The Integration of Programming in Swedish School Mathematics: Investigating Elementary Mathematics Textbooks

Kajsa Bråting \textsuperscript{a} and Cecilia Kilhamn \textsuperscript{b}

\textsuperscript{a}Department of Education, Uppsala University, Uppsala, Sweden; \textsuperscript{b}Department of Pedagogical, Curricular and Professional Studies, University of Gothenburg, Gothenburg, Sweden

ABSTRACT
We characterize the recently included programming content in Swedish mathematics textbooks for elementary school. Especially, the connection between programming content and traditional mathematical content has been considered. The analytical tools used are based on the so-called 5E’s, a theoretical framework of action, developed within the ScratchMath project, and Brennan and Resnick’s (2012) terms computational concepts and practices. The result uncovers “follow a procedure” as the dominating action, in which the concepts stepwise instruction and repeated pattern were frequent. Bridging between programming and mathematics is weak in the sense that the programming content does not enhance the possibility to explore mathematical concepts and ideas.

ARTICLE HISTORY
Received 8 September 2020
Accepted 15 January 2021

KEYWORDS
Mathematics education; programming; textbook analysis; curriculum; digital tools

Introduction
During the past five years, programming has emerged as a new skill in several countries’ national school curricula across all grade levels. Along with a fast-changing and increasing digital world, programming and technology play an important role in several domains of the society (Nouri et al., 2020). Specifically, programming is often emphasized as a pedagogical tool for developing students’ computational thinking. Computational thinking can be explained as a process in which we describe, analyze and solve problems in a way that computers can assist or, as Aho (2012) defines it, “the thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms” (p. 832). The term computational thinking was introduced in the 1980s by Seymour Papert who developed “Logo”, the first programming language for children (Papert, 1980). The outcome of Papert’s ideas gradually disappeared, 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). In the present study, we focus on programming tasks since programming, but not computational thinking, is mentioned in the Swedish curriculum.\textsuperscript{1} Moreover, in this study programming is seen as an aspect of computational thinking and programming tasks as tasks in which computational thinking can be developed.

\textsuperscript{1}In other contexts “computational thinking” is in Swedish referred to as “datalogiskt tänkande”, but this concept is not included in the national curriculum document.
The integration of programming in school curricula has been realized in various ways in different countries, see for example England (Brown et al., 2014), Denmark (Misfeldt et al., 2020), Sweden (Kilhamn & Bråting, 2019) and the USA (Fisher, 2016). Although programming is closely related to mathematics it is, like physics, not traditionally thought of as part of mathematics. Therefore, it is also not evident that it should be included in school mathematics. In England and Denmark, programming has been made part of a new school subject (“Computing” and “Technology comprehension”, respectively), while for example Finland and Sweden have 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 syllabus as an aspect of algebra through all grade levels, which makes the Swedish case unique in an international perspective (Kilhamn & Bråting, 2019). This is somewhat surprising given that neither programming nor computational thinking is mentioned in the extensive research on algebraic thinking and learning from the past decades (e.g., Kieran, 2018).

The present study is part of an ongoing research project whose overall purpose is to contribute to international research regarding the implementation of programming in school curricula by investigating the Swedish case (Bråting et al., 2021). The project as a whole is theoretically embedded in Chevallard’s (2006) framework regarding transposition of knowledge, which describes a praxeology of what is taught and why it is taught and how this changes as knowledge is transposed to different levels of the educational system. When referring to mathematics in this article we mean the content traditionally taught within school mathematics, including subfields such as arithmetic, algebra and geometry. Programming is here not included in this definition of mathematics, although the current transposition of knowledge we attempt to analyze may entail a change in this respect over time.

Previously, the majority of the research on teaching and learning programming focused programming at university level (Heintz et al., 2016), but the ongoing implementation of programming in curricula across Europe has led to an overwhelming demand of research-based knowledge on programming in school (Blikstein, 2018). In the present study, we analyze the programming content that has recently been inserted in Swedish commercially produced textbooks in elementary school mathematics. The aim is to obtain a characterization of the programming content in the textbooks and discuss how this content may affect students’ opportunities to learn mathematics. In general, textbooks are important artifacts in the teaching and learning of mathematics as they often serve as the primary source for teachers when planning their lessons (Johansson, 2006). In Sweden, the role of mathematics textbooks is also used to individualize teaching by allowing students to work through their textbooks at their own pace (Neuman et al., 2015). Therefore, studying the programming content in mathematics textbooks will help us discover what kind of learning opportunities students are offered (Jablonka & Johansson, 2010). In this study, we investigate textbooks for elementary school (grades 1–6, age 7–12), which is relatively rare within the Swedish research field of mathematics education (see Bråting et al., 2019; Hemmi et al., 2019).

In order to classify the programming content and discuss the relation between programming and mathematics we have applied Brennan and Resnick’s (2012) terms computational concepts and practices as well as Benton et al.’s (2017) framework for action, the so-called 5E’s, as bases for our analytical tool. We pose the following two research questions:

1. What characterizes the programming content in Swedish textbooks in school mathematics?
2. In what ways do the programming tasks create a bridge between programming and mathematics?

