To Be Or Not To Be... An Algorithm: The Notion According to Students and Teachers

Carlo Bellettini[*]
carlo.bellettini@unimi.it
Università degli Studi di Milano
Department of Computer Science
Milano, Italy

Violetta Lonati[*]
violetta.lonati@unimi.it
Università degli Studi di Milano
Department of Computer Science,
Lab. CINI "Informatica e Scuola"
Milan, Italy

Mattia Monga[*]
mattia.monga@unimi.it
Università degli Studi di Milano
Department of Computer Science,
Lab. CINI "Informatica e Scuola"
Milan, Italy

Anna Morpurgo[*]
anna.morpurgo@unimi.it
Università degli Studi di Milano
Department of Computer Science,
Lab. CINI "Informatica e Scuola"
Milan, Italy

ABSTRACT

We study how students and teachers conceptualize the notion of 'algorithm', a fundamental concept in computer science and computer science curricula. We analyze the work produced by CS students and teachers during a workshop conducted repeatedly over several years in some outreach activities for schools, computing education courses, and professional development opportunities for teachers. Participants were divided into groups, given some procedures written in natural language, and asked to decide together which of the procedures might be taken as algorithms. The procedures were purposely designed to present flaws or features that could activate discussion. After that, groups were asked to agree upon a definition of 'algorithm' and make its fundamental properties explicit. The activity triggered reflections around the idea of algorithm and its interpreter, going beyond stereotyped definitions, and leading participants to deepen their comprehension of the notion. We report on the aspects that were more debated by the groups, and those that were recurrent in the resulting definitions. We argue that this kind of activities should be offered more often both to students during their study career, and to teachers in professional development opportunities, to prompt them to reflect on computing foundations also in a non-technical, more holistic way.

CCS CONCEPTS

• **Social and professional topics → Computer science education**.

[*]All the authors contributed equally to this research.

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0423-9/24/03
https://doi.org/10.1145/3626252.3630950

KEYWORDS

Concept of algorithm, active learning, observational study, computer science education, teacher professional development

ACM Reference Format:
Carlo Bellettini, Violetta Lonati, Mattia Monga, and Anna Morpurgo. 2024. To Be Or Not To Be... An Algorithm: The Notion According to Students and Teachers. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630950

1 INTRODUCTION

The notion of 'algorithm' is a crucial one in computer science. David Harel, in a popular book intended for the general public [20], called the study of algorithms the "spirit of computing". In fact, to compute means to produce a result (some piece of information) by following a procedure in a way so unambiguous that could, in principle, be carried out by a machine. Computer science arose together with the effort of clarifying what it really means to describe a procedure "unambiguously", with a fascinating intellectual journey through the debate on the foundation of mathematics, the challenge of engineering programmable machines and the physics of information processing [5, 15]. However, the concept of algorithm often remains in the background of computer science studies. Even though algorithms are usually included in computer science degree programs, the courses associated with the term typically focus on the design and analysis of algorithms, discuss algorithmic techniques, present classic or application-specific algorithms with a focus on their correctness and efficiency [10]. A deepest reflection on the very notion of what an algorithm is (and what it is not) is usually proposed only in advanced theoretical courses.

As the notion of 'algorithm' is so pervasive, one could expect students to "absorb" it anyway through computer science study. We argue that this is often not the case and students should be instead prompted to reflect explicitly about the notion. In particular, in this paper we describe a classroom activity that has exactly this goal. The activity was proposed in a three-hour workshop that was conducted repeatedly over several years in computing education courses and outreach activities for K-12 schools. We qualitatively analyzed the work produced by students and teachers during the workshops, in order to answer the question: *how do students and teachers conceptualize the notion of algorithm?* We found that participants, despite feeling somewhat familiar with the concept of algorithm, often disagreed on whether to classify specific procedures as algorithms or not; in particular, the issue of ambiguity and the elicitation of the interpreter of the algorithm often emerge as a turning point for a deeper understanding.

By analyzing the discussions, we matured a belief that this kind of reflection should become commonplace in the study of computer science, even with non vocational goals, and at early stages. In fact, its value is not in the technical empowerment it enables, but in the cultural understanding of what computer science can (or cannot) do for our society. It is hard to imagine any principled discussion of ‘artificial intelligence’ (a current hot topic), without a clear and extensive notion of algorithm. Therefore, more activities on the notion of algorithm should be offered, both to students during their study career, and to teachers in professional development opportunities, in order to prompt them to reflect about about computing’s foundations and impact on our lives.

The paper is organized as follows. In §2 we discuss the related literature and in §3 we introduce our working definition of algorithm. In §4 we describe the workshop and related materials, and in §5 we present the method we used to analyze the works of the workshop’s participants. In §6 we report the findings of our analysis and discuss them in §7. In §8 we draw our conclusions.

2 RELATED WORK

In this paper we are interested in the abstract notion of ‘algorithm’, and in how students conceptualize it. The conceptualization of fundamental notions in education has been investigated in other scientific disciplines [31]. Specific examples are the notion of ‘number’ and the notion of ‘function’ in mathematics [37] or the concept of ‘force’ in physics [12, 28]. In particular, in [28] spontaneous representations of the concept of force produced by teachers were qualitatively analyzed.

More generally, several authors underline the important role of the conceptualization of fundamental notions in science and math education. Carey [9] argues that science education should aim at facilitating *conceptual change*, which occurs when learners move from common-sense, conflicting, and possibly misconceived knowledge to more scientifically-accepted conceptions. In the context of math education, Sfard [37] investigates the dual nature of mathematical conceptions: her theory of *reification* states that mathematical notions may be conceived in two fundamentally different, complementary ways: operationally as ‘processes’ and structurally as ‘objects’. The development of concepts corresponds to a transition from operational to structural conceptions, through the reification process.

