ENGAGING STUDENTS WITH THEORY THROUGH ACM COLLEGIATE PROGRAMMING CONTESTS

How formal methods can be presented in a popular – but mathematically sound – manner to undergraduate students of various disciplines.

The role of formal methods in the development of computer hardware and software increases as systems become more complex and require more extensive efforts for their specification, design, implementation, and verification. Simultaneously, formal methods become increasingly complicated because they must capture actual properties of real systems for sound reasoning.

A comprehensive survey of formal methods and associated discussion of the industrial demand and theoretical supply in this domain is beyond the scope of this article.[1] Instead, we discuss some problems and experiences concerning how to make the mathematical foundations of formal methods more popular with undergraduate students of departments influencing the further development and progress of computer hardware and software. Such departments include not only computer science but also pure and/or applied mathematics, electrical and/or electronic engineering, and information and/or telecommunications technology. We consider the need for such popularization as urgent, since in spite of the importance of formal methods for development of reliable hardware and software this domain is not familiar to non-professionals. In particular, undergraduate students of mathematics, engineering, and technology departments consider formal methods outside of their interests, since they are either too poor for their pure mathematics, or too pure for their poor mathematics. We are especially concerned by this disappointing attitude and conclude that a deficit of popular lectures, tutorials, and papers on this topic is the main reason for this ignorance.

Here, we intend to inspire readers from academia and industry to consider the importance of a popular presentation of theoretical research for better computer science education. Our experiences popularizing program logics via training undergraduates for ACM Collegiate Programming Contests motivate and illustrate our discussion here.

Attitudes Toward Popularization of Theory

A negative attitude concerning computer science theory seems to be prevalent among university students. But the general attitude of computer science theorists toward popularization of their research with undergraduates is not much better. It is useful

[1] Perhaps the simplest way to learn more about the scope and range of formal methods research and its industrial-strength applications is to visit the Web sites administered by Oxford University (archive.comlab.ox.ac.uk/formal-methods.html) and NASA (shemesh.larc.nasa.gov/fm/).

BY NIKOLAY V. SHILOV AND KWANGKEUN YI

to compare these attitudes with those of mathematicians toward popularization of their research.

There are several well-known national and international journals on popular mathematics oriented to a wide community of students and researchers; two examples include Springer-Verlag's *The Mathematical Intelligencer* and the Mathematical Association of America's *The American Mathematical Monthly*. Both journals are published monthly, and according to the Science Citation Index from the Institute for Scientific Information (www.isinet.com), the average citation frequency of the articles in those journals is approximately 0.3 times a year. Moreover, mathematicians have become more concerned about the popularity of mathematics among students. In particular, SIAM recently launched a special section on education in the SIAM Review (SIREV). As is stated in the *Guidelines for SIREV Authors* [5], "In the large majority of cases, articles should be written for students, not to faculty. Articles should provide descriptions, illustrations, and insights regarding established or recent knowledge, as opposed to new research results."

In contrast, a comprehensive list of periodicals related to computer science education [1] does not contain any publication primarily oriented toward popularization of computer science theory. Theoretical research journals do not support popularization. It is possible the *SIGCSE Bulletin* is the unique serial edition that publishes material related to experience with popularizing computer science theories. But it is published quarterly only, is not covered by the ISI's Science Citation Index, and its articles' average citation frequency is not available. Overall, computer science theorists in comparison with mathematicians give at least

$$\frac{\text{2 journals x 12 issues each}}{\text{1 bulletin x 4 issues}} = 6$$

times less attention to popularizing their research.

Attitudes concerning student contests and olympiads are another essential difference between mathematicians and computer science theorists. For example, the ACM Collegiate Programming Contest Finals are sponsored by IBM, not by any special interest group or research or education institution, although this popular contest is an excellent opportunity to engage gifted students with computer science theory. Moreover, it appears the computer science community considered contests of this kind outside the realm of educational activities. The Finals of the 26th ACM contest took place this past March in Hawaii, while one of major annual events related to computer science education—the 33d SIGCSE Technical Symposium in Computer Science Education—took place three weeks earlier in Kentucky. Both events did not provide links to each other. This situation contrasts an attitude toward mathematics contests. For example, the Korean Mathematical Society (KMS) granted equal priority to two major mathematical events hosted by Korean mathematicians in 2000: the 41st International Mathematical Olympiad and the KMS'2000 International Conference "Mathematics in New Millennium." This "regional" example reflects a general situation: mathematicians are seriously concerned by the impact of contests.

