Fun with Formal Methods for Better Education

Nikolay V. Shilov(✉), Evgeniy Muravev, and Svetlana Shilova

Innoplolis University, 1, Universitetskaya Str., Innopolis 420500, Russia
shiloviis@mail.ru

Abstract. Is there a need to popularize Formal Methods in Software Engineering? Maybe industrial demand in Formal Methods is the best way to explain their utility and importance? – We try to argue educational and emotional role of popularization for a better comprehension and a positive attitude to Formal Methods and discuss several Math Olympiad problems that can be solved using Formal Methods (while, unfortunately, Mathematical education suffers of lack of Theoretical Computer science curricular) .

Keywords: Formal Methods · Popularization · Puzzles · Gamification · Education · Mathematical Olympiad and contests · Ontological argument · Fibonacci words · Operational, denotational and axiomatic semantics · Esoteric languages · Recursion elimination · Algorithm transformation · Partial and total correctness

1 Introduction

1.1 A Semiannual Anniversary

Fifty two years have passed since Robert Floyd had published a paper Assigning Meaning to Programs, a pioneering research on Formal Methods [6], fifty – since C.A.R. Hoare published a paper *An axiomatic basis for computer programming* [10], the first paper on axiomatic of program correctness. During these years people frequently questioned the efficiency, the utility, the industrial strength, the educational value, the understandability of Formal Methods (FM).

For example, in 2010 David Parnas published a very polemical article *Really Rethinking "Formal Methods"* [20]; in particular, he wrote in the article that there are much more FM academic experts than industrial developers using FM, and analyzed the reasons why FM have not became a common practice in Software Engineering.

We believe that this sad picture is not true. Indeed, ACM Turing Prize in 2007 was awarded to Edmund M. Clarke, E. Allen Emerson and Joseph Sifakis for *their role in developing Model-Checking into a highly effective verification technology that is widely adopted in the hardware and software industries* [29]. Later in years 2007-12 model checking was successfully used for verification of on-board software of Mars-rover *Curiosity* [11].

© Springer Nature Switzerland AG 2021
A. Cerone and M. Roggenbach (Eds.): FMFun 2019, CCIS 1301, pp. 92–105, 2021.
https://doi.org/10.1007/978-3-030-71374-4_4

We also think that academic theory and/or industrial practice aren't the only dimensions for Formal Methods, there exists (at least) one more aspect—education. In the next paragraph we explain how we understand education and what role FM may/should play in Software Engineering education.

There exists an opinion (sometimes attributed to Karl Weierstrass) that *the education should bring up minds, not just trains skills.* Also we would like to quote an aphorism (commonly attributed to Mikhailo Lomonosov) that *mathematics should be learned just because it disciplines minds.* By citing these maxims we wouldn't like to claim that the purpose of education is bringing up minds, or that the utility of Mathematics is restricted by mind discipline. We just would like to emphasize a value of Formal Methods for Software Engineering education: to bring up and discipline minds of the future engineers and developers.

1.2 Fun for Better Education

A part of the reason why the Formal Methods moves slowly from the academy to industry is FM education, a "transmission": *some students consider FM too poor (inefficient), other students consider FM too pure.* We need to improve transmission, i.e. to improve FM education.

Ascending approach (from simple and easy to the most complex and complicated) is a common practice in education. Nobody starts teaching arithmetic with Peano axioms and formal derivation of formal statements like (for instance) addition associativity $\forall x.\forall y.\forall z \;:\; ((x+y)+z) = (x+(y+z))$; instead the education/teaching starts with elementary exercises/problems like the following: *Dad gave Peter 5 apples and then Peter passes 2 apples to a sister; how many apples Peter has after that?*

If educator would like to engage students with a topic then fun and amusement may be very important and helpful ingredients (maybe, a lubricating oil for the transmission). Even a simple joke can help! (For example, if you think that the answer for the above problem about apples is 3 then you are not right, because the correct answer depends on initial number of apples that Peter had before his Dad gave him these 5 apples;-)

Same should be true for FM education: it should start with simple and easy examples/problems, exploit jokes, fun and amusement for engagement and popularization. Many FM educators use ascending approach altogether with fun and amusement in their educational practice. – Just for example, a very concise, sound and comprehensive textbook [12] on model checking with SPIN is illustrated by many puzzles solved by model checking. (Of course, a renowned *Cabbage, Goat and Wolf* is one of the puzzles used in this book.) But we question is: how common is this practice to engage students with FM via fun, puzzles, games and entertainment?

