Integrating Formal Methods Tools Into Undergraduate Computer Science Curriculum

Sotiris Skevoulis¹, Vladimir Makarov²

Abstract - This paper presents the results of a successfully completed funded project in the area of Computer Science and Formal Methods Education. The project was an effort to study the integration of formal methods software tools into an undergraduate curriculum. Our goals were to pioneer an innovative methodology that will increase the learning experience by introducing the use of mechanized approaches in teaching formal methods and to demonstrate the potential of undergraduate students for learning formal analysis techniques, by using these software tools. This methodology depends on the traditional teaching models and adds the use of automated software tools to enhance the student's learning experience. At the conclusion of this study we demonstrated increased student performance and understanding of difficult concepts in formal software analysis and design problems with the use of mechanized assistants. All tools and educational materials developed as part of this project, are publicly available.

Index Terms - Class invariant, Formal methods, Integrated environments, Z formal notation

INTRODUCTION

Our technological society has become more and more hardware and software dependent. Hardware and software handle every day tasks from traffic light control systems to nuclear reactor plants. The recent Y2K bug exemplified the tremendous impact costs and safety of software and hardware can have. The need for more reliable hardware and software manifests itself rapidly. Formal verification methods over at least the past decade have proved they can deliver the much needed reliability. Their wide-spread application is constrained for a number of reasons. One is the lack of expertise among software practitioners. Indisputably, the supplier of such highly educated and specially trained workforce, is the large number of universities and colleges that offer degrees in computer science and software engineering. However, in the United States, formal methods education is still considered an experimental area and the available instructional materials are limited to a small number of textbooks and lecture notes. As in every other science, experimentation and hands-on experience is of crucial importance for the deep understanding of the contents of mathematical logic and formal methods. To date, there has been a number of research projects [7] in the area of formal methods education that showed clearly the definite advantages of introducing formal methods into undergraduate curricula.

Also a number of initiatives have been announced and implemented [2] with the goal of studying the status and the degree of the integration of formal methods into software engineering curricula and proposing ways to facilitate this process. Similar concerns are raised also by both industry and academia regarding lack of skills from computer practitioners and lack of quality in the developed program code.

However our study takes this idea one step further and introduces the use of software tools to support the formal methods education offered by our School of Computer Science. The research that was carried out involved mainly the development of an Integrated Development Environment (IDE) that supports formal methods. The educational activities had to do with the delivery of two new courses that used as part of the instructional method the IDE. The tool addresses these issues and enhances not only the theoretical background of the students but also the technical skills via the hands-on experience gained with the use of specific formal methods tools such as a type checker, an animator, a theorem prover and a static source code analyzer. Students are given the opportunity to experiment with formal models, specifications and designs. The mechanized analysis provides them with an easy and time-effective way to try and verify their models. Moreover they have the opportunity to receive substantial help in the form of tool-generated feedback of what is wrong with their formal models and, in most instances, why it is wrong. The intellectual challenges and opportunities created by a new formal methods teaching methodology based on the use of cutting edge technology, tools and approaches are obvious.

The tool that is demonstrated in this paper is based on the synergistic integration of three key components: the model analyzer which analyzes the integrated formal Z [11] and semi-formal (UML) input models, the animator which can animate system behaviors and the Java code static analyzer which performs static analysis on hand written source code. The tool assists students in both, type checking their formal models and, in most instances, why it is wrong. Students are given the opportunity to experiment with formal models, specifications and designs. The mechanized analysis provides them with an easy and time-effective way to try and verify their models. Moreover they have the opportunity to receive substantial help in the form of tool-generated feedback of what is wrong with their formal models and, in most instances, why it is wrong.

In the following section we discuss the current status of formal methods education in the United States. Immediately after this section a detailed discussion on the tool itself follows. Research and educational activities are discussed in
the section following the tool discussion. Finally the project results and some planned future work conclude this paper.