**Background and Relevant Research**

In 2018, the Swedish national syllabus was revised with the goal of strengthening students’ digital competence (Bocconi et al., 2018). Digital tools were mentioned across the syllabus, whereas
programming was pointed out as new content in the school subjects mathematics and technology at all grade levels. Technology now includes learning to control objects by means of programming, while in mathematics “pupils should be given opportunities to develop knowledge in using digital tools and programming to explore problems and mathematical concepts, make calculations and to present and interpret data” (Swedish National Agency of Education, 2018, p. 58). Within the core content of algebra, stepwise instructions are introduced in grades 1–3, while in grades 4–6 the focus is on how algorithms can be created and used in programming. The revision was implemented with short notice and without a national teacher development plan. As a result, many mathematics teachers now teach programming without being proficient themselves since it was not part of their teacher training (Kilhamn et al., 2021; Misfeldt et al., 2019; Mozelius et al., 2019). Considering this, and the strong position of mathematics textbooks in Sweden, it is reasonable to believe that teachers will be strongly influenced by the textbooks concerning what and how they teach the new content.

Programming is not entirely new in the Swedish syllabus. In 1980, computer knowledge (datalära) appeared as a content topic in mathematics for grades 7–9 (Swedish National Agency of Education, 1980). While access to both hardware and software was limited, the primary goal was to teach students about computers rather than to engage in programming activities. Computers were introduced as digital tools, but came with many obstacles. They were large and required a lot of power, and programming languages were quite demanding, so coding was introduced only to some students and mainly at upper secondary level (Rolandsson, 2019). Although early enthusiasts saw programming as a problem-solving tool, reality turned it into a struggle with syntax during the 1990’s. Computer knowledge was removed from the syllabus in the next revision 1994, and did not reappear until 2018 as digital competence and programming.

Another concept that has moved in and out of the Swedish syllabus is algorithms. Before 1980, the term referred to traditional algorithms for addition, subtraction, multiplication and division, and was seen as a cornerstone of arithmetic. In the 1990’s, calculators became available and algorithms were replaced by an increased emphasis on mental calculations. The term algorithm was removed from the description of arithmetic in the mathematics syllabus in 2011, but re-inserted in 2017 in connection to programming (Kilhamn & Bråting, 2019; Swedish National Agency for Education, 2018). This change implies a shift of emphasis from mathematical algorithms to a computational use of the word. In mathematics, an algorithm is often defined as “[…] a finite sequence of executable instructions which allows one to find a definite result for a given class of problems” (Brousseau, 1997, p. 130). One example is the subtraction algorithm, which is based on the place value system. Using the pre-defined steps of the algorithm it is possible to solve all subtraction tasks. In computer science, an algorithm is less well-defined, although most definitions used in educational settings describe a computer algorithm as a finite sequence of instructions to compute correct answers (Benton et al., 2018). However, examples of typical introductory examples of algorithms within the context of programming often describe everyday activities that are far removed from arithmetic. Such algorithms are not generalizable outside the activity described, in contrast to mathematical algorithms which are always applicable to a class of problems (Bråting & Kilhamn, 2020).

In an international perspective, programming in schools was to a great extent driven by the introduction of Logo during the 1980s, with the ambition of using programming as a tool to explore and express mathematical ideas (Agalianos et al., 2006; Papert, 1980). A number of programming languages inspired by Logo appeared, along with digital tools such as calculators, spreadsheets and dynamic geometry (Noss & Hoyles, 1996). As these tools became increasingly user-friendly for mathematical exploration, programming aspects were backgrounded. Students became computer users, not computer programmers; consumers rather than designers. In response to that, programmers at Massachusetts Institute of Technology (MIT) developed a visual programming environment called Scratch (Resnick et al., 2009), where programming per se, rather than mathematics, was in the fore. In addition to unplugged activities, the large and quickly growing panoply of visual
environments, also referred to as block programming, have been embraced by educators and are currently shaping the frames around computing and programming in elementary schools in the Nordic countries (Bocconi et al., 2018).

A few studies have recently brought to light that links to mathematics are not always in mind when Swedish teachers talk about students learning programming. In two studies of early adopters, the teachers often referred to more generic competencies and twenty-first-century skills related to collaboration and communication (Nouri et al., 2020) or to programming as a potentially powerful pedagogical tool and as a way to enhance engagement (Kilhamn et al., 2021). Misfeldt et al. (2019) found that many teachers attending seminars about the introduction of programming in school mathematics could not see a clear relationship between mathematics and programming.

**Theoretical Frameworks**

To meet the surge for design principles, teaching strategies and assessment tools connected to programming activities in school, different frameworks have emerged. For example Brennan and Resnick (2012), two of the people behind the Scratch environment, developed a framework incorporating three key dimensions of computational thinking (CT) that students can develop when using Scratch. Based on a detailed study of three children designing activities in Scratch, Brennan and Resnick (2012) elaborated a framework describing the use of the computational concepts sequences, loops, parallelism, events, conditionals, operators, and data. Since this did not sufficiently reflect the children’s learning, four conceptual practices were added as a second dimension: (i) being incremental and iterative, (ii) testing and debugging, (iii) reusing and remixing, and (iv) abstracting and modularizing. Finally, a third dimension describing the children’s change in perspectives when working with Scratch was added. However, this dimension proved difficult to assess within the scope of this article and will therefore not be further discussed here.

