Do mathematical proof skills in continuous areas of Maths develop algorithmic thinking in CS students in HE?

Julia Crossley
Department of Computer Science
City University of London

Abstract

The aim of this proposal is to suggest a possible mechanism for developing abstract skills in Computer Science students during their undergraduate studies. Some research has gone into the development of conceptual skills in school aged students and into the meaning of abstraction. My aim is to develop methods using the idea of Mathematical proofs, to help students generalise, which will help them develop self-efficacy and confidence in using their intellectual curiosity. Students will be exposed to formal proofs, and most of all encouraged to think of some themselves.

Introduction

The aim of my project is to identify whether exposure to general systematic and iterative processes in Maths help non-mathematicians develop stronger algorithmic skills; whether linking discrete and continuous processes helps students produce more efficient code. In doing this, I hope to investigate whether this will help them:

- Generalise results in order to trace bugs and possibly avoid them.
- Identify exceptions or limiting cases.
- Find expressions that result in less time and space complexity. Some expressions may not be common mathematical results that are easy to find – the computer scientists may need to work out the expression themselves!
- Calculating the complexity of an algorithm may also require some mathematical skill – particularly if the computer scientist needs to approximate an expression in a function.

Mathematical Proofs

Mathematical proofs are covered in first year undergraduate degrees in Maths and rarely touched on earlier. This area of pure Maths is often thought to be the most novel and difficult (Almeida, 2010), perhaps due to the following reasons: complex mathematical language and deeper, systematic thinking that may at times seem counter intuitive. The skills gained contribute to a more insightful approach to the broader subject and better abstract reasoning (Schoenfeld, 2009). Students encounter mathematical subtleties, learn to distinguish between them and use them.

Computer Science students as a group rarely study formal mathematical proofs, instead taking a more practical, 'methods based' approach to Maths in the same way Engineers and Scientists do. If they do, these are in areas seen as directly relevant to fundamental CS, such as discrete Maths, and are 'case' proofs rather than ones of general results. Studies have been conducted into measurement of long-term skills in these areas of Maths (Qian & Lehman, 2017); a comparable measurement in continuous and other areas of maths could be of use to CS educational research, particularly if the impact of these gained skills can be seen at a fundamental and advanced level.

Difficulties in Computer Science and Maths

Qian & Lehman (2017) identify Maths as an area causing difficulty, in that adherence to knowledge gained at secondary school impacts understanding of concepts in computer science. I would like to take this further: can this skills gap be tackled by more exposure to continuous and deep Mathematics language and processes? Many CS programs do not emphasize this, particularly at the early stages, and this is seen as detrimental by some even early on (Baldwin, Walker & Henderson, 2013). The latter manifests itself as using a trial-and-error approach to Maths and programming.

Cognitive overload can be a problem, particularly when facing numerous levels of abstraction and more 'practical' considerations such as memory, types, etc. Lack of clarity over the meaning of symbols and how they are processed in a programming language can also be an area of confusion for students.

The practice of Mathematical Proofs certainly provides deeper insight for Mathematics students into abstract processes, by guiding them in breaking down concepts, understanding them and identifying both linguistic and mathematical subtleties. Can it do something similar for Computer Science students, in a manner that can be contextualised in a computer program?

Some research has gone into this, focusing on the areas of discrete maths and logic, as they can be applied to Computer Science at a more fundamental level. Other areas of Maths can be of use but are more relevant to advanced and applied areas of Computer Science (Hartel, Van Es, Tromp, 1995). I'd like to investigate whether proofs in these other areas of Mathematics can also offer insight for students through the abstract skills required to understand and use them. Proofs that require an iterative process and use continuous mathematical variables have the potential to help students understand algorithmic processes on a more abstract level.

