to the design of trajectory of a spaceflight, a trajectory that is guaranteed to hit the target area under all possible uncertainties.

Another important practical application is the design of elementary particle super-colliders. In a super-collider, a group of elementary particles is moving with a speed close to the speed of light, accelerated by the magnets until it hits the target. It is known that this system is unstable, in the sense that under small deviations in the original trajectory and/or in the magnetic fields, the particles will hit the walls of the tunnel way before they reach the target. Interval computations are used to design the colliders in which a way that under

all allowable deviations from the original trajectory and from the magnetic fields, the particles still hit the target.

Interval computations have also been successfully used:

- in industrial robotics, where it is important to make sure that the robot is safe in all possible situations,
- in chemical engineering, where it is important to guarantee that the desired chemical process works well for all allowable concentrations of different chemicals and ores in the input product,
- in nuclear safety, where it is important to guarantee that the reactor remains safe for all allowed values of the parameters,

and in many other practical situations.

Logic (specifically, modal logic) has been efficiently used in interval computations. Many problems of interval computations can be naturally reformulated in terms of *modal logic* – specifically, it terms of the original modal logic of necessity and possibility; see, e.g., [7, 28]. Specifically, in situations like robust control, we want to make sure that the control is stable for all possible values of the parameters from the given intervals, i.e., in terms of modal logic, that it is *necessarily* stable.

On the other hand, a system is controllable if for every state from the desired interval, there exists a control value from the interval of control value that leads to this state. In modal logic terms, this means that the control leading to the desired state is *possible*.

Because of this connection, modal logic has been efficiently used in designing algorithms for interval computations; see, e.g., [5, 8, 18]. There is even a special term *modal interval analysis* for such applications.

From direct to indirect methods of algorithm design: proof mining. Historically, the first existence proofs were *direct* in the sense that they provided an explicit construction. In this sense, the first existence proofs were constructive. Later on, the Greeks invented *indirect* proofs, e.g., proofs by contradiction, that enabled to prove the existence of an object without explicitly constructing such an object. The proofs became non-constructive – but much simpler.

At present, most *constructive* proofs are direct – they provide, in effect, an algorithm for constructing an object. It has been recently shown that in many cases, it is possible to convert *indirect* proofs into constructive ones – namely, for some statements (like the statements of existence and uniqueness) it is often possible to extract a constructive proof from a non-constructive one.

The main idea is that, e.g., if a computable function $f(x)$ has exactly one zero, i.e., exactly one value x_0 for which $f(x_0) = 0$, then we can find this x_0 with increasing accuracy if, for value x_i from the ε-nets $\{x_i\}$ – i.e., from finite sets for which the whole range is a union of their ε-balls $B_\varepsilon(x_i) \stackrel{\text{def}}{=} \{x : \rho(x, x_i) \le \varepsilon\}$ – we compute the minimum $m_i = \min\limits_{x \in B_\varepsilon(x_i)} |f(x)|$ of $|f(x)|$ with sufficient accuracy δ. Due to uniqueness, for sufficiently small ε and δ, all the values m_i will be provably larger than 0 except for points which are close to x_0 – and thus, close to each other. So, when all these points are 2ε-close, this means that x_0 is 2ε-close to all of them – and thus, that each of these points x_i can serve as a 2ε-approximation to x_0.

This possibility of "mining" a non-constructive proof for possible algorithms has been actively used in many areas of computational mathematics; see, e.g., [14]. However, this area of research is only now developing its potential; more applications are potentially possible, more work is needed.

This "proof mining" makes it possible to go beyond the situations in which algorithms are known – and the only problem is how to compute faster – to realistic situations when even an algorithm is not yet known. Thus, we arrive at

Research Direction I.4. *Further develop proof mining, with a special emphasis on its use to develop algorithms for realistic large-scale problems.*

Comment. The idea of proof mining is in line with the general idea of logic as a specification language: once we formulate the original problem in logical terms, and we prove that the problem is *solvable*, the proof mining automatically designs an *algorithm* for solving this problem.

