

Improving Mathematics and Programming Education – The IMPEd Initiative

Ralph-Johan Back and Linda Mannila

Dept. of IT, Åbo Akademi Univ.
Turku Centre for Computer Science
Joukahaisenkatu 3-5 A, Turku, Finland
{backrj, linda.mannila}@abo.fi

Mia Peltomäki and Tapio Salakoski

Dept. of IT, University of Turku
Turku Centre for Computer Science
Joukahaisenkatu 3-5 B, Turku, Finland
{mia.peltomaki, tapio.salakoski}@utu.fi

Abstract

In this paper we discuss the topics that should be included in basic computing education and the levels of education at which they should be introduced. We present the resource centre *IMPED*, which aims at improving education in mathematics and programming by drawing on results from empirical studies. The research focuses on three main topics, which are also briefly summarized.

1 Introduction

Mathematical subjects in general and information technology in particular are vital for the success of the Finnish knowledge-based society. New technologies cannot be adopted without a sufficiently educated and trained workforce. This requires the entire school system to be taken into account, as high quality elementary and secondary school education are prerequisites for an excellent university system.

Both mathematics and computing rest essentially on a solid theoretical foundation. Whereas mathematics instruction at a comprehensive level understandably focuses on learning how to use basic mathematics in practice, high school mathematics is more theoretical. Mathematics is taught quite extensively at both comprehensive and high levels in the Finnish school system. For instance, a student choosing the advanced mathematics syllabus at high school level has 10 compulsory and three elective courses, and a student taking the general syllabus has six compulsory and two elective courses. In other words, all high school students have to take at least six courses in mathematics (Finnish National Board of Education 2003).

By contrast, the Finnish high school core curriculum does not include any courses on computing or any other similar topic. Instead, focus is put on learning how to use the computer and its applications as tools, and only a few high schools offer their students courses covering the more theoretical aspects of computing. Thus, although both mathematics and computing have comparable

theoretical foundations, this does not show in the way computing is introduced in education. On the other hand, the practical applicability of mathematics may also be overlooked if the courses are viewed as purely theoretical. The absence of this link between theory and practice in computing is also prevalent at university level (Almström et al. 2001).

In our opinion, combining mathematics and computing is a natural way of linking theory and practice together. It offers students an insight into the theoretical basis of computing and provides them with a more practical view of mathematics and one of its application areas.

Starting in 2000, researchers at the departments of IT at Åbo Akademi University and the University of Turku have worked closely together with the aim of developing a course sequence that could provide such a link. New methods have been developed and empirically evaluated in classroom settings.

As of fall this year (2007), the research activities are complemented by the resource centre *IMPED* (Improving Mathematics and Programming Education), aimed at disseminating our work to other educational institutions. The centre is a joint project between the IT departments at Åbo Akademi University and the University of Turku, led by Professor Ralph-Johan Back (Åbo Akademi University) and Professor Tapio Salakoski (University of Turku). The project is funded by the *Federation of Finnish Technology Industries*.

In this paper, we will briefly present the *IMPED* initiative and open up a discussion regarding the way that basic education in mathematics and IT could be organized at secondary and tertiary level to support the future of the Finnish knowledge-based society.

2 Teaching Mathematics and Programming

IMPED is supported by our active research which aims to improve the understanding of mathematics and programming among secondary school students and first-year students at universities and polytechnics. So far our research has focused on the following topics:

- Teaching mathematics with a precise notation and a logical foundation (structured derivations)
- Teaching programming using a simple language (Python)
- Teaching the construction of correct programs (invariant based programming)

In the following sections we will briefly describe each of these parts.

2.1 Mathematics with a Precise Notation

Exact formalism is usually avoided at lower levels of education such as high school mathematics courses. When not exposed to proofs and exact definitions, students are not given the opportunity to truly understand the very foundation of mathematics. Mathematical and logical reasoning remains a ‘secret’ and proofs become magic tricks when the formal parts are hidden. The reason for not covering formalism and proofs at lower levels is that these topics are considered too difficult and abstract. This might certainly be the case, if they are taught in the same way as they would be to experts or university students in their final years of study.

Structured derivations is a calculational proof format developed by Ralph-Johan Back and Joakim von Wright (Back et al. 1997; Back & von Wright 1999). They have extended the derivational-style proof approach as presented by Dijkstra (1990) and van Gasteren (1990) by adding nested derivations (subderivations), allowing inferences to be presented at different levels of detail. The method is thus theoretically well founded. A sample derivation is given in Figure 1.

