Algorithmic Abstraction in Computer Science Curricula for Primary Schools: The Case of a National Curriculum for 4th Grade

Mor Friebroon-Yesharim
Department of Science Teaching,
Weizmann Institute of Science
mor.friebroon@weizmann.ac.il

Ronit Ben-Bassat Levy
Department of Science Teaching,
Weizmann Institute of Science
ronit.ben-bassat-levy@weizmann.ac.il

Michal Armoni
Department of Science Teaching,
Weizmann Institute of Science
michal.armoni@weizmann.ac.il

ABSTRACT

We currently conduct an extensive research project that investigates the implementation of a national computer science (CS) curriculum for the 4th grade. This research explores the curriculum's evolution, from the policymakers' visions, through the formal (intended) curriculum, teachers' training, and the actual implementation in class, to the educational outcomes (attained curriculum). As part of this research, we identified the educational goals of the policymakers who initiated this curriculum and examined how their goals were reflected in the formal curriculum. This paper focuses on one of the policymakers' goals – the development of algorithmic abstraction. To this end, we developed a method to investigate how algorithmic abstraction is addressed in the formal curriculum. In a recent paper, we focused on one facet of this method. Here, we will complete the presentation of this method by describing its two other facets and present the corresponding findings. Our findings indicate that the formal curriculum emphasized programming, whereas the more abstract concept of an algorithm was not sufficiently stressed. This was reflected in three ways: Most of the algorithmic problems included in the curriculum were presented using programming-related terms; algorithms and algorithm-related terms comprised a notably smaller part of the curriculum; and CS concepts were mostly introduced in terms of programming. We generalized our method to obtain a framework for assessing CS introductory curricula through the lens of algorithmic abstraction. Since abstraction is widely acknowledged as a CS fundamental idea, such a framework has significant merit for the CS educational research community.

CCS CONCEPTS

• **Social and professional topics** → K-12 education; Computer science education.

KEYWORDS

Algorithmic abstraction, Primary school, Curricular analysis, K-12

ACM Reference Format:
Mor Friebroon-Yesharim, Ronit Ben-Bassat Levy, and Michal Armoni. 2023. Algorithmic Abstraction in Computer Science Curricula for Primary Schools: The Case of a National Curriculum for 4th Grade. In *The United Kingdom and Ireland Computing Education Research (UKICER) conference (UKICER 2023), September 07, 08, 2023, Swansea, Wales Uk*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3610969.3611154

This work is licensed under a Creative Commons Attribution-NoDerivs International 4.0 License.

UKICER 2023, September 07, 08, 2023, Swansea, Wales Uk
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0876-3/23/09.
https://doi.org/10.1145/3610969.3611154

1 INTRODUCTION

Over the past decade, the number of countries interested in teaching computer science (CS) to primary school students and even younger ones has increased [31]. Among the common reasons mentioned are to (1) foster an accurate image of CS [2], (2) promote positive attitudes toward CS [2], (3) increase the motivation and intention to pursue CS at higher age levels (as a means to widen the pipeline [e.g., 10]), and (4) develop students' computational thinking (CT) [4]. We find the teaching of abstraction in CS, even to such young learners, as highly important, since it aligns with all these reasons: Abstraction is acknowledged as a major CT component [39]; it is also widely acknowledged as the most fundamental CS [13]; hence, any attempt to foster an accurate image of CS must also include an introduction to abstraction [36]. Moreover, any change in attitudes, motivation towards, and intentions regarding CS should be based on a reliable introduction to the discipline and its nature, including abstraction. Obviously, the introduction of abstraction should be adapted to the students' age. This is not an easy task, since abstraction in CS is a complex and inherently abstract idea. Nevertheless, it is possible, embracing Bruner's educational spiral teaching approach, according to which "Any subject can be taught effectively in some intellectually honest form to any child at any stage of development" [8, p. 33].

In Israel, a national CS curriculum for primary schools was announced by the Ministry of Education in October 2016 and was first implemented in 300 schools in the 2017/18 school year. During the 2018/19 school year, its implementation was extended to include about 500 schools. The curriculum was planned for two school years, in the 4th and 5th grades. Most of the teachers who were appointed to teach the new subject did not have an adequate CS background, formal or otherwise.