1.3 FWFM Workshop Series

The primary purpose of the workshop series on *Fun With Formal Methods* (FWFM) was (and still is) to popularize and disseminate the best practice of popularization of Formal Methods. Not an exhaustive list of topics of FWFM follows:

- fascinating examples of use of FM in SE;
- simple but interesting educational examples of FM;
- FM for puzzles, games and entertainment;
- FM and programming contests and Olympiads;
- FM elsewhere (outside software and hardware);
- anything and everything related to popularization of FM.

History of the workshop is depicted in brief in the Fig. 1 and explained below in the next paragraphs.

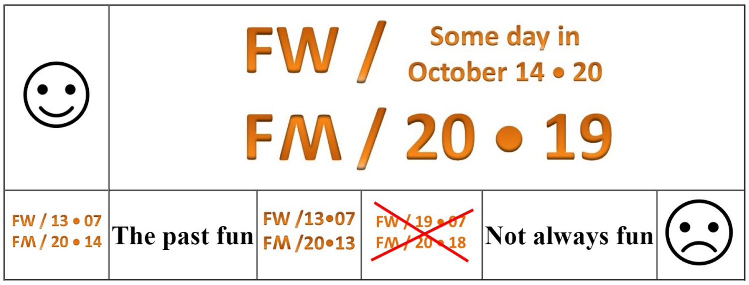

Fig. 1. History of the FWFM workshop series.

The workshops from FWFM series ware organized twice in the years 2013 [30] and 2014 [31] but (for fun!) the same day both times – July 13 – and both times— in affiliation with *the International Conference on Computer Aided Verification* (CAV). Both workshops were successful because of number of submissions, good quality of selected papers and a high attendance.

Then there was non-successful attempt to organize the third workshop in the year 2018 [32]. The attempt had the same affiliation (CAV), but was scheduled for another day then two previous workshops — July 19 instead of July 13. This shift was not fun, it has led to few submissions and cancellation of the workshop. – Of course, we are kidding about the role of the day and its influence on the number of submissions; maybe the main reason of the deficit of interest to the FWFM-2018 was publication policy of the first two workshops: no formal proceedings of the FWFM-2013 and FWFM-2014 have been published.

After an epic failure in 2018, we attempted to revive FWFM series in year 2019 [33]. Because of this intention we had decided to give up fixed affiliation (CAV), fixed day (July 13), and presentation "in person" (offline) policy. (By the way, this *online* move has been implemented months before the outbreak of COVID-19 [34] and hadn't been motivated by the epidemic situation. Maybe, it (the move) had much more in common with climate change concern recently stressed in [28].) In the year 2019 the workshop was organized as a satellite event of another conference *TOOLS 50+1: Technology of Object-Oriented Languages and Systems*, distance (online) presentations (via Skype) were allowed, and live streaming video of the workshop was organized and recorded [35]. – Maybe it is too early to say that the workshop was successful, but it is right time to say that we had the workshop and the FWFM series is alive!

The problem that FWFM-series needs to solve is a proper publication of the workshop proceedings. If not to solve but to compensate a publication deficit,we would like to present in brief in the next section the following 5 talks from FWFM-2013 and FWFM-2014:

– *The Ontological Argument in PVS: What Does This Really Prove?* (John Rushby, FWFM-2013);
– *Tackling Fibonacci words puzzles by finite countermodels* (Alexei Lisitsa, FWFM-2013);
– *Teaching Formal Methods using Magic Tricks* (Paul Curzon and Peter McOwan, FWFM-2013);
– *Chekofv: Crowd-sourced Formal Verification* (Heather Logas et al., FWFM-2014);
– *Using Esoteric Language for Teaching Formal Semantics* (Nikolay Shilov, FWFM-2014).

1.4 Structure of the Paper

In the next Sect. 2 we give a brief overview of selected talks from FWFM-2013 and FWFM-2014. Then in the Sect. 3 we just sketch the programme of FWFM-2019 but present (some) details solutions (with aid of Formal Methods) of two problems from the 60th International Mathematical Olympiad IMO-2019. Finally, we conclude in the last Sect. 4 by summing-up our arguments for popularization of Formal Methods and discussing challenges of further integration of Formal Methods and Artificial Intelligence – Computer Science in general – with Mathematics.