4 Second Approach: Unconventional Computations

Analyzing the second approach: unconventional computations. The main idea of this approach is to use non-standard physical processes to speed up computations.

Quantum computing: successes. The most well-known example of this approach is *quantum computing*, where we can indeed achieve a speedup (see, e.g., [32]); for example:

- quantum computing allows us to search in an un-sorted array of size n in time $\sqrt{n}$ (Grove's algorithm);

- quantum computing allows us to factor large integers in polynomial time (Shor's algorithm).

Limitations of quantum computing and need for other schemes. The main limitation of quantum computing is that so far, there are no provable super-polynomial quantum speed-ups. As a result (unless P=NP), it is not possible to use quantum computing to solve NP-hard problems in polynomial time.

It is therefore desirable to explore other possible schemes that can potentially lead to an exponential speed-up, i.e., that can potentially solve NP-hard problems in polynomial time.

Acausal processes. The simplest example of such a scheme is to use acausal processes, i.e., processes that go back in time and influence the past; see, e.g., [39]. The idea is to spend as much time as needed on computations, and then send the result of the computation back in time, to the moment when the user formulated the problem. Thus, the user will receive the result in no time at all.

The problem with this simple approach is that the actual time travel is known to be paradoxical – e.g., what happens if a time traveler goes to the past and kills his own grandfather before his father was conceived? A reasonable solution to this paradox is that there are always some low-probability events (like a meteorite hitting the Earth at exactly the given spot), so since the time traveler was born, this means that some low probability event prevented the time traveler from this killing. No matter how many cautions the time travel traveler takes, there are always some very low-probability events that cannot be all prevented. So, we arrive at a conclusion that time travel can trigger events with very small probability $p_0 \ll 1$.

Let us show how this conclusion can be used to solve NP-hard problems in polynomial time; for details, see [15, 16, 20, 27, 30]. As an example of an NP-hard problem, we can take the propositional satisfiability problem SAT: given a propositional formula $F(x_1, \ldots, x_n)$, find the values of the propositional variables that make this formula true. Now, the algorithm for solving SAT is as follows:

- generate n bits $x_1, \ldots, x_n$ by using some physically random process,
- check whether the generated bits satisfy the formula $F(x_1, \ldots, x_n)$, and
- if $\neg F(x_1, \ldots, x_n)$, launch the time travel – which is set up in such a way as to generate a very-low-probability event.

For this scheme, nature has two choices:

- generate values x_i that satisfy the formula f; the probability of this is 2^{-n};
- run the time travel and thus, trigger a low-probability event with probability $p_0 \ll 1$.

When $2^{-n} \gg p_0$, the time travel is statistically improbable, so we will generate a sequence that satisfies the formula $F(x)$.

Potential use of curved space-time. Another natural source of speedup is *parallelization*, when several computer work in parallel to perform the same task.

Parallelization does lead to a drastic speedup, but, alas, in Euclidean space, parallelization only leads to a polynomial speed-up; see, e.g., [21, 31]. Indeed, the speed of all the physical processes is bounded by the speed of light c. Thus, in time T, we can only reach computational units at a distance $\leq R = c \cdot T$. The volume $V(R)$ of this area (inside of the sphere of radius $R = c \cdot T$) is proportional to $R^3 \sim T^3$. So, we can use $\leq V/\Delta V \sim T^3$ computational elements, where ΔV is the smallest volume of a single computational element). Hence, we can simulate all these parallel computation on a sequential computer and still get polynomial time.

An interesting fact is that in Lobachevsky space – historically the first curved space – the volume inside a sphere grows exponentially with radius: $V(R) \sim \exp(R)$. According to modern physics, Lobachesvky space is not an adequate description of the physical space, but the same exponential growth of $V(R)$ occurs for some more realistic space-time models. In such space-time models, we can fit exponentially many processors inside the sphere of radius R – and thus get an exponential speedup [21, 31].