We currently conduct a wide research project in which we use the framework of curricular analysis [38] to examine how the 4th-grade CS curriculum evolved through all the layers of curriculum representations and manifestations, starting with the policymakers' visions, and ending with the students' learning outcomes. Several themes, conceptual as well as affective, are examined. The role of abstraction as an educational goal has clearly emerged from the policymakers' statements, which explicitly referred to the importance of teaching abstraction at this young age. Since abstraction was set as a goal by the policymakers and since, as noted above, we

believe that any CS curricular initiative should address it, the treatment of abstraction throughout the evolution of the curriculum is among this projects' main themes. Here, we will focus on this theme and one of the first curricular layers, examining the reflection of one form of abstraction – algorithmic abstraction – in the formal 4^{th}-grade (age 9-10) CS curriculum, as a case study. Then, we will generalize the assessment method developed for this evaluation, to a framework for assessing the reflection of algorithmic abstraction in CS introductory curricula.

The paper is organized as follows: Section 2 discusses related work and Section 3 presents the research questions. Section 4 contextualizes the study reported here in the larger research project. Sections 5 and 6 describe the methodology and findings, respectively. Finally, Section 7 discusses the findings and the general framework.

2 RELATED WORK

2.1 Teaching CS to young students

Many countries now include CS as a primary school subject [12] (e.g., the UK [35], Slovenia [27], Slovakia [28], Australia [15], and New Zealand [14]). Corresponding curricula and standards (see [12] for a European survey) focus on introducing the students to the basic concepts in CS and algorithmics (although many also include topics related to digital literacy, which are outside the discipline of CS). The starting age varies; for example, in New Zealand [5] and the UK [35], CS education starts at the age of five, whereas in Lithuania, it starts in the 5^{th} grade [12]. In most of the countries in which CS is taught in primary school, it is a compulsory subject, although in some countries it may become an elective subject at higher grade levels. Because of the worldwide interest in teaching CS in primary school, this is a very active research area, dealing with issues such as students' learning outcomes, students' difficulties, and teachers' difficulties [e.g., 27, 29].

2.2 Teaching abstraction to young students

Abstraction is a cognitive means of thinking about an idea or a concept at different levels of detail, as necessary, from high levels of abstraction, where one ignores details and focuses on the "big picture", to low levels of abstraction, where one delves into very small details. Solving even basic algorithmic problems should involve transitions between levels of abstraction, from abstract to concrete and vice versa [22]. As was noted above, as the essence of CS [13] and in line with Bruner's framework of spiral teaching and its implications regarding fundamental ideas [8], abstraction should be addressed in K-12 CS curricula already from the beginning of the spiral education process [1, 37]. Numerous publications have reported on teaching and learning abstraction and have discussed students' difficulties in learning abstraction for both undergraduate [e.g., 32, 33] and K-12 students [e.g., 20,21]. The challenge of teaching CS abstraction is probably even greater for primary-school students since their abstraction abilities may still be immature and underdeveloped [34].

Several initiatives for teaching abstraction to young children have been reported [e.g., 17,18]. In addition, since CT has been acknowledged as an important educational goal, and abstraction has been acknowledged as a major aspect of CT, several studies on teaching CT to K-6 students have addressed abstraction [e.g., 9]. However, in the context of CT, abstraction has often been interpreted in a narrow sense that ignores some of its aspects. For example, Brennan and Resnick operationally defined abstraction as modularization only [7]. We aimed at addressing abstraction in the context of primary-school students as a wider and deeper idea.

Hazzan's theoretical framework of abstraction levels [23] has proved very relevant for studies dealing with abstraction in CS [e.g., 19, 24]. According to Hazzan, as an effective means of decreasing the cognitive effort required to learn a new concept, students tend to first perceive it at a lower level of abstraction than the level required to grasp it fully. This may be a transient learning phase; however, students may also remain in it, thus limiting their understanding of the concept. Following Hazzan, the 4-level hierarchy of Perrenet et al. [32, 33] (hereafter referred to as the PGK hierarchy) was designed to assess undergraduate students' perception of an algorithm (i.e., a computational solution). Each of the four levels represents a cognitive abstraction level in which one can grasp the idea of an algorithm; however, all levels are necessary for an appropriate perception of this idea. The levels are as follows:

Level 4: The highest level is the Problem level. At this level, one can reason about an algorithm in terms of the problem it solves and its characteristics. In this phase, an algorithm is viewed as a black box, a perspective that calls for a high level of abstraction.

Level 3: At the Algorithm (Object) level, an algorithm is viewed as an object that is not associated with a specific programming language. At this level, complexity measures such as time and space become relevant.

Level 2: At the Program (process) level, an algorithm is viewed as a process, which is described by a specific executable program written in a specific programming language.

Level 1: The lowest level, the Execution level, interprets an algorithm as a specific run on a specific input and a specific concrete machine.