These arguments can incorrectly lead to a conclusion that an attitude toward popular presentation of computer science theory is completely negative. To counter this sentiment, we point out to readers the existence of, for example, the International Summer Schools in Marktoberdorf (www4.in.tum.de/div/summerschool) and the European Summer Schools in Logic, Language, and (www.cs.bham.ac.uk/~esslli/). But let us also remark that the enrollment of these schools is comparatively small (several hundred students per year) and consists of graduate or postgraduate students, junior scientists, or even professors. In contrast, ACM programming contests are aimed at undergraduates and embrace thousands of students (approximately 10,000 this year).

TRAINING SESSIONS ARE GOOD OPPORTUNITIES TO PRESENT STUDENTS WITH CHALLENGING PROGRAMMING PROBLEMS THAT CANNOT BE SOLVED WITHOUT THEORETICAL BACKGROUND IN SPITE OF SIMPLE FORMULATION. THE TRAINERS SHOULD PROVIDE STUDENTS WITH BACKGROUND THEORY AS SOON AS STUDENTS REALIZE THE PROGRAMMING COMPLEXITY OF THESE PROBLEMS.

A natural question arises: how to engage undergraduates with theory through programming contests? The first possibility, of course, is during training sessions. Training sessions are good opportunities to present students with challenging programming problems that cannot be solved without theoretical background in spite of simple formulation. The trainers should provide students with background theory as soon as students realize the programming complexity of these problems. We would like to sketch here a brief example of how a program logic's tributary creek of a powerful stream called formal methods can be presented in a popular (but mathematically sound) form to undergraduate students of computer science, mathematics, physics, and technical departments.

Before presenting our experiences we discuss the second opportunity for increasing the influence of ACM programming contests on better computer science education. We believe organizers of regional- and international-level contests should provide (after competition) comprehensive Web-based tutorials on how to solve selected problems with background theory and sample programs. The importance of these tutorials is key.

How to Make Program Logics Easy

Here, we sketch an informal framework in which a formal theory of program logics is presented for undergraduate students. The framework is to use logical games, from which we can popularize the notion of fixed-points, from which in turn we can teach program logics, a foundation for formal methods.

The first author of this article was a member of a program committee for a regional middle school mathematics contest in Russia, where the following puzzle was suggested by the contest committee:

A set of coins consists of 14 valid and 1 false coin. All valid coins have one and the same weight while a false coin has a different weight. One of the valid coins is marked while all other coins are unmarked. Is it possible to identify a false coin balancing coins 3 times at most?

Since both authors were coaches of undergraduates for ACM International Collegiate Programming Contests, a natural question concerning how to put the puzzle for programming arose. The corresponding programming problem was designed and offered to undergraduate students during training sessions—a brief form of the problem is:

Write a program with 3 inputs:
- a number N of coins under question;
- a number M of marked valid coins; and
- a limit K of balancings

which outputs either "impossible" or another executable interactive program ALPHA (in the same language) with respect to the existence of a strategy to identify a unique false coin among N coins with use of M marked valid coins and balancing K times at most. ALPHA should implement an identification strategy in the following settings. Each session with ALPHA begins from user choice of the false coin's number and whether it is lighter or heavier. Then a session consists of a series of rounds and the number of rounds in the session cannot exceed K. On each round the program outputs two disjoint subsets of numbers of coins to be placed on pans of a balance. User in his/her turn replies in accordance with his/her initial choice. The session should finish with the final output of the program ALPHA—the false coin's number.

Since the problem is to write a program that produces another program, we refer to the problem as the metaprogram problem.

