STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LV**, Number 3, 2010

TEACHING MODEL CHECKING TO UNDERGRADUATES

A.VESCAN AND M. FRENŢIU

Abstract. The way program verification is taught in our faculty is firstly described. One of the verification methods is model checking, shortly presented to the students in one lecture. One laboratory project consists in using the SPIN tool. The difficulties encountered by students with this project are presented in this paper.

1. Introduction

The need for more reliable software [1, 9, 16, 6], and the role of Formal Methods [9, 3, 14, 19, 7] are well known. During the last two decades, research in formal methods has led to the development of some very promising techniques that facilitate the early detection of defects. These techniques are accompanied by powerful software tools that can be used to automate various verification steps. Investigations have shown that formal verification procedures would have revealed the exposed defect in, e. g., the Ariane-5 missile, Mars Pathfinder, Intel's Pentium II processor, or the Therac-25 therapy radiation machine [4].

Today, the computers are used in all fields of human activities. More and more programs are needed, and software critical systems require error-free programming [1].

If we want to build software of good quality, to increase its reliability, we have to contribute to better education of human resources. It is considered that the main barrier against the usage of Formal Methods is the lack of good professionals, able to use such methods.

Each university must educate better software engineers capable to use the newest methods, which increase the quality of software products and improves the software processes. In this direction, one of the important subject that must be taught to undergraduates is Verification and Validation.

Received by the editors: July 18,2010.

2010 *Mathematics Subject Classification.* 68Q60, 68N30.

1998 *CR Categories and Descriptors.* D.2.4 [**Software**]: Software engineering – *Software/Program Verification*; D.2.5 [**Software**]: Software engineering – *Testing and Debugging*.

Key words and phrases. verification and validation, model checking, tools, education.

2. The role of Verification and Validation (V&V)

One means to Software Quality Assurance is V&V. The course V&V is one of the important courses that contributes to obtain well-educated practitioners. We teach such a course to the third year undergraduates. The theoretical basis for building reliable software products is given here. The course consists of three main parts:

- the theory of program correctness;
- the methods of verification and validation;
- the consequences on software engineering practice.

The entire curricula of this course, and also, the undergraduate study program may be seen at [20].

One cannot understand V&V if he does not know the concept of program correctness. The first part of the course presents this concept and gives methods to prove correctness. More important, the accent is put on the methods to achieve this correctness.

The methods discussed in the second part are: proving correctness, testing, inspection, symbolic execution, and model checking. It is underlined that all of them must be practiced during the software process [8]. Their usage may be informal, or more formal, complete or only some of them, depending on the type of the system which is built. For safety-critical systems all of the above mentioned methods must be used. We consider that all future software engineers should be aware of all verification methods.

The third part of the course presents the Cleanroom methodology [17], the role of V&V for Software Quality Assurance and Software Process Improvement, and the consequences of correctness theory on software engineering practice [5, 6, 8].

At the undergraduate level we cannot afford to teach the mathematical basis of model checking, the theory that lies at the basis of model checking theory. Instead, the main theoretical aspects are presented in a two hours lecture, and the existence of tools and examples of such tools are shortly presented.

Model checking [12] is a verification technique that explores all possible system states in a brute-force manner. In this way, it can be shown that a given system model truly satisfies a certain property. The property specification prescribes what the system should do, or what it should not do, whereas the model description addresses how the system behaves. The model checker examines all relevant system states to check whether they satisfy the desired property. To make a rigorous verification possible, properties should be described in a precise and unambiguous manner. A temporal logic, which is a form of modal logic that is appropriate to specify relevant properties, is used as a property specification language. In term of mathematical logic, one checks that the system description is a model of a temporal logic formula. Temporal logic is basically an extension of traditional propositional logic with operators that refer to the behavior of systems over time. It allows for the specification of a broad range of relevant system properties [12] such as functional correctness (does the system do what it is supposed to do?), reachability (is it possible to end up in a deadlock state?), safety ("something bad never happens"), liveness ("something good will eventually happen"), fairness (does, under certain conditions, an event occur repeatedly?), and real-time properties (is the system acting in time?). Also, model checking may be used to check the conformance of design with the requirements [2].