Following the suggestion put forward by Wing (2006) and the development of Scratch (Resnick et al., 2009), the UK curriculum created a new school subject called Computing.² This caused mathematics education researchers in the UK to initiate the ScratchMath project with the aim of building mathematical knowledge by programming in Scratch.³ Although the project has not yet been fully reported, preliminary results and evaluations show that it is possible to achieve engagement with mathematical ideas while teaching computational thinking using Scratch (Hoyles, 2020). Together with teachers, the research team developed a framework called the 5E’s (Benton et al., 2016; 2017). The aim of the framework is to provide pedagogical strategies to guide the design of programming activities in mathematics from a constructionist perspective. Five constructs are included in the framework: Explore, Explain, Envisage, Exchange and bridgE:

**Explore:** To explore both the tool itself and the ideas that are present in the activity, allowing learners to try things out for themselves, work interactively and debug errors.

**Explain:** To explain and represent knowledge in multiple ways. Explaining a script and its intended outcome step-by-step will enhancing reflection on one’s own thinking. Defining and naming blocks are important features of explaining in a Scratch environment.

**Envisage:** To predict an outcome, to imagine what will happen before it happens and to initiate reflection before testing on the computer.

**Exchange:** To engage in explaining one’s strategies, questioning one another and resolving disagreements through pair work, collaboration and whole class discussions to promote creativity and exploration.

²[https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study](https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study).

³[https://www.ucl.ac.uk/ioe/research/projects/scratchmaths](https://www.ucl.ac.uk/ioe/research/projects/scratchmaths).
BridgE: To find links between programming and powerful mathematical ideas. In order to develop these connections, ideas need to be re-contextualised and rebuilt in the language of mathematics.

There are both similarities and differences between the above-mentioned frameworks; CT developed by Brennan and Resnick (2012) and the 5E’s developed by Benton et al. (2016; 2017). Both take their point of departure in Scratch. The practices described as the second dimension in the CT framework can largely be incorporated in the explore construct of the 5E’s. The computational concepts specifically pointed out in the CT framework are not mentioned in the 5E’s, which instead focus more on the learners’ actions of explaining, exchanging and envisioning. The most important difference between the two frameworks is the bridgE construct, which places programming activities clearly in a mathematical context. Powerful mathematical ideas need to be embedded in the activities and expressed in mathematical words and notation if mathematical learning is to take place. When looking for a framework for analyzing programming tasks in mathematics textbooks we find that component essential. If programming is to enhance mathematics learning it is also important to make the mathematics involved explicit. Since CT is a framework for thinking, and the 5E’s describes design principles, neither of them is automatically suitable for textbook analysis. Inspired by both of these frameworks, we therefore constructed our own analytical tool as described in detail below.

Method

We have conducted a qualitative content analysis with some quantitative elements (Bryman, 2012) of programming tasks in Swedish mathematics textbooks. In this section, we present the data material, the analytical tool and the procedure for the data analysis.

Data Material

Programming has been included in published teaching materials in different ways. Some have published a greater part of this content on the web and a smaller part in the printed textbooks, while others have done the opposite. There is a difference between grade levels, where mathematics textbooks for grades 1–3, in general, have included much more programming content in the printed textbooks compared to grades 4–9. In addition, some publishers have released separate textbooks in programming with more or less connection to mathematics, especially for the higher grades. Within the limits of one article it was not possible for us to include all the different teaching materials. We have restricted our analyses to printed mathematics textbooks because (1) programming is there included in a well-known mathematical context, and (2) web material is revised and changed at a much higher rate than printed textbooks, which can be expected to be in use for a longer period of time.

Previous studies have shown that there are significant differences between different Swedish textbook series in mathematics (Bråting et al., 2019). Therefore, we have included several textbook series in order to increase the possibility of finding different kinds of programming content. In total, we have analyzed 390 programming tasks from four different textbook series in mathematics for grades 1–6. Programming is not included in all textbooks within each textbook series, in total we found programming content in 23 of 56 textbooks. The following textbook series have been analyzed in the study:

- **Favorit matematik**, grades 1–6, published by Studentlitteratur
- **Mondo matematik**, grades 1–6, published by Gleerups
- **Singma matematik**, grades 1–5, published by Natur & Kultur
- **Prima matematik**, grades 1–6, published by Gleerups

Three of the textbook series have inserted programming as small separate sections within a broader chapter. The title of these sections is often “Programming” but in some cases “Programming and pattern”. Only Singma has inserted programming as separate chapters, although the...
programming chapters are smaller than the other chapters. “Favorit matematik” is the only textbook that applies the term computational thinking, a term which is not included in the syllabus (Swedish National Agency of Education, 2018).