Explicit use of Kolmogorov complexity. It is well known that biological processes are often difficult to describe on the level of fundamental physics. To facilitate this description, a Nobelist M. Gell-Mann suggested that physical equations should include terms explicitly depending on complexity [9]. A natural formalization of this complexity is *Kolmogorov complexity* (see, e.g., [24]: the shortest length of a program that generates a given sequence of symbols:

$$K(x) \stackrel{\text{def}}{=} \min\{\text{len}(p) : p \text{ generates } x\}.$$

Under this assumption, by observing physical and biological processes, we can measure the value $K(x)$ [19]. However, it is well known that $K(x)$ is not algorithmically computable [24], and it is also known that the ability to get non-computable values can speed up computations. Thus, Gell-Mann's scheme can indeed potentially speed up computations.

Other schemes using new physical phenomena are based on:

- quantum field theory (G. Kreisel [22]),
- natural idea that every theory is approximate [16, 17], etc.

Unconventional computations and constructive mathematics. All above schemes use or propose a radically *new* physical process.

It is worth noticing that some of the unconventional computation schemes were discovered not by using or proposing a radically new physical process, but rather by a diligent analysis of computability of simple (and seemingly physically reasonable) physical equations such as the wave equation. It turned out that even for the wave equation, there exist computable initial conditions $u(x, 0)$ for which the solution $u(x, T)$ is not computable; see, e.g., [34, 35, 36, 37].

At present, the related research is mainly aimed at analyzing how physical processes can, in principle, "compute" functions which are not computable in the usual sense. From the viewpoint of our main objectives, however, it is desirable to extend this activity to the analysis of what computations can be thus sped up.

Research Direction II.1. *Use constructive mathematics to analyze how the use of physical processes – described by physically meaningful equations – can speed up computations.*

Acknowledgments

This is an expanded version of a paper that Grigori "Grisha" Mints (Stanford University) and myself have prepared for the DARPA Workshop on Unconventional Computing (Stanford, California, March 23–24, 2010). Grisha is, in effect, a co-author of this paper – but of course, I take all the blame for possible shortcomings.

This work was also supported by grant HRD-0734825 from the US National Science Foundation (NSF) and by Grant 1 T36 GM078000-01 from the US National Institutes of Health (NIH).

References

[1] O. Aberth, *Introduction to Precise Numerical Methods*, Academic Press, San Diego, California, 2007.

[2] M. Beeson, *Foundations of Constructive Mathematics: Metamathematical Studies* (Springer, Berlin/Heidelberg/New York, 1985.

[3] M. Beeson, "Some relations between classical and constructive mathematics" *Journal of Symbolic Logic*, **43** (1987) 228–246.

[4] E. Bishop and D. S. Bridges, *Constructive analysis*, Springer-Verlag, Berlin-Heidelberg-New York, 1985.

[5] B. Bouchon-Meunier and V. Kreinovich, "From interval computations to modal mathematics: applications and computational complexity", *ACM SIGSAM Bulletin*, **32**(2) (1998) 7–11.

[6] D. S. Bridges and L. Vîta, *Techniques of Constructive Mathematics*, Springer, New York, 2006.

[7] D. M. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev, *Many-Dimensional Modal Logics: Theory and Applications*, Elsevier, Amsterdam, 2003.

[8] E. Gardeñes, M. A. Sainz, L. Jorba, R. Calm, R. Estela, H. Mielgo, and A. Trepat, "Modal intervals", *Reliable Computing*, **7**(2) (2001) 77–111.

[9] M. Gell-Mann, *The Quark and the Jaguar*, Freeman, New York, 1994.

[10] Interval computations website
http://www.cs.utep.edu/interval-comp

[11] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*, Springer-Verlag, London, 2001.

[12] R. B. Kearfott, *Rigorous Global Search: Continuous Problems*, Kluwer, Dordrecht, 1996.

[13] R. B. Kearfott and V. Kreinovich (eds.), *Applications of Interval Computations*, Kluwer, Dordrecht, 1996.