2 FWFM13-14 in Brief: From Ontological Argument to Esoteric Languages

2.1 The Ontological Argument in PVS

An *ontological argument* is a tradition to prove that God existence using ontology. One of known ontological arguments was formulated by Anselm of Canterbury in 1078 in work Proslogion. Anselm defined God as “that than which nothing greater can be thought”. He suggested that, if the greatest possible exists in the mind, it must also exists in reality and proved it by contradiction: if the greatest possible does exist just in the mind, then an even greater must exists in the mind—one which is greater both in the mind and in reality; therefore, this the greatest possible being must do exist in reality.

Please refer [21] for a formalization and verification of the Ontological Argument in PVS [36]. Although the formalization is consistent, the formal verification doesn't compel the Ontological Argument. The educational value of the formalization and verification of the Ontological Argument with PVS is an opportunity to use it as a case study in graduate programs at Philosophy and Humanities Departments for teaching automated theorem proving.

2.2 Countermodels for Fibonacci Words

An infinite sequence of Fibonacci words w_0, w_1, ... is defined [17] very similar as the infamous sequence of Fibonacci numbers: let a and b be two distinguishable symbols; then $w_0 = b$, $w_1 = a$, and $w_{i+2} = w_i w_{i+1}$ for all $i \geq 0$. It is easy to see that the sequence of Fibonacci words stars as *b*; *a*; *ba*; *aba*; *baaba*; *ababaaba*; *baabaababaaba*.

One can observe that none of the first 7 Fibonacci words listed above contains two *b*'s or three *a*'s in a row (i.e. no sub-words *bb* or *aaa*). This observation leads to a hypothesis that all Fibonacci words contain neither two *b*'s nor three *a*'s in a row. But then the next question arises: how to prove (ore refute) the hypothesis?

A particular way to prove the hypothesis presented in [17] comprises two steps:

1. first-order sound axiomatization of algebraic systems (first-order models) where all elements of the domain may be generated using Fibonacci words
2. and then automatic generation of finite countermodels that meet the axiomatization but refute that some element may be generated using two *b*'s or three *a*'s in a row.

Surprisingly, the countermodels for each of these properties are quite small,—just 5 elements to refute a possibility of two *b*'s and 11 element to refute a possibility three *a*'s in a row [17].

The educational value of the case-study is popularization of non-standard models for proving properties of the standard ("default" or assumed) models and tools like finite model generators for first-order theories.

2.3 Learning Loop Invariant via Card Magic

After great publications by Martin Gardner like *Mathematics, Magic, and Mystery* (1956) or *Mathematical Puzzles* (1961), it is hard to engage magic tricks with any other discipline than Mathematics. But still many magic tricks are much more dynamic and algorithmic in nature than static and Mathematical. Hence many magic tricks can/may be used to teach Computer Science and Formal Methods.

Some examples of engagement of card magic with CS and FM can be found in paper [1] that summarizes some experience accumulated in the science-popular project *cs4fn* (Computer Science for Fun) [37]. Below we present in brief one example of a card magic (borrowed from [1]) and discuss educational value of the example.

1. Take 10 cards consisting of a series of 5 cards of a suit followed by the same 5 cards of a different suit placed in the same order.
2. Fan the cards to show that you have a mix of cards and then turn the pack over, face down and ask a volunteer to touch the back of any card. Cut the pack at this point, putting the top half to the bottom and fan the cards again. Repeat this several times until the audience becomes happy that the cards are sufficiently mixed.
3. Count out 5 cards into a pile on the table, reversing their order as you do so. Place the remaining 5 cards straight down to make a second pile (non-reversed).
4. Give a volunteer 4 coins and ask to put each down on one of the two piles (i.e. the volunteer may spread coins between the two piles arbitrary). Once coins are placed you now do the same number of moves on a pile as the number of coins on the pile. (A move consists just of moving a card from the top to the bottom of the pile.) Then take the resulting top card of each pile and place them together (face down) at the side together with one coin (from 4 that you use). Repeat the same with the remaining 3 (instead of 4) coins and remaining two piles (each with 4 instead of 5 cards), then – with 2 coins and piles with 3 cards each, and finally – with the last coin and piles with 2 cards each.
5. Turn over all pairs of cards and demonstrate to the audience that cards in pairs match each other!