**FORMAL METHODS EDUCATION**

All traditional engineering disciplines have been successful largely due to the symbiotic and highly integrated cooperation of theory and practice. Computer science curricula do not seem to address the need for integration of the theory learned in mathematics into the computer science courses. In modern engineering, the scientific analysis of the designs and models is possible through precise calculations and well-defined mathematical models. On the other hand software developers depend on techniques such as testing and debugging which is a form of the “trial and error” approach. During the software development process, precise calculations and mathematical representations of software models can only be achieved using formal methods. The problem lies not only with the small number of formal methods courses [13] offered but also with the lack of appropriate software tools that can assist students in understanding formal methods and improve their analytical skills and problem solving ability. Experimentation with the assistance of the appropriate tools such as compilers, debuggers, etc. has been a fundamental component in computer science courses. Unfortunately for a variety of reasons, it has not been part of formal methods education. Rushby, in a invited talk at World Congress of Formal Methods, indicates: "Mechanization is not the most important thing about formal methods. It’s the only important thing”[4].

Over the past decade many formal methods tools [10] have been developed. Most of them have understandably targeted the software development industry. Researchers and research labs around the world have been trying to make a case for the wide use of formal methods. Little effort has been made so far to introduce these tools into the undergraduate classrooms.

**THE TOOL**

JAIDE (Java Integrated Development Environment) [12] relies on a simple and intuitive integration of Z formal notation [11], Java programming language and Unified Modeling Language (UML). A snapshot of the tool is shown in figure 1.

It incorporates a number of user friendly features such as a virtual keyboard for entering all symbols of Z mathematical toolkit, coloring schemes for locating errors, etc.

**I. Using the Tool**

JAIDE was created to provide an intuitive and user friendly environment for a set of formal methods tools: the type checker, the theorem prover and the Java code static analyzer. Students that used this IDE offered periodically their feedback and its improvements based on that feedback were constant throughout the entire running period of the project. They were able to get on-the-fly feedback about the type-correctness of their specifications, animate their design models and validate their design decision at a very early stage of the development life cycle. It mainly consists of a Model Analyzer, a Theorem Prover, and a Static Analyzer as shown in figure 2.

JAIDE’s executables and the source code have become freely available to the educational community for use and experimentation. Several educators from across the United States have expressed a strong interest in getting and using it

**II. Analyzing Z Formal Specifications**

The most comprehensive survey on industrial applications of formal method [1] sponsored by Canadian and US government agencies reported over 60 cases of industrial applications of formal methods worldwide (a majority of them in Europe). A number of these projects has been developed using model based formal methods and in particular a formal notation called Z. Developed at the University of Oxford, it uses abstract mathematical data types and decomposes the system into small pieces called schemas. The abstract data types obey a rich collection of mathematical laws, which make it possible to effectively reason about the behavior of a system. The Z notation has found widespread use in specifying functional properties of software in a modular way.
The model analyzer ZTC (Z Type Checker)\[6\] performs extensive analysis on formal specifications written in Z. The analyzer performs the following tasks:

- Extensively analyzes the Z operation and data specifications by type-checking the models
- Reports problems to the student/user and points out where and what is wrong

Each specification is type-checked for potential type mismatch errors. The process is fully automatic and students get a complete listing of all typing errors. The type-checker works in a similar to a compiler mode. A snapshot of a formal specification written in Z along with the type-checking results is shown in figure 3.

The goal of model analysis is not to guarantee absolute formal model consistency and completeness, but to identify anomalies in early stages. The operation schemas in Z provide details about the functionality of the system's operations.

III. Animating Formal Models

The tool can animate formal specifications written in Z. The main goal of the animation process is to validate the user requirements at the earliest possible stage of the project development, and prevent mistakes from being propagated to the late phases of the development cycle, when the cost of debugging is too high. The key to tool’s animation approach is the classification of operation schemas into two categories: explicit and non-explicit. Animation can be performed on explicit operation schemas. Informally, an operation schema is explicit if the values of all its output variables, including post state, can be determined by evaluating some of the expressions in the schema. Explicit operation schemas can be translated into an intermediate format (Extended Guarded Command EGC) \[5,8\]

IV. Static Analysis of Java Code

The analyzer approaches the problem of potential error detection, using a lightweight theorem proving and formal methods approach, with pragmatic goals. It drops the guarantee of absolute correctness in order to deliver a partial solution that will detect the majority of the potential errors we search for. It applies static analysis and verification techniques, to automatically analyze Java programs and detect potential bugs that cannot be detected by compilers and other data flow-based analysis techniques.

