**Grammar-Based Procedural Generation of Programming Exercises**Porfirie-Denissa PILIPĂUȚANU

*Military Technical Academy “Ferdinand I” of Bucharest   
Contact: denissa.porfirie@mta.ro*

**Introduction**: With an increasing number of students enrolled in programming classes, it is a well-known fact that the teachers regularly face the challenge of producing new knowledge assessment content due to the publication of existing materials and open access to them. Even though the research targeted towards the procedural generation of programming exercises has seen some growth in recent years, the currently proposed solutions rely heavily on parameterized templates and do not provide the desired degree of versatility and complexity. Therefore, the main purpose of my research is to improve the currently proposed solutions and provide a greater variety of exercises for students.

**Methods and Results:**

There is a plethora of exercise types that can be used to assess the student’s programming knowledge. My main focus is on the semantics practices for understanding the flow of a program. This kind of exercises can be modelled into a triplet that consists of a problem statement, a code sequence and a correct answer.

In order to generate the full exercise, I defined a pipeline made up of the subsequent steps: function generation, template customization, template compilation, template execution and output processing. If the generated function does not meet the configuration criteria (a minimum and a maximum number of lines, and a certain degree of uniqueness) or if there was a compilation or runtime error, the pipeline will abort the current build and will restart building from step one. By using this approach, the function generation step produces the code sequence member of the triplet I defined previously. The problem statement and the correct answer are generated in the output processing step, after the template compilation and execution.

The function generating step employs a grammar with the help of which a syntax tree is generated using a top-down approach. Based on this syntax tree a terminal definition is built using a bottom-up approach. This implementation is flawed since it consistently generates multiple declarations of the same variable or expressions and statements that use undeclared variables. Therefore, it takes a long time to generate a correct code sequence because of the compilation errors. In order to solve this problem, I implemented a symbol table to keep track of the identifiers and the types of the variables. Consequently, the probability of a compilation error has seen a significant decrease.

**Conclusions:** Taking everything into account, even though grammars helped me tremendously in achieving the desired versatility and complexity of the generated programming exercises, the solution still needed an improvement to be feasible. The problem has been solved by implementing a symbol table that, as a result, increased the quality of the generated code and decreased the time needed for its production.

**Coordinator:** Cristian Chilipirea, PhD

**Keywords:** *Procedural generation, Grammars, Programming exercises, E-learning*

**References:**

1. A. Aho, M. Lam, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison Wesley, Boston, 2006.
2. T. Tiam-Lee, and K. Sumi. “Procedural Generation of Programming Exercises with Guides Based on the Student’s Emotion.” *IEEE International Conference on Systems, Man and Cybernetics*, Miyazaki, Japan, Oct. 2018, pp. 1465-1470.