**Analytical Tool and Data Analysis**

Initially, we tried to analyze the programming sections of the mathematics textbooks using the two frameworks described earlier separately. However, we found that neither of them could describe the content in a sufficiently comprehensive way. The CT framework (Brennan & Resnick, 2012) did not highlight mathematical ideas, and the 5E’s (Benton et al., 2017) did not sufficiently elaborate on computational concepts and practices. In addition, we found many tasks that did not include any of the CT practices or the 5E’s constructs. The reason for this could be that both frameworks describe an ideal situation—what we as researchers would like learners to engage in and develop knowledge about. The textbooks include other types of tasks, which also need to be categorized. We ended up combining the two frameworks to construct an analytical tool consisting of actions and concepts, and then discuss bridging mathematics and programming as an overarching idea.

**Actions**

We categorized the type of actions students were asked to carry out in the tasks. In some of them we found traces of the CT practices debugging (c) and being incremental and iterative (b, d) and the 5E’s constructs explore (b, c), explain (e), and envisage (f), but particularly the first action, to follow a procedure emerged from our data.

(a) Follow a procedure [Följa] – follow stepwise instructions, repeat or continue a pattern.
(b) Figure out [Finna regel] – work out the procedure, the rule or the pattern that generates an outcome, for example a number sequence.
(c) Find mistakes [Felsöka] – debug a code.
(d) Form and create [Forma och skapa] – give instructions, create a pattern, write code, represent with symbols.
(e) Explain [Förklara] – explain using natural language, using words to describe a procedure, a rule, a pattern or a concept.
(f) Envisage [Föreställa sig] – predict what will happen, reflect on possible outcomes when conditions or values are changed.

**Concepts**

Our analytical tool includes two types of concepts that are found in the tasks: mathematical concepts and computational concepts4 (see Tables 1 and 2). The analysis of concepts in the tasks is a descriptive analysis where concepts are identified and classified. When identifying concepts we look at the words used and the meanings these words are given in the context at hand.

We classify a concept as mathematical if it is found in traditional school mathematics and conveys an important mathematical idea. While a mathematical idea may be difficult to define, it is expressed in the mathematical concepts and representations used. Concepts, on the other hand, are often well-defined and connected to specific words and representations. In the analysis, mathematical concepts that appear explicitly in the tasks are first compiled and then categorized on an aggregated level. For example, one task (see Figure 5) is about drawing a square, including instructions to repeat 4 times and turn 90°. Since the mathematical terms are used to describe and construct a geometrical object, they are here classified as geometrical concepts.

---

4In Swedish textbooks and curricula computational concepts are referred to as programming concepts (programmeringsbegrepp).
A concept is classified as computational if it is a concept that bears a specific meaning within the presented programming context. Such concepts are for example code, loop, conditional and bug. Some words, for example algorithm, have slightly different meanings within a mathematical context and a programming context and could therefore be treated as different concepts (Bråting & Kilhamn, 2020). We have classified algorithm as a computational concept if it is defined as such and/or refers to an algorithm used to achieve a specific goal without being applicable to a class of (mathematical) problems. On the rare occasions when a mathematical algorithm is referred to in these tasks it always concerns arithmetic, e.g., the subtraction algorithm, and is therefore classified as an arithmetical concept. Naturally, mathematical algorithms may also appear in other sections of the textbooks not labeled as programming and not included in this analysis. Since the Swedish syllabus introduces stepwise instructions three years before the term algorithm is brought in, we decided to treat stepwise instructions as a concept in its own right. However, as we shall see in the result section, the two concepts seem to merge into one.

Bridging Mathematics and Programming

Finally, we look at the combination of actions and concepts to see in what way any explicit links between programming and mathematics are made or could potentially be made, and discuss in what way these links could lead to mathematics learning. In particular, we look at if learners are asked to reformulate computational ideas using mathematical terminology or traditional mathematical notation, and if the task allows them to explore mathematical ideas that are new to them. Naturally, teachers could highlight links that are not explicit in the textbooks, we can only say what we see in the activities as they are described in the textbook.

When deciding on the unit of analysis, we looked closely at the sections including programming content. Since the textbooks’ introductions as well as descriptions of new concepts are very short and the worked examples are clearly reflected in the tasks, we decided that tasks were more suitable as a unit of analysis, as compared to pages or square centimeters of pages (Valverde et al., 2002). Each task was characterized in terms of the actions included in the task, and all computational concepts and mathematical concepts that were explicitly mentioned in the tasks were identified. Sometimes an instruction or solved example was given at the top of the page followed by several tasks without repeating the instructions. For such tasks, the introductory instruction was included as part of the description of the task.

In order to strengthen the validity of the analysis, both author 1 and 2 analyzed one of the textbook series independently of each other and compared the results. The results were almost the same, and the few discrepancies were discussed until a common view was agreed on. Author 1 then analyzed the other textbooks and discussed any ambiguities with author 2 to reach agreement.

In the following two sections we report the results of the analysis. In the section “Results” we consider the characterization of the programming content (Research question 1) based on actions and concepts. Then, in the section “Bridging between programming and mathematics” we discuss the combination of actions and concepts in the light of Research question 2.

Results

We begin by considering the result of the analysis based on the actions and thereafter the concepts. Textbook examples are given to specify the analysis.