These derivations introduce a new approach to teaching mathematics including exact formalisms and proofs. Structured derivations facilitate problem solving and enhance the possibilities of rereading and discussing solutions afterwards, as compared with traditional informal approaches to writing down solutions. The method is more rigorous and exact than the traditional methods used in secondary level mathematics, and it contributes not only to the preciseness of expression but also to more systematic and straightforward presentation

$$\begin{aligned}
 & \begin{cases} x+2y = 4 \\ 2^x = 8^y \end{cases} \\
 \equiv & \text{ {simplify the second equation} } \\
 & \bullet 2^x = 8^y \\
 \equiv & \text{ {rules of exponents} } \\
 & 2^x = 2^{3y} \\
 \equiv & \text{ {the exponential function with base 2 is monotonic} } \\
 & x = 3y \\
 \dots & \begin{cases} x+2y = 4 \\ x = 3y \end{cases} \\
 \equiv & \text{ {assume the second equation and simplify the first one} } \\
 & \bullet [x = 3y] \\
 & x+2y = 4 \\
 \equiv & \text{ {substitute from the second equation} } \\
 & 3y+2y = 4 \\
 \equiv & \text{ {equation solving} } \\
 & y = \frac{4}{5} \\
 \dots & \begin{cases} y = \frac{4}{5} \\ x = 3y \end{cases} \\
 \equiv & \text{ {substitute } y \text{ from the first equation into the second one and calculate} } \\
 & \begin{cases} y = \frac{4}{5} \\ x = \frac{12}{5} \end{cases}
 \end{aligned}$$

Figure 1: Sample structured derivation

of mathematical reasoning.

The use of structured derivations in mathematics education has been extensively evaluated, starting in 2001, when a longitudinal study teaching the compulsory courses in advanced mathematics using structured derivations was initiated in a Finnish high school. The results have been encouraging, suggesting that this method has the potential to improve students’ performance in mathematics courses as well as their understanding of mathematical reasoning and problem solving (Back et al. 2004; Peltomäki & Salakoski 2004).

Starting in fall 2006, structured derivations are the standard approach used in the basic course on logic at the IT department at Åbo Akademi University. Currently, the evaluation of the use of structured derivations in education involves five institutions at both high school and university level.

2.2 Practical Programming

The question of which language to introduce to novices has been discussed quite extensively, and may ultimately be considered a matter of taste. When teaching programming, the focus should be on developing programming skills, not on the language or its syntax. However, in order to maximize the time spent on programming, one should avoid having to ‘waste’ time on additional and unnecessary language constructs.

Notational overhead and complex syntax render many of the popular languages unsuitable for novices. Java, for instance, is one of the most commonly used languages at universities today, but requires much extra code in order to complete even the simplest program. All time spent on such extras is time away from actual programming.

In 2004 we decided to teach a simpler language, and chose *Python*¹ for this purpose. Python is an interpreted high-level scripting language designed by Guido van Rossum as a general-purpose language but with education firmly in mind; van Rossum has even suggested that everybody could master programming using Python (van Rossum 1999). The language is freely available and has many of the features characteristic to a language suitable for teaching programming (Milbrandt 1993; Weinberg 1998), such as a small and clean syntax, an enforced structural design, dynamic typing and expressive semantics. The interpreter provides immediate feedback and there is an active user community providing books, tutorials, examples and other course material online.

Python comes with a large number of modules which provide extra functionality, and many of these can be used to make even small programs interesting and motivating. The code snippet in Figure 2 illustrates how the *webbrowser* module is used to let students write programs that open web pages in the default browser, while at the same time practicing important concepts

¹ See <http://www.python.org>.

```

import webbrowser

options = {"A" : "http://www.google.com",
          "B" : "http://www.yahoo.com",
          "C" : "http://www.altavista.com"}

for site in options:
    print site + " : " + options[site]

try:
    choice = raw_input("\nChoose a site: ")
    webbrowser.open(options[choice])
except:
    print "You did not pick a valid alternative."

```

Figure 2: Sample Python program

such as user input, iteration, dictionaries and exception handling.

We have now used Python in nearly 15 programming courses at secondary level, and the experiences have been positive (Grandell et al. 2006; Mannila et al. 2006). The benefits of teaching a simple language such as Python have also been pointed out by others (e.g. Agarwal & Agarwal 2006; Guzdial 2003; Miller & Ranum 2006; Radenski 2006). As a result of our experience from secondary level, the IT department at the University of Turku incorporated Python in one of its basic courses in 2006. Starting in fall 2007, the IT department at Åbo Akademi University made the switch from Java to Python as the language of instruction in the first programming course.