Static analysis is carried out by an invariant-based generic component \[8\]. It analyzes Java programs in order to detect potential errors (bugs). It can automatically detect potential bugs: illegal dereferences, array bounds, string out of bounds and report them before the program is executed. For example, an array index out of bounds violation is detected in the example shown in figure 4.

For a Java class, invariants related to the category of error under examination are automatically generated and used to
assess the validity of variable usage in the implementation of this class. Our approach is distinctive in its emphasis to provide a practical and extensible generic mechanism for error detection to help both university students as well as industry practitioners who work with an object oriented language such as Java. Analyzer’s mechanism is capable of addressing error detection for a variety of error categories that cannot be caught by static analysis tools.

FIGURE 4
ARRAY INDEX OUT OF BOUNDS ERROR

The detection mechanism of our approach has certain limitations, mainly related to the intractability of theorem proving process. We cannot guarantee absolute success in finding all the code anomalies even for the restricted types of analysis discussed earlier. The goal is not to find all errors rather to find the majority of them in a fully automatic and transparent to the developer way.

RESEARCH AND EDUCATIONAL ACTIVITIES

A number of research projects in the area of formal methods education [7] showed clearly the advantages of introducing formal methods into undergraduate curricula. We most definitely agree with these results.

A group of 18 students was created to participate in the sequence of courses that would use our tools during the time period of four consecutive semesters. The tool was installed on 18 laptops (one for each student) that were made available from another supporting grant and was distributed to the students participating in the course sequence. Students were able to combine the tool’s automated feedback with the instructors’ more detailed explanations. Students during the Discrete Mathematics developed small hand-written mathematical proofs first and later they checked them against those generated by the automated theorem prover. As an example we used mathematical induction, which is a core element to fully understand every computer science student. The Formal Notations class covered the basics of Z Notation, a very popular formal notation based on set theory and mathematical logic. Students were asked to produce fairly small formal specifications expressed in Z using the paper and pencil approach. With the use of ZTC students were able to type-check their formal specifications, much like a program written in a programming language is checked by the compiler of this language. The Data Structures class mainly covered the invariant properties of a number of data structures such as Java classes, Stacks, queues as well as loop invariants. The formal Software Development class was devoted to the study of small software systems (i.e. a student enrollment manager) and how formal specifications can be animated and logical errors can be discovered. Students also used the Java Analyzer tool to discover logical errors in their hand written java programs.

The first six weeks of each course were devoted to the introduction of the theoretical foundations while the next four weeks, students worked on a set of in-class exercises using the “paper and pencil” approach. During this time absolutely no tools were used in any way. Neither the correct answers nor the laptops were available to the students during these weeks. During the final four weeks of the semester, students received training on how to use the theorem prover (first semester), and the type-checker (third semester) and took on the same set of problems which they tried to solve using the tools. After they turned in their solutions along with printed reports from the tool, the solutions were presented to them and they were asked to practice (modify, find other ways to express the same formal constructs, etc.) with them at home. It is also important to emphasize here that the four courses covered totally different areas of knowledge, i.e. discrete mathematics, Z formal notation, data structures and formal software development.

Apart from the data collected regarding the error count with and without the use of tools, and in order to validate and understand more deeply the quantitative results, a survey was designed and distributed to students at the end of each semester asking them to provide their feedback to the professor and the research team. Survey results were turned in anonymously to avoid any pressure on students and make them more forthcoming about the tools. The results we
obtained were consistent with the quantitative results in that students did learn from the use of tools as an instructional medium. Students universally responded that the tools were helpful, informative and more importantly encourage interactions with their classmates to discuss the automatically tool generated feedback on their assignments. They did, however, feel that there is room for many improvements. Those range from minor user interface changes (error highlighting choice of color) to more substantial critiques about the notation and difficulties they faced with it. What they liked most was the fact that they were able to experiment and try out different ways of specifying logical constructs and learn Z language more easily.

