Presented at Madif–12, Växjö, January 15, 2020

Integrating programming in Swedish school mathematics: description of a research project

Kajsa Bråting[1], Cecilia Kilhamn[2] and Lennart Rolandsson[1]
[1]Uppsala University, [2]University of Gothenburg

This paper describes a new research project investigating the implementation of programming in Swedish school mathematics, specifically in relation to algebra learning. Based on Chevallard's framework of transposition of knowledge, we investigate what types of activities and systems of representations are introduced and argued for as programming makes its way into mathematics teaching, and what these may entail. Tentative results reveal syntactic and semantic differences between programming and algebra that may cause problems for students. Interviews with teachers show that they seek inspiration and activities from social media and internet rather than textbooks and other published teaching material.

Background

During the past five years programming and computational thinking have emerged as new skills in several countries' national school curricula. Although computational thinking was introduced already in the 1980s (Papert, 1980) it did not become widely adopted, possibly because digital technology did not have the impact it has today through the internet and digital devices (Kotsopoulos et al., 2017). However, about thirty years later, Jeanette Wing returned to the term computational thinking arguing that it should be taught in schools alongside reading, writing and arithmetic (Wing, 2006).

The integration of programming and computational thinking in school curricula has been done in various ways (Mannila et al., 2014). For instance, in England, programming was made part of a whole new subject, "Computing" (Berry, 2013), while Finland and Sweden adopted a blend of cross-curriculum and single subject integration with the strongest link to mathematics (Bocconi et al., 2018). Unlike other countries, Sweden included programming in the mathematics curriculum in close connection to algebra through all grade levels, which makes the Swedish case unique in an international perspective. Until now, research on computational thinking and algebraic thinking has run on separate tracks, but the Swedish case offers a great opportunity to investigate the intersection of these two research domains.

The overall aim of the project described in this paper is to contribute to the international research field concerning the complex issue of implementing programming and computational thinking in school mathematics (grades 1-9),

specifically in relation to the learning of algebra. Based on two interrelated studies, the project attempts to explore how computational thinking is connected to algebraic thinking and, ultimately, how this connection may create opportunities, challenges or pitfalls for student learning of algebra. The aim of this paper is to describe the project as well as report some tentative results concerning what aspects of programming and computational thinking that have made their way into school practice. However, we commence with a brief survey over the two research fields of computational thinking and algebraic thinking. Both are fairly new as research fields and therefore attempts at defining and conceptualizing what is meant by these types of thinking is still an ongoing process, which we have described in more detailed elsewhere (Kilhamn & Bråting, 2019).

Computational thinking

Over the past decade, computational thinking (CT) has been paid increased attention in education at all levels. The significance of CT can be explained by the fact that it supports cognitive development and creative problem solving, as well as by the growing interest in artificial intelligence. In the Nordic countries, CT and programming are included in an evolving definition of digital competence (Bocconi et al., 2018). In Sweden, its insertion into the curriculum was a hasty process, handing new responsibilities to teachers without allowing much time for professional development.

Often, the concept represents a product-oriented perspective where various tools are applied in order to form CT skills (e.g. Grover & Pea, 2013). Aho (2012) defines CT as "the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms" (p. 832). Problem solving and algorithmic thinking are thus considered fundamental aspects of CT (Futschek, 2006). Developing CT also requires students to deal with the sometimes counterintuitive notations and conventions in the syntax of different programming languages.

Generally, CT is considered a more extensive concept than programming, although teaching and learning programming requires the use of CT (Hickmott et al., 2018). Furthermore, Brennan and Resnick (2012) highlight the appropriate role of programming as a means to develop CT, as they identify three dimensions of CT: i) *concepts* such as sequences, loops and data; ii) *thinking practices* such as debugging, remixing and abstracting; and iii) the *perspectives* expressing, connecting and questioning. These dimensions come to the fore in programming activities and form a useful framework for both teaching and assessing CT. An early initiative to use this framework has been taken by Nouri et al. (2019) in a thematic analysis of teacher interviews.

In a literature review of studies on CT in mathematics classrooms, Hickmott et al. (2018) found few studies that explicitly linked the learning of mathematics

concepts to CT. When there was evidence of concepts involving numbers, operations or algebra being engaged with, it was usually done with the primary intention of introducing programming concepts.

