Engaging Students with Theory through ACM Collegiate Programming Contests*

Nikolay V. Shilov and Kwangkeun Yi
{shilov,kwang}@ropas.kaist.ac.kr
Research On Program Analysis System†
Korea Advanced Institute of Science and Technology

The role of formal methods in the development of computer hardware and software increases since systems become more complex and require more efforts for their specification, design, implementation and verification. Simultaneously formal methods become more and more complicated since they have to capture real properties of real systems for sound reasoning.

But a survey of formal methods as well as a discussion on the industrial demand and theoretical supply in this domain are out of the scope of this article[1]. In contrast we would like to discuss some problems and experiences regarding how to make popular mathematical foundations of formal methods with undergraduate students of departments which are closely related to further 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 telecommunication technology. We consider this popularization to be urgent, since in spite of the importance of formal methods for development of a reliable hardware and software this domain is not well acquainted to non-professionals. In particular, undergraduate students of mathematics, engineering, and technology departments consider formal methods outside of their interests, since

*To appear in Communcations of the ACM

†http://ropas.kaist.ac.kr National Creative Research Initiatives Center

[1] Maybe the simplest way to get opinion about scope and range of formal methods research and their industrial-strength applications is to visit special sites in Oxford (http://archive.comlab.ox.ac.uk/formal-methods.html) or in NASA (http://shemesh.larc.nasa.gov/fm/).

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 suppose that a deficit of popular lectures, tutorials, and papers on this topic is the main reason for this ignorance.

In this article we would like to attract the attention of people from academia and industry to a popular presentation of theoretical research for better computer science education. Simultaneously we would like to present some experience on popularizing program logics via training undergraduates for ACM Collegiate Programming Contests.

An Attitude to Popularization of Theory

A negative attitude to computer science theory is prevalent among university students. But an attitude of computer science theorists to popularization of their research with undergraduates is not much better. At least this attitude seems non-impressive in comparison with an attitude of mathematicians to popularization of their research.

There are several popular well-known national and international journals on popular mathematics primary oriented to a wide community which comprises students as well as researchers. Just two examples: Springer-Verlag's *The Mathematical Intelligencer* and *The American Mathematical Monthly* of The Mathematical Association of America. Both journals are published monthly, covered by products of Institute for Scientific Information (ISI), and have impact factors around 0.3. Moreover, mathematicians become more concerned by 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 sights 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 primely oriented on popularization of computer science theory. Theoretical research journals do not support popularization also. Maybe *SIGCSE Bulletin* is the unique serial edition which publishes papers related to experience with popularizing computer science theories. But it is published quarterly only, is not covered by ISI products, and its impact factor is not available. Overall, computer science theorists in comparison with mathematicians give at least

$$\frac{2 \text{ journals } \times 12 \text{ issues each}}{1 \text{ Bulletin } \times 4 \text{ issues}} = 6$$

times less attention to popularizing their research.

An attitude to student contests and olympiads is another essential difference between mathematicians and computer science theorists. In particular ACM Collegiate Programming Contest Finals are sponsored by IBM, neither by any special interest group nor any research or education institution. While this popular contest is an excellent opportunity to engage gifted students with theory of computer science. Moreover, it seems that computer science community considered contests of this kind out of scope of educational activities at all. The 24^{th} Final of the ACM contest took place on March, 2000 in Florida, while one of major annual events related to computer science education — the 31^{st} SIGCSE Technical Symposium in Computer Science Education — took place few days earlier in Texas. Both events did not provide links to each other. It contrasts with an attitude to mathematics contests. For example the Korean Mathematical Society granted equal priority to two major mathematical events hosted by Korean mathematicians this year: 41-st International Mathematical Olympiad and KMS'2000 International Conference *Mathematics in New Millennium*. This "regional" example reflects a general situation: mathematicians are seriously concerned by impact of contests.

The above arguments can mislead to a conclusion that an attitude to popular presentation of computer science theory is totally negative. This conclusion is invalid of course. Let us just remind (for example) International Summer Schools in Marktoberdorf [2] or European Summer Schools in Logic, Language, and Information[3]. But let us also remark that an auditory of these schools is comparatively small (a couple of hundreds per year) and (as usual) 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 (around 10,000 this year).