Actions

In total, we identified programming content in 390 tasks (241 for grades 1–3 and 149 for grades 4–6). Figure 1 shows the distribution of number of tasks over the six actions. Observe that a single task can be categorized with more than one action. Grades 1–3 is represented in blue and grades 4–6 in orange.
Figure 1 shows that the distribution of the actions is similar in the two grade spans with slightly more tasks in the first grade span. Only “Envisage” is more common in grades 4–6. The most common action is “Follow a procedure”, which characterizes the activity in more than half of the tasks. A typical example of such a task is given in Figure 2, where the student is requested to draw the robot’s path in a grid based on the stepwise instructions to the left. Moreover, there are many similar tasks in the textbooks where the students themselves are supposed to create stepwise instructions such as those given in Figure 2. That kind of task is categorized as the second most common action “Form and create”. Another common task identified as “Follow a procedure” is to repeat a given pattern, as in Figure 6(a) which will be described later on.

The third most common action, “Figure out” is often identified in tasks where the students are asked to find a rule to continue a pattern. The pattern could either be represented by a number sequence, for example 1, 4, 7, 10 … or as a repeated pattern represented by geometrical objects in different colors (see Figure 6(b)). In tasks categorized as “Explain” the students are most often asked to explain codes to each other.

The actions “Find mistakes” and “Envisage” are the least common actions. The tasks identified as “Find mistakes” mostly consist of finding bugs in a given code, but in a few cases the students themselves are requested to write a code that includes a bug. The latter kind of task has been categorized as both “Find mistakes” and “Form and create”. The action “Envisage” only appeared in 8 of the 390 tasks. One example is when students are asked to create a code from a given set of instructions and thereafter imagine what will happen if some of the instructions change places.

![Figure 1](image1.png)  
**Figure 1.** The distribution of number of tasks over the six actions.

![Figure 2](image2.png)  
**Figure 2.** Example of “Follow a procedure” in a task for grade 1 (Favorit matematik 1B, p. 187). Translation: Follow the instructions. Draw the robot’s path to the parcel. Go 2 boxes → Go 1 box, etc.
Concepts

We begin to consider the computational concepts identified in the tasks. As Table 1 reveals, stepwise instructions is clearly the most common computational concept in the textbooks for grades 1–3, where it is contained in more than half of the tasks. Stepwise instructions is also a frequently appearing concept in grades 4–6, where it occurs in more than one-third of the tasks. A typical task containing stepwise instructions is given in Figure 2.

The most common computational concept in the grade 4–6 textbooks is algorithm, which appears in more than half of the tasks. As Table 1 shows, the term algorithm is not included at all in the textbooks for grades 1–3. This result is reflected in the Swedish syllabus where stepwise instructions is emphasized in the core content of algebra for grades 1–3 and algorithm in the core content of algebra for grades 4–6 (Swedish National Agency of Education, 2018). The definition of “algorithm” is similar in the different textbook series, emphasizing step-by-step instructions:

An algorithm is a collection of detailed instructions for solving a specific task. In programming, the word algorithm is used to describe step-by-step instructions for a computer. (Singma 4B, p. 178)

In Matematik Favorit 4A an algorithm is explained as “an exact step-by-step description with instructions [...] kind of like a recipe” (p. 202). It is clear from these two examples that stepwise

| Computational concepts          | Tasks in Grades 1–3 | Tasks in Grades 4–6 |
|--------------------------------|---------------------|---------------------|
| Stepwise instructions          | 134 (56%)           | 54 (36%)            |
| Algorithm                      | 0 (0%)              | 75 (50%)            |
| Loop, iteration, repetition    | 69 (29%)            | 32 (21%)            |
| Rule                           | 51 (21%)            | 0 (0%)              |
| Code                           | 50 (21%)            | 9 (6%)              |
| Condition                      | 7 (3%)              | 30 (20%)            |
| Bug, debugging                 | 10 (4%)             | 7 (5%)              |

Figure 3. Example of “algorithm” in a task for grade 4 (Singma 4B, p. 175). Translation: Pair each bracelet with the correct algorithm. 5 (2 red, 1 orange, 1 pink) Start – Repeat – red, orange, pink.
instructions and algorithm merge into the same concept, merely introducing the term “algorithm” in grades 4–6.

A typical example of a task that involves the concept algorithm is given in Figure 3, where the student is asked to pair each bracelet with the correct algorithm. The algorithm is written in two different ways. The one to the left is written in a symbolic way, introduced by this specific textbook series in grade 2, and should be interpreted as “repeat 5 times the sequence: 2 red, 1 orange, 1 pink”. Here, the focus is on the whole algorithm rather than the single steps in the instructions, as in the task involving stepwise instructions for grade 1 in Figure 2.

The concept loop appears in about a quarter of the tasks in grades 1–3 and 4–6 respectively (Table 1). Figures 3 and 5 show typical examples of tasks that include loops. The occurrence of the concept condition increases between the grades 1–3 and 4–6, while the concept code decreases between the two grade levels. We notice that code can have different meanings both within and between the textbook series. In some cases, code corresponds to instructions in a programming language and in other cases to a secret message which is supposed to be decoded, more like a cipher. An example of the latter is given in Figure 4 where the task is to “solve the code” in order to find out the names of the animals.