My project

The method of instruction would be pattern-oriented, as suggested by Muller and Haberman, in that fundamental concepts would be illustrated through different processes (Muller and Haberman, 2008). My reason for choosing these mathematical processes, is somewhat "soft" (Hazzan, 2008), in that many of them present important CS concepts such as nesting, iteration, recursion and perhaps even arithmetic overflow using objects and syntax that is already somewhat familiar to students. There are also some similarities between learning to code and learning to prove: students 'trace' proofs; develop an intuition for the appropriate tools to use for a given problem, such as a mathematical inequality; use this intuition to implement a proof.

Specific areas of Mathematics to present to students also need to be decided. My original idea was to use proofs illustrating steps from discrete Mathematics to continuous ones, for example by starting from patterns in sequences and series, then leading on to continuous functions. The reason for this is that many proofs in these two areas are similar, which makes the progression between these two types of sets more natural. It may however be that students benefit more from having a deeper look into the uses of Mathematics in areas they are exposed to at the earlier stages of their CS education, and this would likely be in Discrete Mathematics.

Methods

Some work still needs to go into identifying appropriate methods for the study. The goal of the study is to build on the metacognitive skills of a group of students who will benefit the most from the intervention. My tendency would be primarily to choose an experimental group of students with a broad range of experience in Mathematics, but nevertheless a willingness to learn a new approach and curiosity in a variety of areas including Maths. Students who embody this mindset, despite having lower confidence in their Maths skills, will also be strongly encouraged to take part. The success of the intervention will be determined both from their project process and output, as well as their reported increase in confidence in their coding skills and ability to use new tools. Students will then have been equipped with a toolkit of mathematical expressions and arguments that will be useable to them in their career as Computer Scientists.

References:

Almeida, D.(2010) A survey of mathematics undergraduates' interaction with proof: some implications for mathematics education. International Journal of Mathematics education in Science and Technology, vol 31, 2000 – issue 6, pages 869 – 890. https://doi.org/10.1080/00207390050203360

Baldwin, D., Walker, H., Henderson, P. (2013). The roles of mathematics in computer science. ACM inroads, Volume 4, Number 4 (2013), Pages 74-80.

Cetin, I., Dubinsky, E. (2017). Reflective abstraction in computational thinking. The Journal of Mathematical Behaviour, 47, 70-80. https://doi.org/10.1016/j.jmathb.2017.06.004

Hartel, P.H., van Es, B., Tromp, D. (1995). Basic proof skills of computer science students. In: Hartel, P.H., Plasmeijer, R. (eds) Functional Programming Languages in Education. FPLE 1995. Lecture Notes in Computer Science, vol 1022. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-60675-0_50

Hazzan, O. (2008). Reflections on Teaching Abstraction and Other Soft Ideas. SIGCSE Bull., 40(2), 40–43. https://doi.org/10.1145/1383602.1383631

Ko, Y., Knuth, E. (2013) Validating proofs and counterexamples across content domains: Practices of importance for mathematics majors. The Journal of Mathematical Behaviour, Volume 32, Issue 1,2013,Pages 20-35,ISSN 0732-3123. https://doi.org/10.1016/j.jmathb.2012.09.003

Loksa, D., Ko A., Jernigan, W., Oleson, A., Mendez, C. & Burnett, M. (2016). Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance. 1449-1461. 10.1145/2858036.2858252.

Muller, O., Ginat, D. & Haberman, B. (2007) Pattern-oriented instruction and its influence on problem decomposition and solution construction. In Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE'07). ACM, New York, 151–155. http://dx.doi.org/10.1145/1268784.1268830

Schoenfeld, A. (2009) Teaching and learning proof across the grades: A K-16 perspective. Routledge, New York, NY (2009), pp. xii-xvi

Qian, Y. & Lehman, J. (2017) Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. ACM Trans. Comput. Educ. 18, 1, Article 1 (March 2018), 24 pages. https://doi.org/10.1145/3077618

Zeng, H. & Jiang, K. (2010). Teaching mathematical proofs to CS major students in the class of discrete mathematics. Journal of Computing Sciences in Colleges. 25. 326-332.