The correctness of the magic trick can be explained in terms of partial correctness of the non-deterministic algorithm presented above using pre-conditions, invariants and post-conditions:

– Precondition of the first non-deterministic loop on step 2 is formulated in step 1: *the first 5 cards of one suit are followed by the same 5 cards of another suit in the same order.*
– The invariant and the post-condition of the first loop is very similar to the pre-condition: *the first 5 cards are followed by the same 5 cards in the same order.*
– Pre-condition for the loop on steps 4 results from post-condition for step 2 after implementing step 3: *the order of 5 cards in the first pile is reverse of the order of the 5 cards in the second pile.*
– The invariant of the second loop is closely related to the pre-condition: *the order of cards in the first pile is reverse of the order of the cards in the second pile and cards in pairs that are put aside match each other.*
– The post-condition is what we want to demonstrate to the audience: *cards in pairs match each other.*

So, Formal Method's classics is a magic!

2.4 Gamification and Crowd-Sourcing Loop Invariants

Chekofv [18,19] is a system for crowd-sourced formal verification. It starts with an attempt to verify a given C program using the source code analysis platform

Frama-C. Every time the analysis needs a loop invariant (like in the previous subsection) *Chekofv* translates the problem into a puzzle game *Xylem* and presents it to players.

Xylem [19] is an iPad game where players make mathematical observations about synthetic plants, which are turned into predicates used for the construction of loop invariants. The game is a logical induction puzzle game where the player plays a botanist exploring and discovering new forms of plant life on a mysterious island. The player observes patterns in the way a plant grows, and then constructs mathematical equations to express the observations. These equations are considered as candidates for loop invariants and must be verified by any proof-assistance (PVS in particular).

2.5 Formal Semantics Though an Esoteric Language

Teaching different types of formal semantics (at undergraduate level especially) is not a trivial task. A gentleman's set should include some variants of operational, denotational and axiomatic semantics. A common approach to teaching the topics consists in use of toy programming languages. Instead, [23,24] presented an approach with use of an esoteric language [38].

Every language (artificial or natural) may be characterized by its syntax, semantics, and pragmatics. For example, in one of the 56 Sherlock Holmes short stories, *The Adventure of the Dancing Men*, written by Arthur Conan Doyle, Mr. Hilton Cubitt gives Sherlock Holmes a piece of paper with this mysterious sequence of stick figures of dancing men that had driven driving his young wife Elsie to distraction. Holmes realizes that it is a substitution cipher, cracks the code by frequency analysis and realizes that the syntax was just as in English with dancing men instead of letters, the semantics was provided by transformation to English, pragmatics (usage) of the language was to serve as a cryptography for Chicago gangsters.

Toy Esoteric Language (TEL) is not a programming language at all since it is not design for data processing. Its pragmatics is to introduce and explain what different types of formal semantics are, namely: what are operational, denotational, axiomatic, second-order and game semantics and how they may relate to each other. TEL syntax is easy to explain: correct words look like bodies of structured Pascal programs (with integer variables exclusively). TEL informal semantics can be defined as follows. Since every correct TEL word looks like an iterative program, one can draw a flowchart of this program. Every flowchart is a graph with assignments and conditions as nodes and control passing as edges. Let us count length of a path between nodes in a flowchart by number of assignments in this path (i.e. we do not count conditions at all). Then semantics of a correct TEL "program" is the shortest length of a path through the corresponding flowchart (i.e. from start to finish).

3 How FM and Math Can Help and Boost Each-Other?

3.1 FWFM-2019 in Brief

The programme of FWFM-2019 [33,35] comprises the following 5 talks:

1. *Do we need Fun with Formal Methods?* (Nikolay Shilov and Evgeiy Muravev);
2. *Cables, Trains and Types* (Simon Gay);
3. *On programming content of Math contests* (Nikolay Shilov and Svetlana Shilova);
4. *Towards a Broader Acceptance of Formal Varication Tools* (Mansur Khazeev et al.);
5. *Fun with Formal Methods: teaching unambiguous English to avoid confusions* (Maya Stoyanova).

The first talk presented a tool to play with the axiomatic semantics for the esoteric language TEL from papers [23,24].

The second talk presented a dependent type system for cables and toy railroad, a background paper is available [8] and is under publication right now.

