Towards Practical and Rigorous Automated Grading in Functional Programming Courses

Dragana Milovančević

EPFL, Switzerland

Grading programming assignments is difficult and time consuming. With the ever-growing numbers of students in programming courses, autograding has become the necessity. In this talk, we present our recent and ongoing work on applying formal methods for automated grading. In Part I, we suggest the use of program verifiers for automated grading of programming assignments. For this, we use Stainless, an open-source verifier for Scala programs. Our extensive evaluation shows promising results. We are developing the infrastructure to apply those techniques in EPFL's functional programming course. In Part II, we suggest the use of theorem provers for automated grading of proofs by induction written by students. For this, we use LISA, a proof assistant based on set theory.

Part I: Proving Correctness of Programming Assignments – Lessons Learned

The overarching goal of practical and rigorous automated grading has led to the development of a number of tools to aid teaching programming courses. Researchers have introduced state of the art techniques specifically targeted at grading [4, 18], feedback generation [17, 19], and repair [7, 11, 16, 21]. While highly automated, neither of these tools provide any formal correctness guarantee.

Recently, we have proposed a new approach to automatically and formally verify correctness of programming assignments [13]. Latest advances in the field of formal verification have resulted in tools that can verify strong correctness properties of important pieces of software infrastructure [3, 8, 10]. However, formal verification is challenging. Case studies show ratios such as nine lines of specifications per executable line [2], which is impractical for automated grading. In our approach, we use equivalence checking and functional induction to automatically prove or disprove correctness of student submissions, as well as a clustering algorithm to efficiently treat many submissions at once. We illustrate the underlying techniques on examples[1] throughout the talk. Our approach is fully automated: the only inputs to our system are student submissions and reference solutions, without additional annotations. Moreover, our approach is rigorous: each submission classified as correct is evidenced by a proof of equivalence, and each submission classified as incorrect is evidenced by a counterexample. Finally, our approach is practical: we implement our techniques on top of the Stainless verifier [20], to support equivalence checking of Scala programs; our evaluation shows that our system is highly effective in practice.

Part II: Proving Correctness of Proof Assignments – Ongoing Work and Future Directions

Complementary to the efforts for programming assignments, we consider the problem of grading proof assignments. Proof assistants have been increasingly finding their way in dedicated graduate courses [9, 14, 15], undergraduate courses [12] and high schools [1, 5]. We aim to go one step further, proposing the use of proof assistants for introductory *programming* courses. In our functional programming course, students not only write programs, but also have to prove certain program properties, such as that a tail-recursive function is equivalent to its non-tail recursive counterpart. Such proofs only need instantiation of free parameters and equational reasoning, which makes them feasible for automated grading. We consider the LISA proof assistant [6], whose high-level interface and DSL provide an intuitive and programmer-friendly environment for students. We discuss an example set of proofs from a programming exam, in LISA[2] and in Stainless[3].

---

[1] Available at `https://github.com/epfl-lara/stainless/tree/main/frontends/benchmarks/equivalence`

[2] `https://github.com/epfl-lara/lisa/tree/main/lisa-examples/src/main/scala/MapProofTest.scala`

[3] `https://github.com/epfl-lara/stainless/tree/main/frontends/benchmarks/dotty-specific/valid/MapTr.scala`

References

[1] Visualizing Geometrical Statements with GeoView. *Electr. Notes Theor. Comput. Sci.*, 103:49–65, 11 2004.

[2] Mario Bucev and Viktor Kunčak. Formally verified quite ok image format. In *2022 Formal Methods in Computer-Aided Design (FMCAD)*, pages 343–348, 2022.

[3] Tej Chajed, Haogang Chen, Adam Chlipala, M. Frans Kaashoek, Nickolai Zeldovich, and Daniel Ziegler. Certifying a file system using crash hoare logic: Correctness in the presence of crashes. *Commun. ACM*, 60(4):75–84, mar 2017.

[4] Elena L. Glassman, Jeremy Scott, Rishabh Singh, Philip J. Guo, and Robert C. Miller. Overcode: Visualizing variation in student solutions to programming problems at scale. *ACM Trans. Comput.-Hum. Interact.*, 22(2), March 2015.

[5] Frédérique Guilhot. Formalisation en Coq et visualisation d'un cours de géométrie pour le lycée. *Technique et Science Informatiques*, 24:1113–1138, 11 2005.

[6] Simon Guilloud, Sankalp Gambhir, and Viktor Kuncak. LISA – A Modern Proof System. In *To appear in ITP, Conference on Interactive Theorem Proving*, 2023.

[7] Sumit Gulwani, Ivan Radiček, and Florian Zuleger. Automated clustering and program repair for introductory programming assignments. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2018, page 465–480, New York, NY, USA, 2018. Association for Computing Machinery.

[8] Gernot Heiser, Gerwin Klein, and June Andronick. sel4 in australia: from research to real-world trustworthy systems. *Commun. ACM*, 63(4):72–75, 2020.

[9] Frederik Jacobsen and Jørgen Villadsen. On Exams with the Isabelle Proof Assistant. *Electronic Proceedings in Theoretical Computer Science*, 375:63–76, 03 2023.

[10] Gerwin Klein, June Andronick, Matthew Fernandez, Ihor Kuz, Toby Murray, and Gernot Heiser. Formally verified software in the real world. *Commun. ACM*, 61(10):68–77, sep 2018.

[11] Junho Lee, Dowon Song, Sunbeom So, and Hakjoo Oh. Automatic diagnosis and correction of logical errors for functional programming assignments. *Proc. ACM Program. Lang.*, 2(OOPSLA), oct 2018.

[12] Hendriks Maxim, Cezary Kaliszyk, Femke van Raamsdonk, and Freek Wiedijk. Teaching logic using a state-of-art proof assistant. *Acta Didactica Napocensia*, 3, 06 2010.

[13] Dragana Milovancevic and Viktor Kunčak. Proving and Disproving Equivalence of Functional Programming Assignments. In *ACM SIGPLAN Conf. Programming Language Design and Implementation (PLDI)*, 2023.

[14] Tobias Nipkow. Teaching Semantics with a Proof Assistant: No More LSD Trip Proofs. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 24–38, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[15] Benjamin Pierce. Lambda, the Ultimate TA Using a Proof Assistant to Teach Programming Language Foundations. pages 121–122, 08 2009.

[16] Yewen Pu, Karthik Narasimhan, Armando Solar-Lezama, and Regina Barzilay. Sk_p: A neural program corrector for moocs. In *Companion Proceedings of the 2016 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity*, SPLASH Companion 2016, page 39–40, New York, NY, USA, 2016. Association for Computing Machinery.

[17] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated feedback generation for introductory programming assignments. *SIGPLAN Not.*, 48(6):15–26, June 2013.

[18] Dowon Song, Myungho Lee, and Hakjoo Oh. Automatic and scalable detection of logical errors in functional programming assignments. *Proc. ACM Program. Lang.*, 3(OOPSLA), October 2019.

[19] Dowon Song, Woosuk Lee, and Hakjoo Oh. *Context-Aware and Data-Driven Feedback Generation for Programming Assignments*, page 328–340. Association for Computing Machinery, New York, NY, USA, 2021.

[20] Nicolas Charles Yves Voirol. Verified functional programming. page 229, 2019.

[21] Ke Wang, Rishabh Singh, and Zhendong Su. Search, align, and repair: Data-driven feedback generation for introductory programming exercises. *SIGPLAN Not.*, 53(4):481–495, jun 2018.