![Figure 4. An example of decoding in a task for grade 1 (Matematik Favorit 1A, p. 185). Translation: What are the names of Kurre’s friends? Solve the code. In the pictures you can find Taggis, Kvack, Mimmi, Nyfiken, Amina, Nicko.](image)

Let us now move on to consider the mathematical concepts in the textbooks. In total, we identified mathematical concepts in 211 of the 327 tasks that included programming; 127 in grades 1–3 and 84 in grades 4–6. Table 2 shows the number and the proportion of the programming tasks in grades 1–3 and 4–6 that include the identified mathematical concepts.

![Table 2. Number and proportion of tasks in which each identified mathematical concept is included.](table)

| Mathematical concepts                        | Tasks in Grades 1–3 | Tasks in Grades 4–6 |
|----------------------------------------------|---------------------|---------------------|
| Patterns (including number sequences)        | 115 (48%)           | 16 (11%)            |
| Geometrical concepts (triangle, circle, etc.)| 9 (4%)              | 15 (10%)            |
| Arithmetical concepts                        | 5 (2%)              | 21 (14%)            |
| Rotation                                      | 4 (2%)              | 17 (11%)            |
| Coordinate plane                              | 0 (0%)              | 14 (9%)             |

*Number sequences occurred in 23 of the tasks in grades 1–3 and 5 of the tasks in grades 4–6. All the other patterns were visual, for example colors or shapes.
Table 2 reveals that the dominating mathematical concept within the programming tasks in grades 1–3 is pattern, which appears in almost half of the tasks but considerably less in grades 4–6. This is somewhat surprising considering that the concept pattern is emphasized in the mathematics syllabus for both grades 1–3 and 4–6. We discuss patterns in more detail further on. Geometrical concepts appear in some tasks, mainly as instructions to draw a geometrical figure, as in Figure 5. The occurrence of arithmetical concepts, such as number properties, divisibility, place value and binary numbers, differs between the grade spans, where the appearance is more frequent in grades 4–6 than in 1–3. Two other mathematical concepts, rotation and coordinate plane, mainly appear in grades 4–6, which is in line with the syllabus.

Bridging Between Programming and Mathematics

Now we move on to look at the combination between actions and concepts to operationalize the idea of bridging mathematics and programming, focusing tasks including the most common computational concept and the most common mathematical concept.

Taking Steps in a Grid

Let us first consider tasks that include stepwise instructions which were typical examples of the action “Follow a procedure”. Most of these tasks do not explicitly involve any mathematical concepts. In grades 1–3, they often make use of a grid, such as in Figure 2, where the concept stepwise instructions is instantiated by a robot moving around in the grid. Perhaps one may argue that a grid is a mathematical concept and a way to introduce students to matrixes or to the coordinate plane. However, in a grid, the focus is directed to the squares and not the nodes. The programming tasks do not emphasize the ideas of rows and columns, nor do they describe location in terms of coordinates. Hence, the combination of “follow a procedure” and stepwise instructions in these textbook tasks does not bridge to mathematical ideas or enhance mathematical problem-solving. We also notice that the steps in the stepwise instructions are not associated with how the instruction is divided into smaller pieces, but rather with the robot actually “taking steps” within the grid. This could be seen as a pitfall concerning students’ development of computational thinking, if the term “steps” is not later dissociated with walking about.

Some of the textbooks make explicit connections between programming and a mathematical topic, such as the subsection “Programming geometrical objects” in Matematik Favorit 5A. Figure 6 shows an example of a task from this subsection. The task consists of finding out which geometrical object will be formed when the student follows the stepwise instructions to the left. It is tempting to interpret this as a clear bridge between programming and geometry, but what does it create an opportunity to learn? The task includes the action “Follow a procedure” and the mathematical concept square. The connection between the two is the loop in the algorithm that repeats a 90° angle four times and moves the same distance each time. The task is in a textbook for grade 5, when a square will be a familiar geometrical object. The role of the mathematical concept in this task is therefore to create a context in which students can learn the syntax of block programming. There is a small element of recontextualizing into the language of mathematics since the student is asked to transform the code into a geometrical drawing. However, the task would bridge between these representations much more efficiently if the transformation was made both ways, for example by asking the student to draw another figure and then write the code for the new figure, thus adding the action “Form and create”. If the task had included the action “Explain” by asking the students to explain what it is in the code that produces the square, or the action “Envisage” by asking what would need changing in the code to make a different figure, then there would have been an opportunity to explore geometrical objects and a clear bridge between programming and mathematics.
We notice another pitfall in relation to mathematics when we compare the use of the grid in Figures 2 and 5. In both tasks the word “rutor” is used. In Figure 2 it refers to the inside of the box, in Figure 5 it refers to the distance between two nodes. This could cause confusion in relation to understanding the number line and the coordinate plane.