The third talk addressed relations between Mathematics and Theory of Programming, its content has not been published anywhere else and because of it is discussed in the following subsections of this paper.

The forth talk presented results of in-class study how a small group of master students in Software Engineering accept formal verification tools in general and *AutoProof* system [7] in particular, a background paper has been published in arXiv [13].

The last talk was English-language experience report how to teach future software developers and engineers to speak and write in unambiguous way (especially when it concerns technical writing and specifications in English).

The workshop FWFM-2019 was concluded by a discussion moderated by Hamna Aslam. The topics of the discussion included (but were not limited by) the following questions:

- Who should not be teaching Math & FM?
- Which Math & FM book(s) are not recommended to be proposed as a textbook for freshmen?
- How to promote Math & FM group learning among students?
- How to teach Math & FM to students in their language?

(The purpose of the "negative" questions wasn't to exclude someone or some book as but to learn student's opinions about a "good" and a "bad" education practice.)

3.2 On Relations Between Program Theory and Mathematics

A discourse about historical, cultural, educational relations and connections between Mathematics and Science and Art of Programming (exactly Programming not Computer Science) is quite old: it originated in early days of computing machinery more than 70 years ago (since, at least, since ENIAC was completed and first put to work in 1945). Many programming pioneers—e.g. Edsger W. Dijkstra, Andrey P. Ershov, Donald E. Knuth—had published their reflections on this topic [2–4,14]. (Unfortunately, we are not aware about reflections of mathematicians on this topic while we know and highly recommend a book of outstanding Russian mathematician Vladimir A. Uspensky [27] where he had promoted and advocated a view on Mathematics as a humanitarian science.)

In the talk *On programming content of Math contests* we drew attention to the importance of introduction of programming art and science [5,9,15,16] to mathematical education not just because of industrial demand and/or employment opportunities for graduates but because of a need of programming culture for solving mathematical problems. We would like to advocate this claim by analysis of the problem set [40] of the most recent International Mathematical Olympiad [39] (which was the 60th in the series).

The Olympiad set [40] comprises 6 problems from which 1.5 (exactly *one and a half*) are good examples to demonstrate programming art and science. Namely, we speak about the following problems from the set.

[Problem IMO-19-1]. Let $\mathbb{Z}$ be the set of integers. Determine all functions $f : \mathbb{Z} \to \mathbb{Z}$ such that, for all integers a and b, $f(2a) + 2f(b) = f(f(a + b))$.

[Problem IMO-19-5]. The Bank of Bath issues coins with an H on one side and a T on the other. Harry has n of these coins arranged in a line from left to right. He repeatedly performs the following operation: if there are exactly $k > 0$ coins showing H, then he turns over the kth coin from the left; otherwise, all coins show T and he stops. For example, if $n = 3$ the process starting with the configuration THT would be $THT \to HHT \to HTT \to TTT$, which stops after three operations.

a) Show that, for each initial configuration, Harry stops after a finite number of operations.

b) For each initial configuration C, let $L(C)$ be the number of operations before Harry stops. For example, $L(THT) = 3$ and $L(TTT) = 0$. Determine the average value of $L(C)$ over all 2^n possible initial configurations.

The problem IMO-19-1 can serve as an example of recursion elimination [15,22] using reduction of a monadic recursion to a tail recursion, we discuss this programming technique and its application to the problem IMO-19-1 in the next subsection. (A pure mathematical solution can be found at [40] in the original *Problems (with solutions)* provided by the 60th International Mathematical Olympiad, and watched at [41] among other Math videos by Presh Talwalkar.)

The problem IMO-19-5(a) is a "typical" problem on algorithm termination to be solved by Floyd method [9,25] (but this time some preliminary equivalent algorithm transformations are required), we present a programming solution of the problem in the subsection after the next one.

3.3 Problem IMO-19-1 via Recursion Elimination

A classic example monadic recursion elimination by reduction to the tail recursion is a so-called John McCarthy function $M_{91} : \mathbb{N} \to \mathbb{N}$ [15,22] that is defined as follows below:

$$M_{91}(n) = \; if \; n > 100 \; then \; (n - 10) \; else \; M_{91} \; (M_{91}(n + 11)).$$

It turns out that $M_{91}(n) = \; if \; n > 101 \; then \; (n - 10) \; else \; 91$. A key idea in recursion elimination is move from a monadic function $M_{91} : \mathbb{N} \to \mathbb{N}$ to a binary function $M2 : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $M2(n, k) = (M_{91})^k(n)$ for all $n, k \in \mathbb{N}$ (where $(M_{91})^k(n)$ is k-time application of the function, i.e. $(M_{91})^k(n) = \overbrace{M_{91}\,(\dots M_{91}(n)\dots)}^{k}$); of course, $M2(n, 0) = (M_{91})^0(n) = n$ for every $n \in \mathbb{N}$.

