Panel: Engage in Reasoning with Tools

Gregory Kulczycki
Virginia Tech
Computer Science Department
Falls Church, VA 22043
1-703-538-8370
gregwk@vt.edu

Murali Sitaraman
Clemson University
School of Computing
Clemson, SC 29634
1-864-656-3444
murali@clemson.edu

Nigamanth Sridhar
Cleveland State University
Electrical Engineering & Computer Science
Cleveland, OH 44115
1-216-687-5341
n.sridhar1@csuohio.edu

Bruce W. Weide (Moderator)
The Ohio State University
Computer Science and Engineering
Columbus, OH 43210
1-614-292-5813
weide.1@osu.edu

SUMMARY

A central goal of computer science education is to teach students how to reason about the correctness of the code they write. Typically, students use a trial and error process and check that their logic “works” by running it on test inputs. Typically, instructors encourage them towards logical reasoning through manual tracing of the code. Rarely reasoning tools are used in the process, at least partly because few instructors are familiar with them and fewer have the time to investigate and experiment. The purpose of this panel is to introduce the attendees to a variety of reasoning tools the presenters have used in their classrooms. In some cases, the tools have been used in only one or two classes of a course to illustrate specific points. In other cases, entire projects have been done using the tools. The courses range from the introductory sequence and discrete structures to software engineering and graduate-level courses. The tools are freely available on the web and attendees will be encouraged to experiment with the reasoning tools on their own laptops to solve simple reasoning problems.

CCS Concepts

Software and its engineering—software creation and management—software verification and validation—formal verification
Software and its engineering—software notations and tools

Keywords

Automated tools, components, correctness proofs, programming by contract, specification, verification, and web tools.

Helping students learn how to reason that the code they write behaves as expected is a central goal of computer science education. This panel will demonstrate the use of tools in achieving this objective. There are several advantages of using reasoning tools, some of which are freely available through online interfaces.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

Copyright is held by the owner/author(s).

SIGCSE '16, March 02-05, 2016, Memphis, TN, USA

ACM 978-1-4503-3685-7/16/03.

<http://dx.doi.org/10.1145/2839509.2844657>

Inside the classroom, reasoning tools make it possible to demonstrate quickly certain non-trivial aspects of reasoning, helping students ask and answer a variety of “what if” questions that would be difficult or impossible to do in the absence of tools. Outside the classroom, students can learn to check the logic of their code at their own pace at a place and time of their convenience, and understand the ideas using a number of exercises. The overall goal is to help educators see the potential benefits of using reasoning tools in the context of introductory computer science, discrete structures, programming languages, and software engineering courses. The panelists will use a variety of examples to illustrate the ideas, ranging from simple straight-line code with subtle errors to code involving objects, loops, and recursion.

The panel will be organized as follows. Each panelist will present a reasoning concept or two in about 8 minutes, and then follow it up with 8 minutes of “what if” questions from either from the panelist to the attendees or from the attendees to the panelist (asked in the style of students learning the topic). The goal is to motivate and demonstrate how one might approach teaching rigorous reasoning about software correctness. Some of the exercises might involve the attendees trying the tools on their own laptops. The goal is to introduce a range of concepts that can be effectively taught with tools using various course contexts so that the attendees to incorporate the ideas in their own courses.

1. BRUCE W. WEIDE

Reasoning tools have been used in the second semester of the *introductory sequence* in “software” for CS majors at Ohio State. The software sequence (designed by Weide and others) consists of two semester-long courses, the second of which is taken concurrently with the first of two courses in “foundations.” The first software course introduces some logic concepts and notations so that students can read and understand (but not write) formal specifications, and the second one also instructs them in how to write some specifications. In the first foundations course, students become familiar with some proof techniques. This means that by late in the second software course, all these concepts can be tied together nicely via examples using automated verification tools that students should increasingly be expected to encounter in the workplace shortly after graduating.

Bruce W. Weide is Professor Emeritus of Computer Science and Engineering at The Ohio State University. He and his colleagues

in the RSRG at Ohio State have studied and taught design, specification, and verification of component-based software for over 30 years. Weide and his colleague, Tim Long, were awarded the IEEE Computer Society Computer Science and Engineering Undergraduate Teaching Award in 2000 for their innovations.

2. MURALI SITARAMAN

In three required courses for CS majors at Clemson, Sitaraman and other instructors have used a web-based reasoning tool supported by the RESOLVE verifying compiler, details of which may be found in [1, 2]. For connecting proofs with establishing software correctness in a *discrete structures* course, he has used a variety of exercises, including ones that involve inductive proofs and recursive code, for example [3]. Clemson has also used the reasoning tool more extensively in a software engineering course for underscoring the role of formal contracts in establishing the correctness of component-based code in a modular fashion. Details of these reasoning projects may be found in [4].