**Patterns in Programming and Algebra**

In the Swedish mathematics syllabus, programming is included in the core content of algebra. Further, algebra in grades 1–3 includes patterns and equalities, and in grades 4–6 patterns, unknown numbers, expressions and equations. Given the emphasis on patterns in the syllabus, it is not surprising that pattern is the most common mathematical concept included in the analyzed programming tasks. Figure 6(a) gives a typical example of a task included in a programming section for grade 1, where the students are asked to repeat the given pattern, categorized here as “Follow a procedure”. In this task, there is no explicit connection to any computational concept. In fact, this kind of task has appeared frequently in Swedish mathematics textbooks before the implementation of programming in mathematics (Bråting et al., 2019). Why it is suddenly labeled as a programming task is not apparent, so we do not see a bridge between programming and mathematics here.

A similar task is given in Figure 6(b), except that the students are also asked to describe the pattern with a programming code. Here the link between the mathematical idea of a repeated pattern is bridged with the programming concept loop, since they have to identify the part in the pattern that is repeated and describe it with a code.

The patterns that appear in these programming tasks rarely go beyond a repeated pattern. A few instances of linear number sequences are found in the tasks but on the whole, through all grade levels the concept of pattern as a repetition is consolidated. Again, the role of the mathematical concept seems primarily to be to create a context in which students can learn programming, rather than using programming to explore mathematical concepts.

**Discussion**

One of the most characteristic features of the textbooks’ tasks was the frequent use of stepwise instructions. This is no surprise, given that the term “stepwise instructions” is explicitly addressed in the mathematics syllabus for grades 1–3 (Swedish National Agency of Education, 2018). However, in an international perspective “stepwise instructions” is not a generally accepted term. It is a more colloquial word for square than the mathematic word “kvadrat”.

---

5Rutor in Swedish means boxes or squares. It is a more colloquial word for square than the mathematic word “kvadrat”.

---

**Figure 5.** Example of a programming task connected to geometry in grade 5 (Matematik Favorit 5A, p. 162). Translation: Follow the instructions. What is the name of the geometrical object created? Place the tip of the pencil at the starting point on the paper. Repeat 4 times. Move the pencil 2 boxes. Turn 90° counterclockwise. Remove the pencil from the paper. Do like this: Mark a starting point in your book.
for instance never mentioned among Brennan and Resnick’s (2012) computational concepts, nor within the ScratchMaths project (Benton et al., 2016). In the Swedish syllabus, there is a progression from first introducing stepwise instructions in grades 1–3 to focusing on algorithms in grades 4–6. This differs from the English computing syllabus where the term “algorithm” is introduced from the very beginning (Benton et al., 2018). It is not clear to us what the benefits of the strong emphasis on “stepwise instructions” are, especially since these textbook tasks often connected to the literal meaning of steps by taking steps in a grid or in a room, rather than dividing a problem into smaller pieces.

In mathematics, an algorithm is a general method for solving a given class of problems (Brousseau, 1997). In computer programming in elementary school, as it is represented in our data, an algorithm is connected to the representation of stepwise instructions to achieve a specific goal, not necessarily a mathematical problem and rarely a class of problems. The results mirror our findings when analyzing government-provided teaching materials (Bråting & Kilhamn, 2020) and are well in line with the Swedish syllabus. By removing the term algorithm from arithmetic and inserting it into the section on programming in the syllabus, a transposition of the what and the why of algorithms has occurred. Instead of being a mathematical idea for efficient calculations based on the structure of the place value system, an algorithm has become a computational idea of stepwise instructions given as a code. For a professional computer programmer, these ideas are merged, but in the transposition to the level of knowledge to be taught at school (Chevallard, 2006), the concept of algorithm becomes diluted. Also Benton et al. (2018) highlight algorithms
as a particularly challenging concept for both pupils and teachers, pointing to the limited characteristics of the concept visible in what they call typical lessons in primary school (in England). The framework they developed for designing programming tasks in mathematics addressed this by introducing the constructs “envisage” and “explain” using multiple forms of representation. In our data, the strong focus on the action “follow a procedure” and the limited occurrence of the actions “envisage”, “find mistakes” and “explain” indicate that Swedish students meet the same limited version of algorithms. Given the difficulties Swedish teachers have in making connections between mathematics and programming (Kilhamn et al., 2021; Misfeldt et al., 2019; Nouri et al., 2020), we suggest a clearer distinction between computer algorithms and an algorithm as a mathematical concept. Teachers need to understand both what they have in common and in what ways the concept has different connotations in the different fields.

There are some features that are fundamental for learning programming, whose appearance we found surprisingly low in the textbooks. One of them is debugging, which is emphasized in both the frameworks that inspired our analytical tool. Considering the recurrent idea that students can learn mathematics through the process of testing and debugging (Papert, 1980), it is reasonable to believe that debugging would have been emphasized when including programming in mathematics’ textbooks. We also notice that the tasks did not give any opportunity to learn about defining and naming blocks/instructions, which is fundamental for a programmer and also highlighted within Benton et al.’s (2016) action “explain”. We see great potential to broaden the types of programming tasks used in Swedish textbooks, where in particular the actions “explore”, “envisage” and “find mistakes” could serve as tools to enrich the programming content and strengthen the bridge to mathematics by, for instance, exploring geometrical objects or learn mathematics by testing and debugging.