Let us apply the idea presented in the previous paragraph to the problem 1. Since $f(2a) + 2f(b) = f\,(f(a + b))$ is true for all $a, b \in \mathbb{Z}$, then $f(0) + 2f(b) = f(f(b))$ for all $b \in \mathbb{Z}$. Let us define a binary function $F : \mathbb{Z} \times \mathbb{N} \to \mathbb{Z}$ such that $F(b, k) = f^k(b)$ and $F(b, 0) = f^0(b) = b$ for all $a \in \mathbb{Z}$ and $k \in \mathbb{N}$. Then for all $a \in \mathbb{Z}$ and $k \in \mathbb{N}$

$$\begin{aligned} F\,(b, (k + 1)) &= 2F(b, k) + f(0) = 2\,(2F\,(b, (k - 1)) + f(0)) + f(0) = \dots \\ &= 2^{(k+1)}F(b, 0) + (2^{(k+1)} - 1)f(0) = 2^{(k+1)}b + (2^{(k+1)} - 1)f(0), \end{aligned}$$

and, hence, $f(b) = f^1(b) = F(b, 1) = 2b + f(0)$ and thus the problem 1 is solved!

3.4 Problem IMO-19-5(a) via Proving Algorithm Termination

Let us start with the following formalization (in pseudo-code) of the algorithm specified in the problem statement 5:

```
var W: a word in the alphabet {T,H};
var k: a natural number;
while H exists in W
do k:= number of H in W;
   if W[k] = T then W[k]:= H else W[k]:= T
od
```

Because of the loop condition `while H exists in W`, the only thing we need to prove is the loop termination.

For this purpose, let us transform the above algorithm as follows:

```
var W: a word in the alphabet {T,H};
var k, i: natural numbers;
while H exists in W
do k:= number of H instances in W;
   i:= k;
   while W[i] = T
```

```
        do W[i]:= H; i:= i+1 od;
   while W[i] = H
        do W[i]:= T; i:= i-1 od;
od
```

This transformed algorithm is equivalent to the original one because it first serializes conversions of T to H and then serializes conversions of H to T. Remark that the conjunction of the following three clauses

- the number of H in W is i;
- $k \leq i \leq$ the index of the rightmost instance of H in W;
- W[k..(i-1)] consists of H only (i.e. hasn't any instance of T)

forms an invariant [9] of each of both internal loops (i.e. if the conjunction is true before any exercise of a loop body then it remains true after the exercise). It implies that for each legal iteration of the external loop (i.e. when W has any instance of H)

the number of instances of H in W
before the loop body exercise
(that is the value of k)
is positive and greater than
the number of instances of H in W
after the loop body exercise
(that is the final value of i);

in other words, the number of instances of H in W decreases on each legal iteration of the external loop. Hence, the number of instances of H in W is the loop variance and (according to Floyd method of proving termination [9]) the transformed algorithm as well as the original one always terminates.

4 Conclusion: What Else and Next?

Fun, jokes, puzzles, games and entertainment in teaching is not the unique ingredient needed to improve Formal Method education (more general — Computer Science and Software Engineering education). All these should be used to engage (undergraduate) students with learning/study/comprehension/mastering Formal Methods. We believe that experience of individual educators and expertise of research groups in the field of Formal Method popularization deserves a positive attitude from Computer Science and Software Engineering academic community and industry.

Another opportunity to engage students is a competitive spirit that is so appropriate to young people (in particular — to students of CS and SE departments). International competitions between FM tools (e.g. automated theorem provers and satisfiability solvers) are popular, useful and valuable from industrial and research perspectives, but not from undergraduate education perspective. Unfortunately, competitions especially designed for (undergraduate) students (like Collegiate Programming Contest [42]) are still not involved into education process in general and in FM education in particular. We hope that competitions of this kind may be used better for engaging students with Theory of Computer Science and Formal Methods in Software Engineering [26].