These levels also describe the conceptual levels that one undergoes (back and forth) when solving algorithmic problems; hence, the entire hierarchy reflects the idea of algorithmic abstraction [1]. Therefore, it can also be used to assess students' perceptions of it, and as a guide for teaching it [1]. Note the double use of abstraction in this hierarchy: it consists of cognitive levels of abstraction and expresses the conceptual idea of algorithmic abstraction.

Following Perrenet et al., several studies have used the PGK hierarchy to examine the perception and use of algorithmic abstraction by high-school and middle-school students [e.g., 30, 36 respectively]. It was also used regarding the primary-school curriculum, focusing on teachers, where Level 3 (Algorithm) was mostly interpreted as a preliminary design phase [40].

2.3 The national primary-school CS curriculum

The design of the national Israel CS curriculum for primary schools began in 2016. It is planned for two school years starting in the 4^{th} grade, with an elective extension module for the 6^{th} grade. The goals of the curriculum are to foster a correct image of the discipline of CS and promote the acquisition of conceptual CS knowledge as well as major CS skills. The curriculum does not include digital literacy, and it aims to teach more than just programming [26]. The

curriculum is planned for 60 hours per year, 2 hours per week. In the 4th grade, the curriculum utilizes Scratch and in the 5th grade it involves robotics activities combined with Scratch. Specifically, in the 4th grade, the curriculum includes the following topics: introduction to algorithmics, input and output, logical expressions, variables, conditional execution, and iterative execution. Since the curriculum uses Scratch, it also covers concepts related to event-driven programming, such as communication between agents and concurrency [26].

The entire curriculum was published by the Ministry of Education in a very comprehensive document, which also includes worked examples and assignments for teachers' use. Despite its length, it is not a textbook, but instead, a resource that introduces the teachers to the curriculum and supports them in using it. The 4th-grade curriculum consists of about 90 pages.

3 RESEARCH QUESTIONS

The above rationale leads to the following research questions:

(RQ1) How is algorithmic abstraction reflected in the Israeli 4th-grade CS curriculum?

(RQ2) Which indicators can be used to assess the extent to which algorithmic abstraction is reflected in a CS introductory curriculum?
The research was approved by the IRB of the Weizmann Institute of Science and by the Ministry of Education.

4 RESEARCH CONTEXT

As noted above, teaching abstraction was set as an educational goal by the policymakers. Specifically, an analysis of the policymakers' statements (which was conducted in an earlier phase of the research project, reported elsewhere) identified three objectives that concern abstraction: 1) the idea of abstraction should be emphasized; 2) the curriculum should use data structures to teach data abstraction and develop skills for working with multiple levels of data abstraction; 3) The curriculum should aim at teaching algorithmic abstraction, and develop skills for working with multiple levels of algorithmic abstraction. As reflected in our research questions, this paper focuses only on the third objective of algorithmic abstraction. The assessment of the first two objectives will be reported elsewhere; here we can say that its outcomes indicated that the formal document of the 4th-grade national CS curriculum did not address the first two objectives to a satisfactory extent.

5 METHODOLOGY

We now turn to the inspection of the formal curriculum from the perspective of algorithmic abstraction. The data for our analysis included the 90 pages of the 4th-grade formal curriculum document. We employed the qualitative method of document analysis, which is highly applicable for case studies and combines content analysis and thematic analysis [6]. We also employed Chi's quantification of qualitative findings [11].

Since the policymakers explicitly mentioned algorithmic abstraction and levels of abstraction, we started by looking for an explicit mentioning of these concepts, but none was found. Then we turned to examine how algorithmic abstraction is reflected in the curriculum document. As mentioned in Section 2, several studies employed the PGK hierarchy to assess the learning of algorithmic abstraction and its teaching. Since this hierarchy captures the essence of algorithmic abstraction [1], and specifically the movements between levels of abstraction, we found it appropriate for our purposes. We decided to use it to analyze the curriculum regarding three different facets:

A unified perspective, which finds the relative volume of each of the PGK levels within the 4th-grade curriculum document. These values express the implicit emphasis regarding each level. They also express the relative extent to which a user of this curriculum is exposed to each level. For example, in a curriculum that is mostly written at the Programming level, the Algorithm level is hardly addressed; thus, it is unlikely to induce effective teaching of the Algorithm level or encourage students to work at that level when appropriate.

A conceptual perspective, which examines the abstraction-related treatment of CS concepts: For each CS concept included in this curriculum, we identified the PGK levels used when referring to this concept. For example, the control structure of conditional repetition can be introduced for the first time as an algorithmic control structure, or as the Scratch block "Repeat".