In this metaprogram problem case the following game interpretation (which is very popular in mathematics and computer science theory) is useful due to its methodological and pedagogical importance. Let M and N be non-negative integer parameters

and let (N+M) coins be enumerated by consequent numbers from 1 to (N+M). Coins with numbers in [1…M] are valid while there is a unique false coin among those with numbers in [(M+1)…(M+N)]. A GAME(N,M) of two players User and Prog consists of a series of rounds. On each round a move of Prog is to partition the coins for the balancing: a pair of disjoint subsets (with equal cardinalities) of [1…(M+N)]. A possible move of User is either <, = or >, but his/her reply must be consistent with all constraints induced in previous rounds. Prog wins GAME(N,M) as soon as a unique number in [1…(M+N)] satisfies all constraints induced during the game. In this setting the metaprogram problem can be reformulated as follows:

Write a program which, for all $N \geqq 1$, $K \geqq 0$, and $M \geqq 0$, generates (if possible) a winning strategy in GAME(N,M) for Prog that uses at most K rounds.

What is a proper formalism for solving the metaprogram problem in this new setting? A hint is that a formalism should have explicit fixed-points. The following informal statement is very natural for every finite game of two players A and B: that a player A is in a position where he/she wins against B is equivalent to that A has a move prior to and after which the game is not lost, but after which every move of B leads to a position where (again) A wins against B. In other words: a set of positions with a winning strategy is a fixed-point of some transformation of sets of positions.

A functional paradigm is a well-known paradigm with explicit fixed-points, and the metaprogram problem can be solved in this framework. But, in accordance with regulations of ACM Collegiate Programming Contests, programs should be written in an imperative language such as Pascal, C, C++, or Java. So another paradigm would be better in this case than the functional one. It should be a paradigm that captures imperative style and fixed-points simultaneously. This is a Program Logics paradigm in general, and the formalism of the propositional mu-Calculus [3] in particular. But this formalism is in the most comprehensive form that relies upon transfinite induction and some basic modal logics: it is not easy to make the mu-Calculus easy for undergraduates. In this case another hint is an incremental approach to the introduction of the mu-Calculus in finite models only. The technical details of this approach are not appropriate for a *Communications* article, but readers should be aware the technical memo *Program Logics Made Easy*, which contains a solution for the metaprogram problem and background theory, is available on the Web [4].

Conclusion

The title of this article is an overt paraphrase of the title of the earlier *Communications* article "Engaging Girls With Computers Through Software Games" [2]. This article does not purport to be a comprehensive survey of early education on computer science theory or mathematical foundations of formal methods, nor of the educational role of the ACM Collegiate Programming Contests. This article is merely an expression of a strong belief of both authors that there is a deficiency of popular lectures, tutorials, and papers on the topics discussed here. We conclude by offering the following comments:

- Programming contests are a good opportunity for better education and popularization of computer science theory and mathematical foundations of formal methods;
- Computer science journals and magazines should promote popularization of computer science theory; and
- An attitude of theorists to the popularization and contests should and can be improved. C

References

1. Gal-Ezer, J. and Harel, D. What (else) should CS educators know? *Commun. ACM 41*, 9 (Sept. 1998), 77–84.
2. Gorriez, C.M. and Menida, C. Engaging girls with computers through software games. *Commun. ACM 43*, 1 (Jan. 2000), 42–49.
3. Kozen, D. and Tiuryn, J. Logics of programs. *Handbook of Theoretical Computer Science*. Elsevier and The MIT Press, 1990, 789–840.
4. Shilov, N.V. and Yi, K. *Program Logics Made Easy*. ROPAS Technical Memo No. 2000-7, Korea Advanced Institute of Science and Technology; ropas.kaist.ac.kr/lib/doc/ShYi00.ps
5. *Guidelines for SIAM Review Authors*; www.siam.org/journals/sirev/Revguide.htm
6. *SIGCSE Annual Report*. July 1997–June 1998; www.acm.org/sigcse/98annrpt.html

Nikolay V. Shilov (shilov@ropas.kaist.ac.kr) is a senior researcher in the A.P. Ershov Institute of Informatics Systems, Novosibirsk, Russia.

Kwangkeun Yi (kwang@ropas.kaist.ac.kr) is an associate professor in the Computer Science department at the Korea Advanced Institute of Science and Technology in Daejeon, Korea.