2.3 Constructing Correct Programs

Universities give various courses on logic, formal methods, program semantics etc. aiming at giving the students the skills needed to write correct programs. However, these are not directly linked to the practical programming courses in which students learn to write code. As a result, it is common that students do not apply the issues learnt in the theoretical courses when doing actual programming. Instead, students learning programming usually go about it by ‘trial and error’; they iteratively write code, test it, and modify it as needed. The modifications might introduce new errors into the code, calling in turn for further changes. This iterative process makes it difficult to create correct programs, since one can never prove that a program is correct by testing, one can only point out the errors one happens to find using some, perhaps arbitrarily chosen, test cases. Another approach is needed to learn to construct programs that are known to work.

Invariant based programming (Back 1983; Back 2005) is a diagrammatic hands-on approach to constructing correct programs. Similar ideas were earlier presented by Reynolds (1978) and van Emden (1979). In invariant based programming, the program invariants are formulated before the code is constructed. The process starts with drawing pictures illustrating the basic data structures involved and how they will be changed during execution of the algorithm (Figure 3). From the pictures, the initial, final and intermediate situations are identified. These situations are generalized and the program is constructed as a (nested) invariant diagram (Figure 4). The final program is correct if it is consistent (all

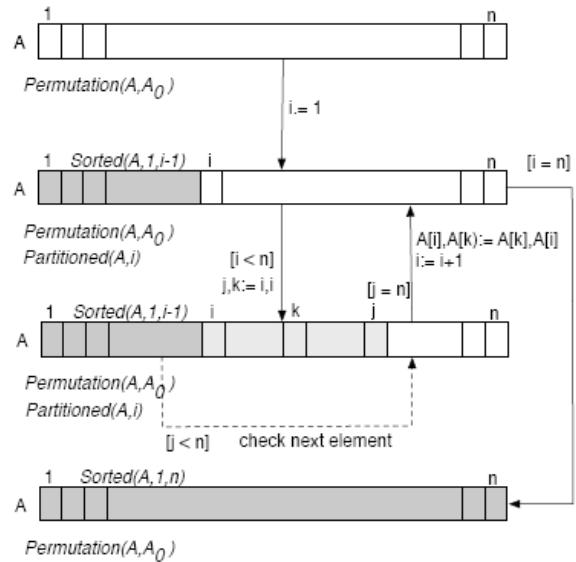


Figure 3: Final version of the pictures drawn to illustrate the algorithm at work

invariants are preserved), terminates (no infinite execution loops exist) and live (termination only occurs at final situations). The programs can be constructed and verified using only pen and paper, but tool support is also provided (SOCOS) (Back et al. 2006).

The invariant based approach was introduced in a first-year undergraduate course at Åbo Akademi University in spring 2007 with positive results (Back 2007; Back et al. 2007).

3 Putting the Pieces Together

The approaches to teaching mathematics and programming presented in the previous section work well together and give a solid foundation in both the theory and practice of programming. For instance, a course covering structured derivations and another covering practical programming in Python give students all the background knowledge they need to successfully complete a course covering invariant based programming.

Although the approaches are well suited for creating a continuum of courses, they can also be introduced separately. Structured derivations can be used solely as a

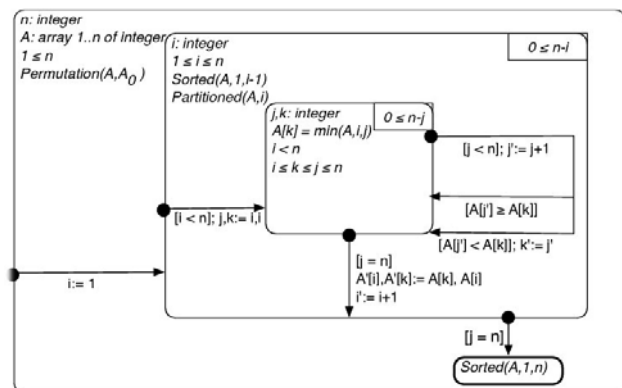


Figure 4: Final version of the invariant diagram for selection sort

way for improving students' understanding of mathematics, and invariant based programming can be introduced to students who have background knowledge in a proof format other than structured derivations and a programming language other than Python.

4 Discussion

In our opinion, there is a need for a closer link between theory and practice in CS education, and in this paper we have briefly presented the IMPED resource centre and the research and activities that take place within the centre aimed at bridging the gap between theory and practice. To what degree should the problem be addressed at university level and to what degree at lower levels? What background knowledge would be desirable from students enrolling for CS studies? And should we, as educators in the computing field, take actions in order for computer science to be reintroduced as an independent subject at high school level? In that case, what measures would be appropriate?