A problem-solving perspective: The extent to which the problem-solving processes included in this document employed and exhibited algorithmic abstraction.

Here we focus on the first two facets. The third one is reported in [16]. We started with a preliminary content analysis of the entire curriculum document (Facet 1) in which the categories related to the research questions, in our case to the employment and reflection of algorithmic abstraction. Utilizing the PGK hierarchy, we used its four levels as a deductive category system, which is strongly related to algorithmic abstraction. The preliminary analysis then integrated the thematic approach, in which the themes recognized inductively in the data become new categories. Upon stabilization, we obtained a set of categories, based on which we coded the entire document, for Facets 1 and 2, respectively.

During the preliminary analysis, we noted the existence of all four PGK levels. In addition, an emerging theme was inductively identified. The problems included in the curriculum (as worked examples or assignments) can be divided into two groups: The first group included problems that were described using general terms, independent of programming-related details. Such problems can be *understood* by CS laymen, even if they do not know how to *solve* them. According to [1], when teaching abstraction in CS, it is best to use such problems, since they better demonstrate the Problem level of the PGK hierarchy, in which a problem is detached from a specific solution and certainly from a programmed implementation of a solution. Such problems also demonstrate better the nature of CS, as a discipline that deals with different kinds of problems, and the authentic process of CS problem solving. The second group included problems that were explicitly described using Scratch terms. For example: "Write a script in which upon receiving the message 'Hi' from the dog, the cat says, 'Hello'". In contrast, a similar problem but at a higher level of abstraction is: "Create an animation in which the dog and the cat are in the yard. When the dog says 'Hi', the cat says, 'Hello'". Although the first example is an algorithmic problem, it cannot be considered as a manifestation of the highest PGK level. It is too close to the Program level and does

not emphasize the gap between the levels in terms of abstraction. It also might encourage the students to move directly from the problem to programming, skipping the algorithm phase.

This theme yielded a new category. The higher-level problems were assigned to the original category of the Problem level, whereas the other problems, reflecting a lower abstraction level, were assigned to a new category, the Program-Problem level.

Our category set is a revised version of the PGK hierarchy, hereafter referred to as PGK*, which is described and exemplified in Section 5.1. Thus, our assessment method consists of four components: a category set based on the PGK* hierarchy, and three analysis protocols that use this set, a *unified* document analysis of the entire text, a *conceptual* document analysis, and a *problem-solving-related* document analysis. We will now elaborate on the first two analyses corresponding to Facets 1 and 2, respectively.

Unified document analysis: We coded each text segment by assigning it to the PGK* abstraction level that it manifests (using the level names as tags). A text segment could be a picture, a sentence, or where needed, segments of sentences that are fully contained at a specific level of abstraction, and any expansion of them would violate this condition. Then, following Chi's method [11] for quantifying qualitative data, we counted the segments coded by each level name. However, considering that the text segments could be of a different length and that the longer use of a certain abstraction level may be more effective than a shorter one, we counted the contribution of each segment as its relative length. A balanced normalized distribution between the abstraction levels would indicate that all levels are equally represented in the text, thus exposing the students to the full-fledged concept of algorithmic abstraction. A distribution in which the higher levels are represented more in the text, compared with the lower ones, indicates an emphasis on the higher levels of algorithmic abstraction, which students often tend to skip [32, 33]. The latter is consistent with the recommendations of [1] for teaching CS abstraction to novices. In contrast, a higher representation for lower levels indicates that exposure to higher levels of CS abstraction is limited, potentially encouraging students to work at lower levels.

Conceptual content analysis: We examined each of the CS concepts included in the curriculum and used the PGK* levels to categorize their appearances. We quantified this categorization to obtain for each CS concept the distribution of its appearances across the different abstraction levels, thus operationalizing the treatment of the CS concepts in terms of abstraction. Ideally [1], we would expect to see the introduction and treatment of a concept moving back and forth between all levels of abstraction, starting with the highest, and with many occurrences at the higher levels.

These analyses were performed by the first author. Then, the second author performed an independent analysis of more than 50% of the data. Disagreements were discussed and ideally resolved, reaching 75% agreement. Then, the third author, who is an expert in the field, joined the discussion. Finally, all disagreements were discussed among all three authors, and a full agreement was reached, after which the entire text was analyzed again from all perspectives based on the agreements that were reached. Coding was performed using Atlas.ti software. The coding process is exemplified in the following section for each of the abstraction levels. The results are presented in Section 6.