Apparently, the textbooks’ tend to focus heavily on a few concepts and actions but almost leave out others that are fundamental for learning programming. One possible explanation to this is that, in Sweden, programming is implemented in two already existing school subjects; mathematics and technology. In doing so, the focus is shifted from what is crucial for learning programming to ensuring that the new content is relevant for learning traditional mathematics (and technology). A typical example is that tasks dealing with loops are most often connected to the mathematical concept pattern rather than the computational concept condition.

In the Swedish syllabus, programming in mathematics is described as a core content in algebra, emphasizing the concepts stepwise instruction and algorithm. In addition to the issues already discussed, the connection to algebra is confounding. In our data, the tasks often include geometry or arithmetic, but apart from the repeated patterns, there are few connections to algebra. The use of variables is of course a tacit ingredient that will become explicit when they move on to text programming in higher grades, but the concept is not explicit in the textbooks for grades 1–6. Working with variables would be a way of making the algorithms applicable to a class of problems rather than used to solve a specific problem, thus bridging the gap between programming and mathematics (see also Bråting & Kilhamn, 2020). More tasks including the action “envisage” could achieve this, for example by posing questions like “What would happen if the variable is changed?” and “Under what conditions, or for what values of the variable, will the algorithm produce what we want?” The goal, as expressed in the syllabus, is after all not to learn programming per se, but to use programming to explore problems and mathematical concepts, to make calculations and to present and interpret data.

Let us conclude our discussion by addressing some limitations of our study and how we can continue our research in the future. Although research has shown that textbooks have a great impact on mathematics teaching both in Sweden and in an international perspective (e.g., Jablonka & Johansson, 2010), we have to bear in mind that there are limitations connected to textbook studies. Studying the content in mathematics textbooks helps us to discover what kind of learning opportunities students are offered, but we cannot draw any conclusions about how the textbooks affect students’ mathematical performance (Hiebert & Grouws, 2007). Moreover, we have restricted our study to printed mathematics textbooks. Several textbook publishers also offer web-based programming materials or textbooks solely about programming, which are not included in this study. We do
not know to what extent teachers are actually using the programming content in the printed textbooks, which is important in light of the explosive growth in the amount of teaching materials on the internet. In parallel to this study, we have recently started to investigate what programming materials teachers are using and how they are using it (Kilhamn et al., 2021), which we will continue studying in the future.

Disclosure Statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the Swedish Research Council [grant number 2018-03865].

ORCID

Kajsa Bråting http://orcid.org/0000-0002-8169-5670
Cecilia Kilhamn http://orcid.org/0000-0003-2294-4996

References

Agalianos, A., Whitty, G., & Noss, R. (2006). The social shaping of logo. Social Studies of Science, 36(2), 241–267. https://doi.org/10.1177/0306312706053809
Aho, A. V. (2012). Computation and computational thinking. The Computer Journal, 55(7), 832–835. https://doi.org/10.1093/comjnl/bxs074
Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2016). Building mathematical knowledge with programming: Insights from the ScratchMaths project. In A. Sipitakiat & N. Tutiyaiphungrprasert (Eds.), Constructionism in Action 2016, Conference Proceedings (pp. 25–32). Sukasattana Foundation.
Benton, L., Hoyles, C., Kalas, I., & Noss, R. (2017). Bridging primary programming and mathematics: Some findings of design research in England. Digital Experiences in Mathematics Education, 3(2), 115–138. https://doi.org/10.1007/s40751-017-0028-x
Benton, L., Kalas, I., Saunders, P., Hoyles, C., & Noss, R. (2018). Beyond jam sandwiches and cups of tea: An exploration of primary pupils’ algorithm-evaluation strategies. Journal of Computer Assisted Learning, 34(5), 590–601. https://doi.org/10.1111/jcal.12266
Blikstein, P. (2018). Pre-college computer science education: A survey of the field. Google LLC. Retrieved May 2020 from https://goo.gl/gmSiVm
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.
Bråting, K., & Kilhamn, C. (2020). Exploring the intersection of algebraic and computational thinking. Mathematical Thinking and Learning. Advance online publication. https://doi.org/10.1080/10986065.2020.1779012
Bråting, K., Kilhamn, C., & Rolandsson, L. (2021). Integrating programming in Swedish school mathematics: Description of a research project. In Y. Liljekvist, L. Björklund Boistrup, J. Häggström, L. Mattsson, O. Olander, H. Palmér (Eds.), Sustainable mathematics education in a digitalized world. Proceedings of MADIF12. SMDF.
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.
Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In Proceedings of the 2012 Annual Meeting of the American Educational Research Association (pp. 1–25). American Educational Research Association.
Brousseau, G. (1997). Theory of didactical situations in mathematics. Kluwer.
Brown, N., Sentance, S., Crick, T., & Humphreys, S. (2014). Restart: The resurgence of computer science in UK schools. ACM Transactions on Computing Education, 14(2), 1–22. https://doi.org/10.1145/2602484
Bryman, A. (2012). Social research methods. Oxford University Press.
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). FUNDEMI IQS-Universitat Ramon Llull.
Fisher, L. (2016). A decade of ACM efforts contribute to computer science for all. Communications of the ACM, 59(4), 25–27. https://doi.org/10.1145/2892740