Algebraic thinking and algorithms

Researchers have suggested several frameworks for conceptualizing algebraic thinking in elementary grades. Many of these draw on Kaput's (2008) description of early algebra in terms of three content strands; the study of structures and relations; the study of functions; and the application of a cluster of modelling languages. Although a conclusive definition of algebraic thinking (AT) does not [yet] exist, most definitions include the two core aspects of expressing generalizations and symbolizing in formal or informal systems of representation (for several examples see Kieran, 2018).

The introduction of algebra in school mathematics was traditionally assumed to build on a thorough knowledge of arithmetic and was therefore not introduced until secondary school. For some decades, however, this division between arithmetic and algebra has been rejected, and it is now generally accepted that supporting algebraic thinking and the use of algebraic tools already in the early grades is beneficial to both the learning of arithmetic and the learning of algebra (Kieran, 2018).

Before 1980, traditional algorithms were seen as a cornerstone of arithmetic, but following the invention of pocket calculators a debate flourished on the necessity of these algorithms (Kamii & Dominick, 1997). Traditional algorithms were replaced by an increased emphasis on number sense and conceptual understanding. In Sweden, the term algorithm was removed from the description of arithmetic in the national curriculum in 2011, but re-inserted in 2018 as a core concept in algebra in connection to programming, implying a shift of emphasis from a procedural use of algorithms to a conceptual understanding of algorithms. In mathematics education, an algorithm is defined as a finite sequence of executable instructions which allows one to find a definite result for a given class of problems (Brousseau, 1997). The general structure of an algorithm connects algorithmic thinking to AT, so potentially, algorithms and algorithmic thinking may lie in the intersection of AT and CT.

Theoretical frames of the project

Within the research project we use the theory of transposition of knowledge (Chevallard, 2006) in order to study the implementation of programming in school mathematics, see Figure 1. While the relevance of *Scholarly knowledge* is what is achieved in practice, *Knowledge to be taught* is made legitimate by different actors making decisions about what to teach, when and why, and thus broken into smaller parts (Kang & Kilpatrick, 1992). When computer programming is transposed to

become a body of teachable knowledge in school, decisions are made for example concerning where to place the topic in relation to other topics, what kinds of activities and symbolic representations to use, as well as how and in what order different aspects of programming and CT are to be taught at specific age levels.

The process of transposition of computer programming has already started in Sweden through the choices made in the revision of the national curriculum (Swedish National Agency of Education, 2018). The next step of didactic transposition will happen as the national curriculum is interpreted and operationalized in textbooks and teaching materials, and further by teachers, to become *knowledge taught* in the classroom. The project described in this paper offers a unique opportunity to study the transposition of knowledge related to aspects of computer programming, computational thinking and algebra as it occurs in the implementation of the revised national curriculum in Sweden.

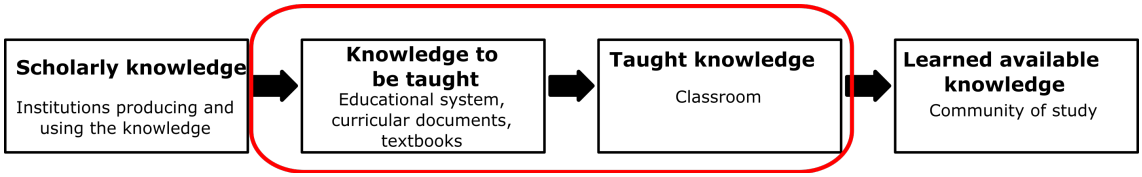

Figure 1: The four phases in the didactic transposition process. The encircled processes appear in focus of the project described in this paper.

To deepen our analysis, we identify and describe the systems of representations that appear in different semiotic representations (Duval, 2006), how and why these are chosen, discarded or taken for granted in each phase of the didactic transposition. Especially, we investigate how computer-related representations interact with already present algebraic systems of representation.

Method

Within the project, we conduct two studies that both separately and related to each other help us to discern important aspects of the issue of integrating programming in school mathematics. Next, we briefly describe them and indicate what has been done so far.

Study 1: Teaching materials and textbooks

This study focuses on the transposition from *Scholarly knowledge* to *Knowledge to be taught* in the didactic transposition process. We investigate the current steering documents in mathematics education, government produced teaching materials, and commercially produced mathematics textbooks including teacher guides. The selection of textbooks is made according to their popularity and diversity due to earlier studies showing that there are substantial differences between textbook series (Bråting, Madej & Hemmi, 2019). At present, five textbook series from two different publishers have been chosen for analysis. We

will analyse syntax and semantics in programming contexts in order to discover similarities and differences in the systems of representations related to programming and algebra. We will also analyse possible progression lines of computational and algebraic thinking through grades 1-9 in the textbooks.