[14] U. Kohlenbach, *Applied Proof Theory: Proof Interpretations and Their Use in Mathematics*, Springer Verlag, Berlin, 2008.

[15] M. Koshelev, "Maximum entropy and acausal processes: astrophysical applications and challenges", In: G. J. Erickson et al. (eds.), *Maximum Entropy and Bayesian Methods*, Kluwer, Dordrecht, 1998, pp. 253–262.

[16] O. M. Kosheleva and V. Kreinovich, "What can physics give to constructive mathematics, In: *Mathematical Logic and Mathematical Linguistics*, Kalinin, Russia, 1981, pp. 117–128 (in Russian).

[17] O. M. Kosheleva and S. V. Soloviev, "On the logic of using observable events in decision making. In: *Proceedings of the IX National Symposium on Cybernetics,* Moscow, 1981, pp. 49–51 (in Russian).

[18] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1998.

[19] V. Kreinovich and L. Longpré, "Why Kolmogorov complexity in physical equations, *Intl J. of Theor. Physics*, **37** (1998) 2791–2801.

[20] V. Kreinovich and L. Longpré, "Fast quantum algorithms for handling probabilistic and interval uncertainty", *Mathematical Logic Quarterly*, **50**(4/5) (2004) 507–518.

[21] V. Kreinovich and M. Margenstern, "In some curved spaces, one can solve NP-hard problems in polynomial time", *Notes of Mathematical Seminars of St. Petersburg Department of Steklov Institute of Mathematics*, **358** (2008) 224–250; reprinted in *Journal of Mathematical Sciences*, **158**(5) (2009) 727–740.

[22] G. Kreisel, "A notion of mechanistic theory, *Synthese*, **29** (1974) 11–26.

[23] B. A. Kushner, *Lectures on Constructive Mathematical Analysis*, American Mathematical Society, Providence, Rhode Island, 1985.

[24] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer, Berlin, Heidelberg, New York, 2008.

[25] J. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, Berlin, Heidelberg, New York, 1987.

[26] J. McCarthy, *Formalizing Common Sense*, Ablex, Norwood, New Jersey, 1990.

[27] S. Yu. Maslov, *Theory of Deductive Systems and Its Applications*, MIT Press, Cambridge, Massachusetts, 1987.

[28] G. Mints, *A Short Introduction to Modal Logic*, Center for the Study of Language and Information CSLI, Stanford University, Stanford, California, 1992.

[29] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*, SIAM Press, Philadelphia, Pennsylviania, 2009.

[30] H. Moravec, *Time travel and computing*, Carnegie-Mellon Univ., CS Dept. Preprint, 1991.

[31] D. Morgenstein and V. Kreinovich, "Which algorithms are feasible and which are not depends on the geometry of space-time", *Geombinatorics* **4**(3) (1995) 80–97.

[32] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, Massachusetts, 2000.

[33] U. Nilsson and J. Maluszynski, *Logic, Programming, and Prolog*, Wiley, New York, 2000.

[34] M. Pour-El and J. I. Richards, "A computable ordinary differential equation which possesses no computable solution", *Ann. Math. Logic*, **17** (1979) 61–90.

[35] M. Pour-El and J. I. Richards, "The wave equation with computable initial data such that its unique solution is not computable", *Adv. Math.*, **39** (1981) 215–239.

[36] M. Pour-El and J. I. Richards, *Computability in Analysis and Physics*, Springer-Verlag, Berlin, 1989.

[37] M. Pour-El and N. Zhong, "The wave equation with computable initial data whose unique solution is nowhere computable", *Math. Log. Q.*, **43** (1997) 499–509.

[38] S. Rabinovich, *Measurement Errors and Uncertainties: Theory and Practice*, Springer-Verlag, New York, 2005.

[39] K. S. Thorne, *Black Holes and Time Warps: Einstein's Outrageous Legacy*, W. W. Norton, New York and London, 1995.