Teaching model checking to undergraduates was already proposed by others [13, 18]. The undergraduate curricula cannot contain an entire course about model checking, but we consider that it is an important verification method and must be present in such a course.

3. Problems with teaching model checking

As a laboratory project one tool was presented to the students and this tool was Spin [10].

The language Promela was presented and introduced to students during the first part of the laboratory. Several examples [15] were presented and explained to better understand the syntax of the language but more than that, the semantics of the structures were also explained. The non determinacy was explained by using several examples. The notion of a process in Promela was presented and discussed. The concurrency, interference and interleaving between processes were described. The students played with the provided examples [15] and experienced the concurrency and interleaving. Several ways/methods for deterministic steps/atomic executions of statements of different processes were explained. So, one of the difficulties encountered with teaching model checking was the unfamiliarity with concurrent systems.

The structure of the laboratory consisting in model checking may be found here [21]. The laboratory consists in two hours and for the assignment the students had two problems: in class assignment problem and homework assignment problem.

The properties to be checked were first expressed using assertions. As in class assignment the students had to implement in Promela a process to compute a given value and then to use assertions to establish the correctness of the computation. They also had to use assertions to express preconditions and postconditions. The majority of them were able to work alone, without any additional help. This was a simple exercise to help student create a process and use assertions for correctness, and also to prepare them for the homework assignment problem. They have already used assertions (preconditions, postconditions, invariants) in a previous laboratory [21] when they use ESCJAVA and JML [11].

Another problem faced by some of the students was to make them model a system in class, immediately after the presentation of similar examples. Obviously, they need more time to process and understand modeling using Promela and thinking about verification of a system/model using a model checker.

For the second part of the class we have discussed the use of LTL formula to express the properties that the model should have. They have run the prepared examples [15]: about critical section in two processes, about deadlock and starvation. They have used LTL formula to express these properties and used the JSpin tool to verify them. During the use of JSpin tool with LTL formula the students were very enthusiastic about the "power" of the tool, especially about the checking process.

For homework assignment they have received the following problem: Consider the frog pond shown in Figure 1. Three female frogs are on the three stones on the left and three male frogs are on the three stones on the right. Find a way to exchange the positions of the male and female frogs, so that the male frogs are all on the left and the females are all on the right. The constraints that your solution must satisfy are as follows: frogs can only jump in the direction they are facing. They can either jump one rock forward if the next rock is empty or they can jump over a frog if the next rock has a frog on it and the rock after it is empty. Model the above system using a Spin model, and show that it is possible to reach the desired end state.

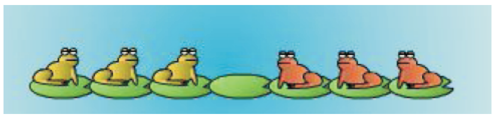

FIGURE 1. The Frog Pond Puzzle

The students played the game and tried to find a solution using [22]. Some of them found the solutin quickly and were able to sketch an algorithm for the solution.

Only a few of the students were able to model the system and check for solution, some of them used assertions and others used a LTL formula. Only a few of them understood that the property is checked in all states of the system model. They were very satisfied about the outcome of their finding.

Other students didn't understand correctly what they should do and they modeled the solution of the problem and not the system and the rules. They were very excited that the JSpin model checker always "gave" them the solution, and each time they run (randomly execution) the created processes they reached the solution!

Other students found the solution of the problem in the JSpin example directory but when asked about the model, the way that the model should be used in JSpin, they didn't know what to answer. They were not able to explain (even explained during the previous laboratory) how the model is used and how the model checker verify the LTL formula.

4. Conclusions

Verification by Model Checking had motivated the students, although they met the above difficulties. Since the allocated time for this subject was small, the examples were small and they could be considered as toys examples. Nevertheless, they offer to the students the possibility to acquire this new method of verification. They saw that Model Checking is a good mean to catch errors earlier in the model, to eliminate the rework, and to improve the quality of the product and the software process.

Nevertheless, we can improve this part of the course by choosing more suitable examples and give them to the students as homework projects.