In this paper, we will report some results from our initial analysis of two textbook series for grades 1-6 as well as our investigation (Kilhamn & Bråting, 2019) of programming activities suggested in a government-provided teaching material available online [1] based on the 2018 revised curriculum in Sweden. In the latter, we have utilized Duval's (2006) framework to highlight syntactic and semiotic aspects of algebraic concepts that appear in both algebra and programming, such as the equal sign, variable, algorithm and function.

Study 2: Teachers' voices

The second study focuses on the transposition from *Knowledge to be taught* into *Taught knowledge* (Figure 1). Data is gathered from teachers who are in the process of implementing programming in their mathematics classrooms. In the analysis we focus on what didactical choices teachers make and why, as well as what opportunities, challenges and pitfalls they identify. At present, we have data from two sources; a) teachers' written documentations of lesson studies (c.f. Fernandez & Yoshida, 2012) and b) individual teacher interviews with early adopters.

Lesson study plans and teachers' written reports of enacted lesson studies were collected from 24 groups of teachers, attending an in-service development programme; in total approximately 135 primary and secondary school teachers. The teachers involved in the programme came with mixed ability and motivation towards teaching programming as part of the mathematics curriculum. Each lesson study consisted of two or three cycles of co-planning, enacting and revising a lesson about programming in mathematics. In some of the lesson studies the lesson was taught by the same teacher in each cycle, in others by different teachers. The written documentations have been scrutinized to identify types of activities and programming environments used, as well as challenges and questions raised by the teachers. These results will be instrumental in helping us identify themes for focus group interviews further on.

Individual semi-structured interviews have been made with "early adopters", i.e. teachers who actively teach programming at an early stage of the implementation and who identify themselves as enthusiastic about bringing programming into mathematics lessons. The early adopters were recruited through our teacher education networks. At present 19 interviews of around 30 minutes each, covering teachers from grades one to nine, have been conducted, audio-recorded and transcribed. Some of the teachers will later participate in a second round of interviews. Using NVivo software, a content analysis is being conducted, relating interview data to theoretical definitions of AT and CT. What mathematics

and what types of activities teachers choose to present in their classrooms as well as how they justify their choices will be analysed as a means of understanding the transposition of knowledge (Chevallard, 2006). Some tentative results are presented here.

Tentative results of Study 1: Teaching materials and textbooks

The tentative results of Study 1 show that there are differences regarding syntax and semantics between programming and algebra that may cause problems for students' algebra learning. In our initial analysis of government-provided teaching material available online (Kilhamn & Bråting, 2019), the different meanings of the equal sign and the concepts variable and algorithm have been discussed in detail. For instance, in algebra the equal sign is normally used as a *relational* operator. Therefore, it would be meaningless to write $a = a + 1$ since it is not true for any value of a. Meanwhile, in programming the same expression is understood as the *assignment* "add 1 to the value a" which is often used when a program needs to loop through a range of consecutive integers. Instead, the double equal sign (==) holds a relational meaning in programming. These kinds of differences can afford the development of algebraic thinking through contrasting examples and awareness of accuracy, or constrain it if the teacher is unaware of the different experiences students have. In addition, we need to take into account that the equal sign already causes problems in school mathematics since students tend to interpret it as an operator symbol (4 + 3 *make* 7) rather than a relation (c.f. Kieran, 1981).

The initial analysis of textbooks in mathematics for grades 1-6 reveal that the implemented content is similar in the different textbook series, but it has been included in different ways. Regarding the latter, some textbook publishers have revised all textbooks in mathematics in order to include programming and computational thinking. Meanwhile, other publishers have offered most of the new content online as supplementary material and in teacher guides. Regarding the programming content, the textbook series seem to have focused on two of the three dimensions of CT suggested by Brennan and Resnick (2012): the *concepts* pattern, loop and algorithm and the *thinking practices* stepwise instructions, iteration, repetition and debugging. There is a high correspondence between the textbooks' content and the prescribed content in the revised 2018 curriculum document. However, we cannot identify almost any connection between algebra and programming in the textbooks. Programming content is either added as separate chapters, or integrated in already existing chapters of arithmetic, statistics, geometry and problem solving. We find this result interesting given that a major part of the programming content in the 2018 curriculum document is included within the core content of algebra.