RESULTS

The use of formal methods tools produced two remarkable results. Error count in developing formal specifications dropped by 20%. A snapshot of raw data tables that were created for each course is shown in Table 1. (P&P stands for Paper and Pencil approach). The particular set of results corresponds to the results of the fourth course offered in the Spring semester of 2004. The overall results obtained from this project suggest the following ideas:

- **Learning.** Students are indeed learning using the tools as evidenced by the smaller error count for the same problems worked with and without the help of the tools.

- **Attitude.** Student attitudes toward formal methods are becoming more positive when they have automated software assistant by their side. We consider this to be positive and encouraging.

- **Cooperative learning environment.** It was also deeply gratifying to realize that the use of tools encouraged the creation of a cooperative learning environment. Students were free to take their laptop across the classroom and discuss the feedback received from the tool with fellow classmates, exchange ideas and experiment with them. In a sense, it resembled the successful environment that XP (Extreme – Pair Programming) advocates in software development.

A set of tangible contributions also include the software which was created and is freely distributed to anyone interested, the tutorial, the users manual and documentation developed, the examples and of course the website which we will keep maintaining and enhancing. As part of the National Science Foundation funded project we established the Center for the Advancement of Formal Methods Education (CAFME- www.cafme.org) which guarantees the continuity and accessibility of the results of this and other related projects. CAFME’s web site is the spring board of a more general effort to study the problems in formal methods education, experiment with existing and innovative approaches and suggest solutions in the years to come.

**TABLE I**

<table>
<thead>
<tr>
<th>Student</th>
<th>HW1 P&amp;P Tool</th>
<th>HW2 P&amp;P Tool</th>
<th>HW3 P&amp;P Tool</th>
<th>HW4 P&amp;P Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>5</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>5</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>4</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
<td>9</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>7</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>8</td>
<td>5</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>9</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>12</td>
<td>6</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>18</td>
<td>9</td>
<td>3</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>

**RELATED WORK**

Over the past thirty years there have been many approaches that intend to formally specify and verify functional correctness of programs. For that purpose programming languages have been specifically invented for those approaches. More recently, research work at Iowa State University has resulted in JML. It uses formal methods to identify particular kinds of bugs in Java and provide also some kind of feedback to the programmer about those potential bugs. This approach is focused on Object-Oriented technology and Java. Our tool can handle Z formal notations along with both Object-Oriented and Function Oriented models.

Z/Eves [10] from ORA Canada, is an integrated formal methods environment which is capable of working with Z formal specifications and provides theorem proving capabilities. It requires a programming language (Python) to be installed before you can run it. Since there is not direct mapping between Z symbols and ASCII character set, the only way to input them is to use the virtual keyboard. Our approach does provide a sophisticated virtual keyboard, but also included (as part of ZTC) a complete set of mappings between the entire set of Z mathematical kit and ASCII characters. We have also included a number of features such as error line coloring scheme, click-and-find utilities which makes navigating through files and identifying lines with errors much easier task.

On theoretical basis, our approach most closely resembles the work at Z/EVES. We also incorporate techniques such as symbolic execution and simplification of algebraic expression before and during the formal model analysis.

**FUTURE WORK**

Our work continues on a number of directions. We are focused on two major ideas. We will continue improving the current IDE based on the feedback of our students who use it as part of their courses. Immediate plans call for a more sophisticated and informative error message mechanism. Students will be
getting more informative messages which will not only highlight the line where the error occurred but also try to give an idea of where the error was originated from.

We will also make the tool more flexible in terms of working with other formal notations. A great number of them, except Z, is currently available. Some of them offer certain advantages over Z in specifying software requirements within particular domains (i.e. financial, web, etc). The IDE will be flexible enough to work with those notations and the corresponding type checkers, analyzers, etc. Based on that development, we will continue experimentation on a variety of formal notations always striving to make our formal methods classes more informative and fulfilling.

ACKNOWLEDGMENT

This research has been partially supported by the National Science Foundation (NSF) under the grant DUE-0126991 of the CCLI directorate. The authors would also like to thank the Dean of the School of Computer Science and Information Systems, Dr. Susan Merritt for the support she has offered for the entire period of this grant.

REFERENCES