Murali Sitaraman is Professor of Computer Science in the School of Computing at Clemson University. His research spans a spectrum of foundational, practical, and educational topics in software engineering. He has been involved in developing tools and techniques for enhancing mathematical reasoning in undergraduate CS education for nearly 25 years. He has offered tutorials and workshops to educators on teaching mathematical reasoning at SIGCSE several times, including four of the last five years.

3. NIGAMANTH SRIDHAR

At Cleveland State University, Nigamanth Sridhar uses reasoning tools in two different courses. In the senior-level course on *software engineering*, students learn the role of formal assertions in the context of specifying software interfaces. Students work in the Visual Studio environment and use Code Contracts [5] to check correctness of code with respect to specifications. The other is a graduate-level formal methods course that is focused exclusively on tools for model checking and software verification, such as Dafny, NuSMV, RESOLVE, and SPIN.

Nigamanth Sridhar is an Associate Professor in Electrical Engineering and Computer Science at Cleveland State University. He co-directs the graduate program in Software Engineering. For the last several years, Sridhar has worked on the application of formal methods in reasoning about correctness of software for small embedded systems. Sridhar, along with his colleagues, is currently running an NSF-funded project to train high school teachers across Ohio to teach the new CS Principles course.

4. GREG KULCZYCKI

Greg Kulczycki has introduced reasoning tools in *Object-Oriented Software Specification and Construction*, a course in the core curriculum for software engineering graduate students at George Mason University. The course uses [6] to present a specification-centric view of Java development and introduces assertions such as requires clauses, ensures clauses, abstraction functions, and representation invariants. The course uses Dafny [7], an object-oriented language that can be annotated with assertions. Using the Dafny compiler, code exercises can be written, compiled, and verified online, though Dafny is limited in its support of abstraction and modularity.

Gregory Kulczycki served as an assistant professor at Virginia Tech and has worked as a formal methods researcher at Battelle Memorial Institute. He is currently a professor of practice at Virginia Tech and is coordinating the Computer Science Depart-

ment's involvement in the online Masters of Information Technology program. He has been introducing students to formal reasoning in one form or another for over a decade and has significant experience with instruction via online courses.

5. PANEL AUDIENCE

The panel should be of interest to all educators who might be interested in infusing tools to engage students and help them see the benefits of logical reasoning. Since examples from a variety of courses using different tools will be presented, the audience will be quickly introduced to a number of ideas with which they can experiment in their classrooms. SIGCSE routinely has a sizeable contingent of attendees interested in mathematical thinking [8], code analysis, and tools. The panel will be of interest to a good number of these attendees who are looking for ways to help students learn to reason better about their code. The panel will involve significant tool demonstration and active audience participation to make it engaging.

6. ACKNOWLEDGMENTS

We acknowledge NSF grants CCF-1161916, CCF-1162331, DUE-0942542, and DUE-1022941. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

- [1] Sitaraman, M., Adcock, B., Avigad, J., Bronish, D., Buccì, P., Frazier, D., Friedman, H.M., Harton, H., Heym, W., Kirschenbaum, J., Krone, J., Smith, H., and Weide, B.W., "Building a Push-Button RESOLVE Verifier: Progress and Challenges," *Formal Aspects of Computing* 23, 2011, 607-626.
- [2] Cook, C.T., Harton, H.K., Smith, H., and Sitaraman, M., "Specification Engineering and Modular Verification Using a Web-Integrated Verifying Compiler," *Proc. 34th International Conference on Software Engineering*, IEEE/ACM, 2012, 1379-1382.
- [3] Drachova, S. V., Hallstrom, J. O., Hollingsworth, J. E., Krone, J., Pak, R., and Sitaraman, M., 2015. Teaching Mathematical Reasoning Principles for Software Correctness and Its Assessment. *ACM Trans. Computing Education* 15, 3, Article 15 (August 2015), 22 pages. DOI=10.1145/2716316 <http://doi.acm.org/10.1145/2716316>
- [4] Cook, C.T., Drachova, S. V., Sun, Y-S., Sitaraman, M., Carver, J., and Hollingsworth, K. E., "Specification and Reasoning in SE Projects Using a Web-IDE," *Proc. 26th Conference on Software Engineering Education and Training*, IEEE, 2013.
- [5] Fahndrich, M., Barnett, M., Leijen, D., and Logozzo, F., "Integrating a Set of Contract Checking Tools into Visual Studio," *Procs. Second International Workshop on Developing Tools as Plug-ins (TOPI 2012)*, IEEE, 2012.
- [6] Liskov, B. and Guttag, J., *Program Development in Java*, Pearson, 2000.
- [7] Dafny Tutorial, <http://rise4fun.com/Dafny/tutorial> (5 Sept 2014)
- [8] Baldwin, D., "Math-thinking-1 – Mathematical reasoning in CS curricula", at: <http://mail.geneseo.edu/mailman/listinfo/math-thinking>