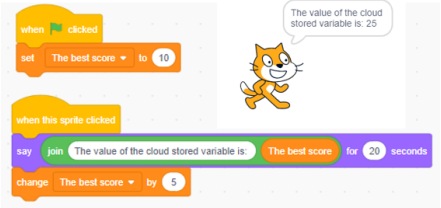

Figure 1: An example of the Execution level

5.1 The PGK* hierarchy

As noted above, following the inductive analysis, we extended the PGK hierarchy by adding a new level, the Program-Problem level. Since this level blends the Problem level with programming, one could choose to include it either below or above the Algorithm level. We chose to define this as the new Level 4, between the Problem level and the Algorithm level, thus emphasizing problem solving, since often the Problem level does not receive sufficient attention [1]. This is the new Hierarchy:

Level 1: The Execution level
Level 2: The Program level
Level 3: The Algorithm level
Level 4: The Program-Problem level
Level 5: The Problem level

We will demonstrate the categorization to the five abstraction levels by providing for each level an example or two of segments that were coded at this level.

The Execution level includes the results of executing Scratch code or any other reference to Scratch code execution.

For example, in Figure 1, the right side depicts the stage of the Scratch environment, as it looks following the execution of the scripts on the left side. This figure was coded by the tag Execution.

Similarly, the sentence, "It is important to practice with the students a change in the order of the script's instructions and ask them to see which instructions were executed, in which order, and how the change influenced the script's outcomes" was coded by Execution.

The Program level includes direct or indirect references to Scratch code or Scratch blocks that did not involve execution. For example, Figure 2, taken from the curriculum, presents the scripts designed as part of a worked example. They were coded by the tag Program.

Similarly, the sentence, "It is important to emphasize and internalize that 'TRUE' and 'FALSE' are constant values in the language", was coded by the tag Program.

The Algorithm level includes algorithmic descriptions (which do not involve Scratch blocks) of a solution or a part of one, as well as any references to algorithms and any discussion on solving a problem or a sub-task that does not involve Scratch code.

For example, Figure 3, taken from the curriculum, presents an example of a simple algorithm.

Similarly, the sentence, "Write with the students what is required of each player in the world of the problem, which variables we define

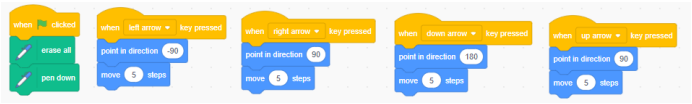

Figure 2: An example of the Program level

1. counter <= = 0
2. repeat **'forever'**
   2.1 **'say'** "I am in an endless loop"

Figure 3: An example of the Algorithm level

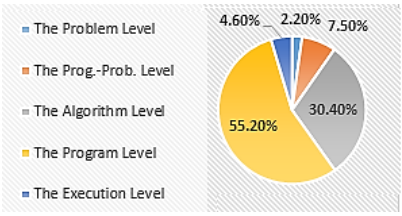

Figure 4: Distribution of the segments by levels of abstraction

for each player, which states each player has, and how they change in the process", was coded by the tag Algorithm.

The Program-Problem level: A problem description that included references to components of the Scratch language was coded at this level. For example, the following sentence, taken from a worked example, was coded by the Program-Problem tag: "Write a script in which two sprites slide towards each other".

The Problem level: A problem description given in natural language without any references to CS concepts (which may be used for solving it) or Scratch terminology was coded by this level. Note that such a problem description may be followed by a hint of the solution, which may include such references; however, in such cases the hint was not considered as part of the description, but rather, as a new text segment. For example, the following sentence, taken from a worked example, was coded by the tag Problem: "The fox tries to reduce the distance between him and the duck. The duck follows the mouse cursor. Will they meet?"

6 FINDINGS

Sections 6.1 and 6.2 present the results of the unified and conceptual analyses, respectively.

6.1 The unified perspective

Figure 4 depicts the results obtained from coding all the segments by the level names of the PGK* hierarchy, after accounting for their length. The Problem level occurred very infrequently. In contrast, more than half of the segments were at the Program level.

Figure 5: The gradient representing the different levels of abstraction

Table 1: The percentages of the number of occurrences in each of the abstraction levels for three main CS concepts

	The Problem Level	The Prog.-Prob. Level	The Algo. Level	The Program Level	The Exec. Level
Events	1.23	11.11	8.64	65.43	13.58
Concurrency	0	15.79	10.53	68.42	5.26
Variable	3.82	5.1	28.66	49.68	12.74

6.2 The conceptual perspective

We examined each of the 73 CS concepts mentioned in the 4th-grade curriculum document. For each concept, we coded each of its occurrences according to the PGK*-level of abstraction it reflected, and then calculated the distribution of its occurrences over the five PGK* levels. Table 1 depicts the results for three of the 73 concepts: events, concurrency, and variable. The darker the shade of a cell in this table, the larger the number of occurrences that were coded at the level corresponding to this cell (Figure 5). Namely, a shade depicts the relative frequency of an abstraction level. For example, the concept concurrency has 68.42% occurrences at the Program level, 15.79% at the Program-Problem level, 10.53% at the Algorithm level, 5.26% at the Execution level, and no occurrences at the Problem level.