We would like to conclude by drawing attention to a so-called *IMO Grad Challenge* [43]:

> *The International Mathematical Olympiad (IMO) is perhaps the most celebrated mental competition in the world and as such is among the ultimate grand challenges for Artificial Intelligence (AI).*
> *The challenge: build an AI that can win a gold medal in the* ***competition****.*
> *To remove ambiguity about the scoring rules, we propose the formal-to-formal (F2F) variant of the IMO: the AI receives a formal representation of the problem (in the Lean Theorem Prover), and is required to emit a formal (i.e. machine-checkable) proof. We are working on a proposal for encoding IMO problems in Lean and will seek broad consensus on the protocol.*
> ...
> *Challenge. The grand challenge is to develop an AI that earns enough points in the F2F version of the IMO (described above) that, if it were a human competitor, it would have earned a gold medal.*

So, it is a high time for mathematicians not only to learn the art and the science of programming, but technologies and tools of the Artificial Intelligence!

References

1. Curzon, P., McOwan, P.: Teaching formal methods using magic tricks. In: Contributed talk at the CAV Workshop Fun With Formal Methods, St.Petersburg, Russia, 13 July 2013. http://www.chi-med.ac.uk/publicdocs/WP122.pdf. Accessed 20 Jan 2020
2. Dijkstra, E.W.: On a cultural gap. Math. Intell. **8**(1), 48–52 (1986). https://doi.org/10.1007/BF03023921
3. Ershov, A.P.: Aesthetics and the human factor in programming. Commun. ACM **15**(7), 501–505 (1972)
4. Ershov, A.P.: Programming as the second literacy (1980) (In Russian). http://ershov.iis.nsk.su/ru/second_literacy/article. Accessed 20 Jan 2020
5. Ershov, A.P., Knuth, D.E. (eds.): Algorithms in Modern Mathematics and Computer Science. LNCS, vol. 122. Springer, Heidelberg (1981). https://doi.org/10.1007/3-540-11157-3
6. Floyd, R.W.: Assigning Meaning to Programs. In: Proceedings of Symposium on Applied Mathematics, vol 19, pp. 19–32. Amer. Math. Soc. (1967)
7. Furia, C.A., Nordio, M., Polikarpova, N., Tschannen, J.: AutoProof: auto-active functional verification of object-oriented programs. Int. J. Softw. Tools Technol. Transfer **19**(6) 697–716 (2017)
8. Gay, S.J.: Cables, trains and types. In: Chris Hankin's Festschrift (to appear). http://www.dcs.gla.ac.uk/~simon/publications/CablesTrainsTypes.pdf. Accessed 20 Jan 2020