Tentative results of Study 2: Teachers' voices

Our tentative results from analyses of the lesson study documentations show that, as expected, activities vary according to grade in the school system, but also seem to vary depending on the programming environment and language that teachers have chosen to use. An unexpected finding concerns students' engagement as teachers decide to use different programming environments: features in block programming could become distractive for some students through its abundance of side effects and aesthetics to control, while textual programming languages seem to conform education. Some teachers in the lesson studies prefer the latter, as it is easier to manage in the classroom, in spite of the challenges in symbol use and logic when students struggle with the syntax. As an example, a conditional can, in block programming, be written in many ways enhancing students' creativity and motivation, while in a textual setting students are obliged to follow specific predefined structures.

Another finding relates to semantic differences in different programming environments. For example, in unplugged activities arrows pointing up, down, left and right signify points on the compass while on a robot they signify forward, backward, and which way to turn. Another example is the meaning of variables in textual languages such as Python (see also Kilhamn & Bråting, 2019).

The types of activities and choices of programming environments described in our interviews with early adopters mirror those found in the lesson studies. Scratch, Code.org and Python are most frequently used. The preferred source of inspiration and ideas for most of the early adopters is social media, where they participate in special groups for mathematics teachers. Rather than using textbooks or publisher-produced digital teaching aids, they choose internet-based environments that are free of charge and easily accessible. Although most of them have taken a basic course of 7.5 ECTS credits in programming for teachers, only a few of them have a profound skill in programming themselves, enabling them to create their own programming activities.

We find that the early adopters have quite different ideas about what to teach and why. They express uncertainty about what types of activities and environments to include in the teaching of programming, for example, if activities with spreadsheets or interactive geometry programs such as GeoGebra should count as programming or not. They all agree that programming increases motivation in mathematics, which will hopefully affect students' learning of mathematics. The activities they describe include a limited variety of mathematical content, the most common being movement or lines and figures in a coordinate system, calculations, probability or statistics. Working with patterns is quite common, but mostly those activities are limited to finding a repeated pattern that could be coded as a loop. Connections to algebra are scarce in the activities and not particularly highlighted

in the interviews. When specifically asked, variables are brought up by some as the link to algebra.

Discussion

In conclusion, it could be said that teachers are in a challenging situation. As students move through the grades different programming environments will be used, causing them to learn and relearn the semantics and syntax of symbols. In addition, we identify challenges connected to aspects of learning symbols and concepts that appear in both programming and mathematics with slightly different meanings and syntax. We therefore consider teachers' knowledge about these matters tremendously important. Our preliminary analyses of teaching materials, textbooks, lesson study documentations and interviews indicate that the didactic transposition of knowledge concerning aspects of programming and computational thinking is shaky and diversified, that teachers take an active part in it through social media, but at the same time expose themselves to influences they may not be able to view critically.

The emphasis on social media as a source for ideas and inspiration about how to fulfil the new curricular demands implies two things. One is that there is obviously a very active community of teachers who interact and help each other. The other is that there is a lack of authority in decisions about what to teach and why to teach it. Moreover, the difference between commercial marketing and peer support is not always clear.

The teachers who now teach programming in schools are educated mathematics teachers with no, or very limited, programming skills. From the interviews with early adopters it is clear that such skills are necessary in order to see potentials and pitfalls well enough to create activities suitable for specific learning goals. In particular, moving from a procedural use of algorithms to a conceptual focus on the structure of algorithms, i.e. connecting CT with AT, is not possible with limited knowledge of programming. Furthermore, the difference between "using digital tools" and "teaching programming" is not always clear. For some teachers, programming essentially means coding, focusing the first two dimensions in Brennan and Resnick's (2012) framework; computational concepts and thinking practices. For others, it seems to be an overarching concept incorporating the third dimension, that is the computational perspectives expressing, connecting and questioning, along with skills in handling digital tools.

We also note that the integration of programming into the core content of algebra in the Swedish national curriculum is not mirrored in the transposition process. Neither textbook authors nor teachers seem to see the connection clearly, and programming activities in mathematics more often deal with other mathematical content. In addition, the focus on programming per se seems stronger

than the focus on mathematical concepts in our study, in a similar way as described by Hickmott et al. (2018).

According to Nouri et al. (2019) and the teachers they interviewed, the so called 21st century skills could be taught with block programming, i.e. Scratch, but according to the teachers in our study, it is not that simple, as block programming also brings a tension between students' self-expression and teachers' management. We agree with Nouri et al. (2019) that teachers need increased knowledge about different programming concepts, but would also like to add the need of a thorough understanding of how different mathematical concepts, e.g., variables can be used and denoted differently in different programming environments. Therefore, in the next stage of the project, we will scrutinize to what extent teachers are aware of the affordances of mathematical symbols and concepts in relation to Brennan and Resnick's (2012) framework.