Although Table 1 provides only three rows out of 73, there is a high resemblance between these rows and most of the other 70. To illustrate this, Figure 6 provides another view of the 73-row table. It has five charts, one for each of the five shades of green. Each chart presents the distribution of the concepts by the abstraction level with the corresponding shade. For example, Figure 6(a) corresponds to the darkest shade; it presents the distribution of concepts by their most frequent level of abstraction. In this chart, all three concepts included in Table 1 belong to the Program level, which is the darkest shade level for these three concepts. Similarly, in Figure 6(b), corresponding to the second-darkest shade or the second most frequent abstraction level, the concept 'events' belongs to the Execution level, whereas the concept 'variable' belongs to the Algorithm level. Note that the number of concepts decreases from (a) to (e), since most concepts did not appear at all five levels. We can see that most of the concepts tended to appear at the Program level,

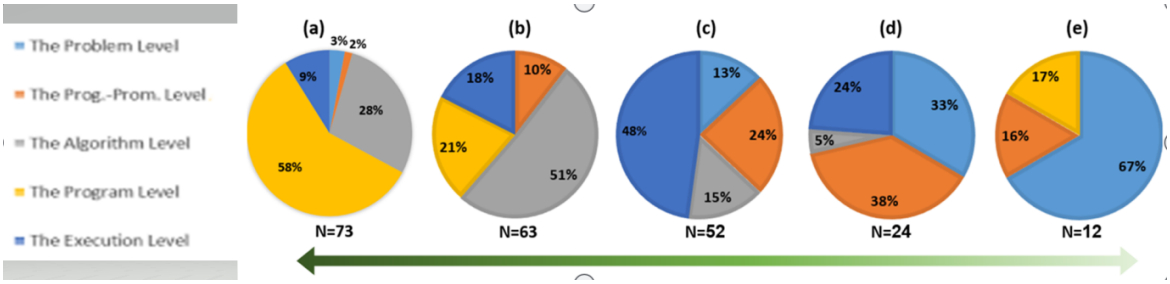

Figure 6: The distribution of each shade of green over the abstraction levels

followed by the Algorithm level, with relatively few occurrences at the other abstraction levels.

7 DISCUSSION AND CONCLUSIONS

We investigated how CS abstraction was reflected in the formal CS curriculum for 4th grade. We focused on abstraction since the policymakers perceived it as a curricular learning goal and because as a fundamental idea of CS, it should be taught spirally, from early stages [8]. The policymakers referred to algorithmic abstraction and to working with multiple levels of algorithmic abstraction. However, our findings regarding the reflection of algorithmic abstraction in the 4th-grade curriculum indicate an imbalanced treatment. The highest level of the PGK* hierarchy was hardly used. Less abstract descriptions of problems, which refer to solutions or the Scratch language, were more prevalent but were still among the least frequent levels. Hence, apparently this curriculum fails to take advantage of the opportunity to encourage the students to take the time to understand the problems at hand, as recommended in [1], and it greatly prefers program-related problem descriptions over more general and abstract ones. The curriculum does not neglect the concept of an algorithm, which comprises about 30% of the entire text; nevertheless, the level of programming is much more frequent, comprising more than half of the text. To emphasize the more abstract level of an algorithm (in line with [1]) and to encourage the students to devote their time and effort to this level, which they often tend to skip [32, 33], one might expect a more balanced treatment, which emphasizes the importance of the higher level. In general, a balanced distribution that covers all levels exposes the students to the full-fledged concept of algorithmic abstraction. In this curriculum the levels that relate to programming or the programming environment are represented more, potentially encouraging students to work at these levels.

The conceptual perspective supports these findings, since the distributions of the concepts over the abstraction levels were very unbalanced, tending towards the lower ones. The two levels of Program and Algorithm were both prevalent; however, the Programming level was much more dominant.

Overall, this curriculum tends towards programming. Since it is the basis of the CS spiral, any inclination of students towards programming or any misconceptions that equate CS with programming are likely to affect the later stages of the spiral and be harder to resolve (for both students and teachers). We study the effect of this tendency in other parts of our research project.