5 References

- Agarwal, K. K. and Agarwal, A. (2006), Simply Python for CS0, 'J. Comput. Small Coll.', vol. 21, no. 4, pp. 162-170.
- Almstrum, V. L., Dean, C. N., Goelman, D., Hilburn, T. B., and Smith, J. (2001), Support for teaching formal methods, SIGCSE Bulletin vol. 33, no. 2, pp. 71-88.
- Back, R.-J. (1983), Invariant based programs and their correctness, in W. Biermann, G. Guiho & Y. Kodrato, eds, 'Automatic Program Construction Techniques', MacMillan Publishing Company, pp. 223 – 242.
- Back, R.-J. (2005), Invariant based programming revisited, Technical Report 661, TUCS - Turku Centre for Computer Science, Turku, Finland.
- Back, R.-J. (2007), Basic Approach and Teaching Experiences. To appear in 'Formal Aspects of Computing Science'.
- Back, R.-J., Eriksson, J. & Mannila, L. (2007), Teaching the Construction of Correct Programs Using Invariant Based Programming. Accepted to the '3rd South-East European Workshop on Formal Methods', Thessaloniki, Nov. 30 – Dec. 1, 2007.
- Back, R.-J., Eriksson, J. & Myreen, M. (2006), Verifying invariant based programs in the SOCOS environment, in 'Teaching Formal Methods: Practice and Experience'. BCS-FACS.
- Back, R.-J., Grundy, J. & von Wright, J. (1997), 'Structured calculation proof', Formal Aspects of Computing, 9, pp. 469 – 483.
- Back R.-J., Peltomäki, M. & Salakoski T. (2004), Structured Derivations Supporting High-School Mathematics, in 'Current research on mathematics and science education', Univ. of Helsinki, pp. 104 – 122.
- Back, R.-J. & von Wright, J. (1999), A method for teaching rigorous mathematical reasoning, in 'Proceedings of Int. Conference on Technology of Mathematics', University of Plymouth, UK.
- Dijkstra, E. W. & Scholten, C. S. (1990), Predicate Calculus and Program Semantics, 'Texts and Monographs in Computer Science', pp. 21 – 29.
- Finnish National Board of Education (2003), Lukion opetusuunnitelman perusteet (Principles of the Upper Secondary School Curriculum, in Finnish), Valtion painatuskeskus, Helsinki.
- Grandell, L., Peltomäki, M., Back, R., & Salakoski, T. (2006), Why complicate things?: introducing programming in high school using Python. In 'Proceedings of the 8th Australian Conference on Computing Education', pp. 71 – 80.
- Guzdial, M. (2003), A media computation course for non-majors. In 'Proceedings of the 8th Annual Conference on innovation and Technology in Computer Science Education' (Thessaloniki, Greece, June 30 - July 02, 2003), ACM, New York, NY, pp. 104-108.
- Mannila, L., Peltomäki, M. & Salakoski, T. (2006), What About a Simple Language? Analyzing the Difficulties in Learning to Program. 'Computer Science Education', vol. 16, no. 3, 2006, pp. 211 – 228.
- Milbrandt, G. (1993), Using problem solving to teach a programming language in computer studies, 'Journal of Computer Science Education', 8(2), pp. 14 – 19.
- Miller, B. and Ranum, D. (2006), Freedom to succeed: a three course introductory sequence using Python and Java, 'J. Comput. Small Coll.', vol. 22, no. 1, pp. 106 – 116.
- Peltomäki M. & Salakoski T. (2004), Strict Logical Notation Is Not a Part of the Problem but a Part of the Solution for Teaching High-School Mathematics, in 'Proceedings of Kolin Kolistelut – Koli Calling. The 4th Annual Finnish/Baltic Sea Conference on Computer Science Education', pp. 116 – 120.
- Radenski, A. (2006), "Python first": a lab-based digital introduction to computer science. In 'Proceedings of the 11th ITiCSE' (Bologna, Italy, June 26 - 28, 2006), ACM, New York, NY, pp. 197 – 201.
- Reynolds, J. C. (1978), Programming with transition diagrams, in D. Gries, ed., 'Programming Methodology', Springer Verlag, Berlin, pp. 159 – 165.
- van Emden, M. H. (1979), Programming with verification conditions, 'IEEE Transactions on Software Engineering', volume SE-5, number 2, pp. 148 – 159.
- van Gasteren, A. J. (1990), On the Shape of Mathematical Arguments, 'Lecture Notes in Computer Science', pp. 90 – 120.
- van Rossum, G. (1999), Computer Programming for Everybody, Corporation for National Research Initiatives, CNRI Proposal #90120-1a. Available online: <http://www.python.org/doc/essays/cp4e.html>.
- Weinberg, G. M. (1998), 'The Psychology of Computer Programming', Dorset House Publishing Company.