Acknowledgment

This work was supported by the Swedish Research Council [Grant no. 2018-03865]. We would like to thank the Ifous group at Stockholm University for valuable cooperation.

Notes

1.https://larportalen.skolverket.se/#/moduler/1matematik/Grundskola/L%C3%A4rare%20i%20matematik

References

Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832–835.

Berry, M. (2013). *Computing in the national curriculum. A guide for primary teachers*. Bedford: Computing at School.

Bocconi, S., Chioccariello, A., & Earp, J. (2018). *The Nordic approach to introducing computational thinking and programming in compulsory education*. Report prepared for the Nordic@BETT2018 Steering Group.

Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. Paper presented at *the 2012 annual meeting of the American Educational Research Association*, Vancouver, Canada.

Brousseau, G. (1997). *Theory of didactical situations in mathematics*. Dordrecht: Kluwer.

Bråting, K., Madej, L., & Hemmi, K. (2019). Development of algebraic thinking: Opportunities offered by the Swedish curriculum and elementary mathematics textbooks. *Nordic Studies in Mathematics Education, 24*(1), 27–49.

Chevallard, Y. (2006). Steps towards a new epistemology in mathematics education. In M. Bosch (Ed.), *Proceedings of the Fourth Congress of the European Society for Research in Mathematics Education, CERME 4* (pp. 21–30). Barcelona: FUNDEMI IQS-Universitat Ramon Llull.

Duval, R. (2006). A cognitive analysis of problems of comprehension in a learning of mathematics. *Educational Studies in Mathematics, 61*, 103–131.

Fernandez, C., & Yoshida, M. (2012). *Lesson study: A Japanese approach to improving mathematics teaching and learning.* New York: Routledge.

Futschek G. (2006) Algorithmic Thinking: The Key for Understanding Computer Science. In: R.T. Mittermeir (Eds.), *Informatics Education – The Bridge between Using and Understanding Computers. ISSEP 2006. Lecture Notes in Computer Science,* vol. 4226. Springer, Berlin, Heidelberg.

Grover, S., & Pea, R.D. (2013). Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher, 42*(1), 38–43.

Hickmott, D., Prieto-Rodriguez, E., & Holmes, K. (2018). A Scoping Review of Studies on Computational Thinking in K–12 Mathematics Classrooms. *Digital Experiences in Mathematics Education*, *4*(1), 48–69.

Kamii, C., & Dominick, A. (1997). To teach or not to teach algorithms. *The Journal of Mathematical Behaviour, 16*(1), 51–61.

Kang, W., & Kilpatrick, J. (1992). Didactic transposition in mathematics textbooks. *For the Learning of Mathematics, 12*(1), 2–7.

Kaput, J. (2008). What is algebra? What is algebraic reasoning? In J. Kaput, D. Carraher & M. Blanton (Eds.), *Algebra in the early grades* (pp. 5–18). New York, NY: Lawrence Erlbaum.

Kieran, C. (1981). Concepts associated with the equality symbol. *Educational Studies in Mathematics, 12*(3), 317–326.

Kieran, C. (Ed.). (2018). *Teaching and learning algebraic thinking with 5- to 12-year-olds.* Cham: Springer.

Kilhamn, C. & Bråting, K. (in press). Algebraic thinking in the shadow of programming. In U. T. Jankvist, M. van den Heuvel-Panhuizen, & M. Veldhuis (Eds.), *Proceedings of the Eleventh Congress of the European Society for Research in Mathematics Education, CERME11*. Utrecht, the Netherlands: Freudenthal Group & Freudenthal Institute, Utrecht University and ERME.

Kotsopoulos, D., & Floyd, L., Khan, S., Namukasa, I.K., Somanath, S., Weber, J., & Yiu, C. (2017). A Pedagogical Framework for Computational Thinking. *Digital Experiences in Mathematics Education, 3*(2), 154–171.

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). Computational thinking in K-9 education. In A. Clear & R. Lister (Eds.), *Proceedings of Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference* (pp. 1–29). New York: ACM.

Nouri, J., Zhang, L., Mannila, L., & Norén. E. (2019). Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Education Inquiry*. First online.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* New York, NY: Basic Books.

Swedish National Agency for Education. (2018). Läroplan för grundskolan, förskoleklassen och fritidshemmet, Lgr11. Stockholm: Fritzes.

Wing, J.M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33–35.