The curriculum hardly mentions the term 'abstraction' explicitly. Indeed, CS abstraction and abstraction levels can be taught without explicitly using this professional term. Perhaps this would even be better for such young students (for example, in [36], the term 'algorithm' was replaced by 'verbal description'). However, bearing in mind that the formal curriculum was intended for teachers, one would expect a more professional treatment.

These results provide the answer to RQ1, concerning this specific curriculum. However, the contribution of the systematic inspection of this formal CS curriculum is way beyond this specific case of one curriculum in one country. It portrays a general framework for assessing curricula in terms of their treatment of algorithmic abstraction, thus also illuminating the way and extent to which these curricula reflect the nature of CS and its most fundamental idea of abstraction. It also highlights the challenges in developing a new curriculum and the pitfalls to which one should pay special attention, since in those pitfalls, ideas may be lost, narrowed, or misinterpreted, and hence may be absent from later layers of the curriculum, potentially missing some educational goals. This framework provides the answer to the RQ2, where the indicators correspond to the reflection of the PGK* levels for each of the facets. This framework may also be applicable to introductory CS textbooks, to assess their reflection of procedural abstraction.

The generality of this framework is rooted in PGK*. Note that the addition of the new level has emerged inductively from the data; hence it characterizes this specific curriculum. In other curricula, the thematic document analysis that starts with the 4-category set of the PGK levels, might not find evidence for the Program-Problem level. Moreover, new levels may emerge. Thus, in the general framework, the identifier PGK* denotes 'the extension of the PGK hierarchy obtained by adding new levels that emerged during a preliminary curriculum document analysis.'

ACKNOWLEDGMENTS

This research was supported by the Israeli Science Foundation (ISF), Grant 556/18.

REFERENCES

[1] Michal Armoni, 2013. On teaching abstraction in CS to novices. *Journal of Computers in Mathematics and Science Teaching* 32(3), 265-284.
[2] Michal Armoni, and Judith Gal-Ezer ,2014. Early computing education – Why? What? When? Who? *ACM Inroads Magazine* 5(4), 54-59.
[3] Theophilus Azungah, 2018. Qualitative research: deductive and inductive approaches to data analysis. *Qualitative Research Journal* 18(4), 383-400.

[4] Valerie Barr and Chris Stephenson, 2011. Bringing computational thinking to K-12: what is involved and what is the role of the computer science education community? *ACM Inroads* 2(1), 48-54.

[5] Tim Bell, Peter Andreae, and Anthony Robins, 2014. A case study of the introduction of computer science in NZ schools. *ACM Transactions on Computing Education* (TOCE) 14(2), 1-31.

[6] Glenn, A. Bowen, 2009. Document analysis as a qualitative research method. *Qualitative Research Journal* 9(2), 27-40.

[7] Karen Brennan and Mitchel Resnick, 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* (Vol. 1), 25pp.

[8] Jerome S. Bruner, 1960. *The Process of Education*. Harvard University Press. Boston, MA.

[9] Ünal Çakiroğlu and İsak Çevik, 2022. A framework for measuring abstraction as a sub-skill of computational thinking in block-based programming environments. *Education and Information Technologies* 27(7), 9455-9484.

[10] Michael E. Caspersen, Ira Diethelm, Judith Gal-Ezer, Andrew McGettrick, Enrico Nardelli, Don Passey, Branislav Rovan, and Mary Webb, 2022. *Informatics Reference Framework for School*. https://www.informaticsforall.org/wp-content/uploads/2022/03/Informatics-Reference-Framework-for-School-release-February-2022.pdf

[11] Michelene T. H. Chi, 1997. Quantifying qualitative analyses of verbal data: A practical guide. *The Journal of the Learning Sciences* 6(3), 271-315.

[12] The Committee on European Computing Education (CECE), 2017. *Informatics Education in Europe: Are We All in the Same Boat?* ACM.

[13] Edsger W. Dijkstra, 1972. The humble programmer. *Communications of the ACM* 15(10), 859-866.

[14] Caitlin Duncan and Tim Bell, 2015. A pilot computer science and programming course for primary school students. In *Proceedings of the Workshop in Primary and Secondary Computing Education* (pp. 39-48).

[15] Katrina Falkner, Sue Sentance, Rebecca Vivian, Sarah Barksdale, Leonard Busuttil, Elizabeth Cole, Christine Liebe, Francesco Maiorana, Monica M. McGill and Keith Quille, 2019. An international study piloting the measuring teacher enacted computing curriculum (metrecc) instrument. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (pp. 111-142).