9. Gries, D.: The Science of Programming. Monographs in Computer Science. Springer-Verlag, New York (1981). https://doi.org/10.1007/978-1-4612-5983-1
10. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (1969)
11. Holzmann, G.J.: Mars code. Commun. ACM **57**(2), 64–73 (2014)
12. Karpov, Y.G.: Model checking: verification of concurrent and distributed systems. BHV-Petersburg (2010) (In Russian)
13. Khazeev, M., Mazzara, M., De Carvalho, D., Aslam, H.: Towards a broader acceptance of formal verification tools: the role of education. arXiv:1906.01430 [cs.SE]. https://arxiv.org/abs/1906.01430. Accessed 20 Jan 2020
14. Knuth, D.E.: Computer science and its relation to mathematics. Am. Math. Mon. **81**(4), 323–343 (1974)
15. Knuth, D.E.: Textbook Examples of Recursion. https://arxiv.org/pdf/cs/9301113.pdf (1991). Accessed 20 Jan 2020
16. Knuth, D.E.: The Art of Computer Programming, Volumes 1–3 Boxed Set, 2nd edn. Addison-Wesley, Reading (1998)
17. Lisitsa, A.: Tackling Fibonacci words puzzles by finite countermodels. In: Contributed talk at the CAV Workshop Fun With Formal Methods, St.Petersburg, Russia, 13 July 2013. http://cgi.csc.liv.ac.uk/ alexei/Fibonacci_Challenge/fun2013.pdf. Accessed 20 Jan 2020
18. Logas, H., Kirchner, F., Murray, J., Schaf, M., Whitehead, E.J. (Jr.): Chekofv: crowd-sourced formal verification. In: Contributed talk at the CAV Workshop Fun With Formal Methods, Vienna, Austria, 13 July 2014
19. Murray, J., Whitehead, J., Kirchner, F.: Crowd-sourced help with emergent knowledge for optimized formal verification (CHEKOFV). SRI INTERNATIONAL, March 2016, FINAL TECHNICAL REPORT. https://users.soe.ucsc.edu/~ejw/papers/Chekofv%20Final%20Report%20Part%20A.pdf. Accessed 20 Jan 2020
20. Parnas, D.L.: Really rethinking "Formal Methods". IEEE Comput. **43**(1), 28–34 (2010)
21. Rushby, J.: The ontological argument in PVS. In: Invited talk at the CAV Workshop Fun With Formal Methods, St.Petersburg, Russia, 13 July 2013. http://www.csl.sri.com/users/rushby/papers/ontological.pdf. Accessed 20 Jan 2020
22. Shilov, N.V.: Etude on recursion elimination. Model. Anal. Inf. Syst. **25**(5), 549–560 (2018)
23. Shilov, N.V.: Using esoteric language for teaching formal semantics. Contributed talk at the CAV Worthe same languagekshop Fun With Formal Methods, Vienna, Austria, 13 July 2014
24. Shilov, N.V.: Make formal semantics popular and useful. Bull. Novosibirsk Comput. Center Ser. Comput. Sci. IIS Special Issue **32**, 107–126 (2011)
25. Shilov, N.V., Shilova, S.O.: On mathematical contents of computer science contests. In: Enhancing University Mathematics: Proceedings of the First KAIST International Symposium on Teaching. American Society, CBMS Issues in Mathematics Education, vol. 14, 193–204 (2007)
26. Shilov, N.V., Yi, K.: Engaging students with theory through ACM collegiate programming contests. Commun. ACM **45**(9) (2002)
27. Uspensky, A.V.: Mathematics Apology. Amphora, Sant-Petersburg (2009) (In Russian)
28. Vardi, M.Y.: Publish and Perish. Commun. ACM 63(1), 7 (2020)

29. A.M. Turing Award Winners. http://amturing.acm.org/award_winners/clarke_1167964.cfm, http://amturing.acm.org/award_winners/emerson_1671460.cfm. http://amturing.acm.org/award_winners/sifakis_1701095.cfm. Accessed 20 Jan 2020
30. Fun With Formal Methods (2013). http://www.iis.nsk.su/fwfm2013. Accessed 20 Jan 2020
31. Fun With Formal Methods (2014). http://www.easychair.org/smart-program/VSL2014/FWFM-cfp.html. Accessed 20 Jan 2020
32. Fun With Formal Methods (2018). https://persons.iis.nsk.su/en/FWFM2018. Accessed 20 Jan 2020
33. Fun With Formal Methods (2019). https://persons.iis.nsk.su/en/FWFM19. Accessed 20 Jan 2020
34. COVID-19 pandemic. https://en.wikipedia.org/wiki/COVID-19_pandemic. Accessed 15 June 2020
35. Tools 50+1 conference. Day 3. Fun With Formal Method Workshop. https://www.youtube.com/watch?v=QqLRUWD9Ngg. Accessed 20 Jan 2020
36. PVS specification and verification system. https://github.com/SRI-CSL/PVS/. Accessed 20 Jan 2020
37. CS4F. www.cs4fn.org. Accessed 20 Jan 2020
38. Esoteric Programming Languages. https://en.wikipedia.org/wiki/Esoteric_programming_language. Accessed 20 Jan 2020
39. International Mathematical Olympiad. https://www.imo-official.org/default.aspx. Accessed 20 Jan 2020
40. Problems (with solutions). In: 60th International Mathematical Olympiad. Bath - UK, 11th-22nd July 2019. https://www.imo2019.uk/wp-content/uploads/2018/07/solutions-r856.pdf. Accessed 20 Jan 2020
41. Solving An Insanely Hard Problem For High School Students. MindYourDecisions - Math videos by Presh Talwalkar. https://www.youtube.com/watch?v=uJqbHaFqjmI. Accessed 15 June 2020
42. ICPC. International Colegiate Programming Contest. https://icpc.baylor.edu/. Accessed 20 Jan 2020
43. IMO Grand Challenge. https://imo-grand-challenge.github.io/. Accessed 20 Jan 2020