Also, we must insist on improving the (first) model chosen by students. And, as well, we must insist that the students should pay attention to a broadly verification of a system, at least in the following three directions:

- to use all verification methods;
- to carefully design the testing cases according to the chosen criteria;
- to verify the robustness of the system.

References

[1] R. W. Butler, S. C. Johnson, *Formal Methods for Life-Critical Software*, in Computing in Aerospace 9 Conference, San Diego, California, 1993, pp. 319–329.

[2] M. Chechik, J. Gannon, *Automating Analysis of Consistency between Requirements and Designs*, IEEE Transactions on Software Engineering, 27(2001), no.7, pp.1–21.

[3] E. M. Clarke, J.M. Wing, *Formal Methods: State of the Art and Future Directions*, ACM Computing Surveys, 28 (1996), no. 4, pp. 626–643.

[4] N. Dershowitz, *Software Horror Stories*, `www.cs.tau.ac.il/~nachumd/horror.html`.

[5] M. Frentiu, *On Program Correctness and Teaching Programming*, Computer Science Journal of Moldova, 5(1997), no.3, pp. 250–260.

[6] M. Frentiu, *Correctness, a very important quality factor in programming*, Studia Univ. "Babe-Bolyai", Seria Informatica, L(2005), no.1, pp. 12–21.

[7] M. Frentiu, *The Need to Teach Formal Methods*, Analele Universităţii Bucureşti, LV, 2006.

[8] M. Frentiu, *Verificarea si Validarea Sistemelor*, Ed. Presa Universitara Clujeana, Cluj-Napoca, 2010, pp. 232, ISBN 978-973-610-979-9.

[9] C. M. Holloway, *Why Engineers Should Consider Formal Methods*, in 16th AIAA/IEEE Digital Avionics Systems Conference, Volume 1, 1997, pp. 1.3-16 – 1.3-22.

[10] G. J. Holzman, *The Model Checker SPIN*, IEEE Transactions on Software Engineering, 23(1997), no.5, pp. 279-295.

[11] JML, Java Modeling Language Home Page, `http://www.eecs.ucf.edu/~leavens/JML/`

[12] J. P. Katoen, *Principles of Model Checking*, MIT Press, 2008, pp. 995.

[13] H. Liu, D.P. Gluch, *A proposal for introducing model checking into an undergraduate software engineering curriculum*, Journal of Computing Sciences in Colleges, 18(2002), no. 2, pp.259–270.

[14] M. J. Lutz, *Alloy, Software Engineering, and Undergraduate Education*, in First Alloy Workshop, colocated with the Fourteenth ACM SIGSOFT Symposium on Foundations of Software Engineering, Portland, 2006, pp. 96–97.

[15] B. R. Mordechai, *Principles of the Spin Model Checker*, ISBN: 978-1-84628-769-5, 2008, pp. 216.

[16] B. Meyer,*Software Engineering in the Academy*, IEEE Computer, 34 (2001), pp. 28–35.

[17] H. Mills, M. Dyer, R.Linger, *Cleanroom Software Engineering*, IEEE Software, 4(1987), no.5, pp.19–25.

[18] H. Nishihara, K. Shinozaki, K. Hayamizu, T. Aoki, K. Taguchi, F. Kumeno, *Model Checking education for Software Engineering in Japan*, ACM SIGCSE Bulletin - COLUMN: Special section on formal methods education and training, 41(2009), no. 2, pp. 45–50.

[19] L. Yilmaz, S. Wang, *Integrating Model-Based Verification into Software Design Education*, Journal of STEM Education, vol.6 (2005), no.3–4, pp.29–34.

[20] Undergraduate study program - Babes-Bolyai University, `www.cs.ubbcluj.ro`

[21] A. Vescan, Software Systems Verification and Validation - course, seminar, laboratory - `http://www.cs.ubbcluj.ro/~avescan/`

[22] The Frog Pond Puzzle, `http://www.hellam.net/maths2000/frogs.html`

Department of Computer Science, Faculty of Mathematics and Computer Science,, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: {`avescan,mfrentiu`}`@cs.ubbcluj.ro`