[16] Mor Friebroon-Yesharim and Michal Armoni ,2023. Abstraction and Problem Solving in a CS Curriculum for 4th Grade. In *Proceedings of the 7th APSCE International Conference on Computational Thinking and STEM Education* (pp. 19-24).

[17] Paul J. Gibson. 2008. Formal methods: Never too young to start. In *Proceedings of the Formal Methods in Computer Science Education Workshop* (pp. 151-160).

[18] Paul J. Gibson. 2012. Teaching graph algorithms to children of all ages. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* (pp, 34-39).

[19] David Ginat and Yoav Blau, 2017. Multiple levels of abstraction in algorithmic problem solving. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 237-242).

[20] Bruria Haberman, 2004. High-school students' attitudes regarding procedural abstraction. *Education and Information Technologies* 9(2), 131-145.

[21] Bruria Haberman, Haim Averbuch, and David Ginat, 2005. Is it really an algorithm: The need for explicit discourse. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (pp. 74-78).

[22] Juris Hartmanis, 1994. Turing award lecture on computational complexity and the nature of computer science. *Communications of the ACM* 37(10), 37-43.

[23] Orit Hazzan, 1999. Reducing abstraction level when learning abstract algebra concepts. *Educational Studies in Mathematics* 40(1), 71-90.

[24] Orit Hazzan (2003) How students attempt to reduce abstraction in the learning of mathematics and in the learning of computer science. *Computer Science Education* 13(2), 95-122.

[25] Orit Hazzan, 2008. Reflections on teaching abstraction and other soft ideas. *ACM SIGCSE Bulletin* 40(2), 40-43.

[26] Israel Ministry of Education, 2016. *The Israeli CS Curriculum for Elementary Schools* (in Hebrew).

[27] Franc Jakoš and Domen Verber, 2017. Learning basic programing skills with educational games: A case of primary schools in Slovenia. *Journal of Educational Computing Research* 55(5), 673-698.

[28] Martina Kabátová, Ivan Kalaš, and Monika Tomcsányiová, 2016. Programming in Slovak primary schools. *Olympiads in Informatics* 10(1), 125-159.

[29] Kyungbin Kwon, Anne T. Ottenbreit-Leftwich, Thomas A. Brush, Minji Jeon, and Ge Yan, 2021. Integration of problem-based learning in elementary computer science education: effects on computational thinking and attitudes. *Educational Technology Research and Development* 69(5), 2761-2787.

[30] Liat Nakar, 2023. Pattern-oriented instruction its practical application and the connection to various manifestations of abstraction in computer science. [Doctoral Dissertation, Weizmann Institute of Science].

[31] Michiyo Oda, Noborimoto Yoko and Horita Tatsusya, 2021. International Trends in K-12 Computer Science Curricula through Comparative Analysis: Implications for the Primary Curricula. *International Journal of Computer Science Education in Schools* 4(4), n4.

[32] Jacob Perrenet, Jan Friso Groote, and Eric Kaasenbrood. 2005. Exploring students' understanding of the concept of algorithm: Levels of abstraction. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (pp. 64-68).

[33] Jacob Perrenet, and Eric Kaasenbrood, 2006. Levels of abstraction in students' understanding of the concept of algorithm: the qualitative perspective. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (pp. 270-274).

[34] Jean Piaget, 1936. *Origins of Intelligence in the Child*. Routledge & Kegan Paul. London, UK.

[35] The Royal Society, 2012. *Shut down or restart? The way forward for computing in UK schools*. http://royalsociety.org/education/policy/computing-in-schools/report/.

[36] David Statter and Michal Armoni, 2020. Teaching abstraction in computer science to 7th grade students. *ACM Transactions on Computing Education* 20(1), 8:1-37.

[37] Fulya Torun and Arif Altun, 2022. The Effects of Instructional Environments and Cognitive Abilities on Abstraction Performance. *Psycho-Educational Research Reviews* 11(3), 656-674.

[38] Jan Van den Akker, 2007. Curricular development research as a specimen of educational design research. In Tjeerd Plomp and Nienke Nieveen (Eds.) *Curriculum Design Research* (pp. 52-71).

[39] Jeannette M. Wing, 2008. Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 366(1881), 3717-3725.

[40] Jane L. Waite, Paul Curzon, William Marsh, Sue Sentance, and Alex Hadwen-Bennett, 2018. Abstraction in action: K-5 teachers' uses of levels of abstraction, particularly the design level, in teaching programming. *International Journal of Computer Science Education in Schools* 2(1), 27pp.