A natural question arises: how to engage undergraduate with theory through programming contests? First, of course, during training sessions. Training sessions are good opportunities to suggest students challenging programming problems which can not be solved without theoretical background

[2]http://www4.in.tum.de/div/summerschool/

[3]http://www.cs.bham.ac.uk/~esslli/

in spite of simple formulation. And then trainers should provide students with background theory as soon as students realize programming complexity of these problems. We would like to sketch below (as an example) how a program logics 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.

But prior to present our experience we would like to discuss the second opportunity for increasing an impact of ACM contests on better computer science education. We think that organizers of contests on regional and international levels should provide (after competition) comprehensive web-based tutorials on how to solve selected problems with background theory and sample programs. We regret that the importance of these tutorials is not well acknowledged by the organizers of the ACM competitions.

How to Make Program Logics Easy

Now we are ready to 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 paper was a member of a program committee for a regional middle school contest on mathematics in Russia. Several years ago the following puzzle was suggested by the committee on the contest:

> 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 trainers of undergraduates for ACM regional programming contests, a natural question on how to put the puzzle for programming arose. Finally the corresponding programming problem was designed and offered to undergraduate students on training sessions in Russia. A brief form of the problem follows:

Write a program with 3 inputs
- a number N of coins under question,
- a number M of marked valid coins,
- 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 can not 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 which produce another program, we would like to refer to the problem as the *meta-program problem*.

In this meta-program 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 meta-program problem can be reformulated as follows:

> Write a program which, for all $N \geq 1$, $K \geq 0$, and $M \geq 0$, generates (iff possible) a winning strategy in *GAME(N,M)* for *prog* that uses at most K rounds.

What is a proper formalism for solving the meta-program problem in this new setting? A hint is that a formalism should have explicit fixed-points. Really, 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 no doubt the meta-program 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 like Pascal, C, C++, or Java. So another paradigm would be better in this case than the functional one. It should be a paradigm which captures imperative style and fixed-points simultaneously. This is a Program Logics paradigm in general, and the formalism of the propositional μ-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 μ-Calculus easy for undergraduates! In this case another hint is an incremental approach to the introduction of the μ-Calculus in finite models only. Technical details in this approach are out of scope of CACM contributions but we would like to remark that lecture notes of a tutorial "Program Logics Made Easy" are available on the web [4]. The reader may want to see them for full story with a solution for the meta-program problem and background theory.

Conclusion

The title of the article is an obvious paraphrase of the title of the recent publication in the CACM "Engaging Girls with Computers through Software Games" [2]. Our article does not pretend to be a comprehensive survey neither of early education on computer science theory or mathematical foundations of formal methods, nor of an educational role of the ACM Collegiate Programming Contests. This paper is just an expression of a strong belief of both authors that

- there exists a deficit of popular lectures, tutorials and papers on the topics listed above;
- programming contests are a good chance 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.
- an attitude of theorists to the popularization and contests should and can be improved.

References

[1] J. Gal-Ezer and D. Harel, *What (Else) Should CS Educators Know?* Communications of the ACM, v.41, n.9, 1998, p.77-84.

[2] C.M. Gorriez and C. Menida, *Engaging Girls with Computers through Software Games.* Communications of the ACM, v.43, n.1, 2000, p.42-49.

[3] D. Kozen and J. Tiuryn, *Logics of Programs.* Handbook of Theoretical Computer Science, v.B, Elsevier and The MIT Press, 1990, p.789-840.

[4] N.V. Shilov and K. Yi, *Program Logics Made Easy.* ROPAS Technical Memo No. 2000-7, Korea Advanced Institute of Science & Technology, http://ropas.kaist.ac.kr/lib/doc/ShYi00.ps

[5] *Guidelines for SIAM Review Authors.* http://www.siam.org/journals/sirev/Revguide.htm

[6] *SIGCSE Annual Report. July 1997 - June 1998.* http://www.acm.org/sigcse/98annrpt.html