Several studies in computer science education investigate how children or students conceptualize some specific computer science concepts. For instance [1] analyzes how students think of data structures; the conceptions of class and object by novice Java students is the subject of [14]; [35] and [2, 8] present literature reviews about the children’s conceptions of computers and of internet, respectively. In particular, many papers deal with the conceptualization of (at the time) “emerging” themes as: the internet [32], smartphones [6], social networks [7], or, very recently, artificial intelligence [39] and machine learning [30]. In these papers, children’s and students’ conceptions have been categorized with different methods. For instance [2] distinguishes between (1) intuitive, (2) elaborate and (3) misconception, whereas [32] use four categories: (1) simplistic, naive or vague responses, (2) responses that comprise elements of scientific thought, but are erroneous, (3) responses that comprise elements of scientific thought, but are incomplete, and (4) scientifically correct and elaborate responses.

To the best of our knowledge, there are no studies aimed at investigating how students conceptualize the notion of algorithm. On the contrary, most of the computer science education papers devoted to teaching and learning algorithms focus on the learning of specific topics or concepts that are usually covered by algorithms and data structures courses, and the related misconceptions [13, 16, 18], or on the development of skills concerning the design and analysis of algorithms [19, 23].

The notion of *algorithm* has a rich and complex history that spans multiple disciplines and time periods [5, 15]. In fact, the mathematical idea of algorithm and the roots of algorithmic thinking can be traced back to ancient civilizations [24], where general mechanical procedures were already used for instance to solve practical problems such as calculating volumes or determining the position of celestial objects. The term ‘algorithm’ gained new meanings in the early 20th century, throughout the debate over the notion of *effective computability* initiated by Hilbert’s stimulus [21], and concretized by Church [11], Turing [38] and Post [34]. Despite they did not use the term ‘algorithm’, or used it with its mathematical meaning, their contributions in fact gave rise to the modern computer science concept of algorithm, which explicitly refers to the *interpreter* of the procedure and highlights the need for a *notation* to characterize and express the algorithm itself.

The first use of the term under this computer science meaning was probably due to Markov in 1954 [27] (as cited in [15]): «*‘Algorithm’ is commonly understood to be an exact prescription, defining a computational process, leading from various initial data to the desired result.*» In the 1950s and 1960s, the term spread with this meaning in the computing community.

A relevant contribution for our work is by Perrenet et al. [33], who defined four gradual levels of abstraction to describe how computer science students think about the concept of algorithm: 1. execution, 2. program, 3. object, 4. problem. Even though their goal was measuring the abstraction level of students, the materials they used are somehow similar to the material used in our workshop. They built a list of statements about algorithms (*e.g.,* “An algorithm is a program, written in a programming language”, or “The correctness of an algorithm generally can be proven by testing the implementation on cleverly selected test cases”), asked students to discuss whether they agree or not with the statements, and then categorized their answers according to the abstraction level shown. In our workshop, instead of general statements we used a list of concrete procedures to be discussed in a group, in order to trigger a reflection on the notion of algorithm itself.

Given both the problem and the device, an algorithm is the precise characterization of a method of solving the problem, presented in a notation interpretable by the device. In particular, an algorithm is characterized by these properties:

(1) application of the algorithm to a particular input set or problem description results in a finite sequence of actions;
(2) the sequence of actions has a unique initial action;
(3) each action in the sequence has a unique successor;
(4) the sequence terminates with either a solution to the problem, or a statement that the problem is unsolvable for that set of data.

Figure 1: Extract of entry 'algorithm' in the ACM Encyclopedia of Computer Science [25].

3 OUR WORKING DEFINITION OF ALGORITHM

For the current study we rely on the entry in the ACM Encyclopedia of Computer Science for 'algorithm' [25], which in particular includes the extract reported in Fig. 1.

Differently from naive definitions of algorithm like "a step-by-step procedure for solving a problem or accomplishing some end"[1], the above definition encompasses several important aspects and in particular it clearly covers two crucial ones. First, the role of the *interpreter* of the method ("a given device") is explicitly acknowledged. The interpreter is a computational agent able to either execute the algorithm ideally or to execute the program physically [26]. This implies a consistent operational model of the interpreter, which assumes a predefined set of specific actions that the interpreter is able to carry out. The examples in [25] clarify that the "device" may be a human or a machine. Second, the definition acknowledges the need for a *notation* to present the method, which may be the natural language if the device is a human. In fact, algorithms are usually distinguished from programs precisely on this basis: the latter must be written according to the syntax of a formal language, while the former are typically described more freely, by a mixed use of natural language and high-level semi-formal languages (*i.e.,* in pseudocode) [26].

We notice that the role of notation has to do with the endeavor of describing and conveying the algorithmic method more than with the method itself (in a sense, this is the same relationship as the one between an idea and the way one is able to communicate it to another person, for example, in written form).

Moreover, the definition starts by explicitly mentioning a *given problem* (notably, 'problem' instead is not a term deserving an autonomous entry in [25]), and it distinguishes between the *application* of the method (which produce a sequence of *actions*) and its *characterization* (which has to be *precise*). The issue of precision is a fundamental one, and is strongly related to the elicitation of an interpreter. In a sense, it is indeed the explication of an axiomatically precise interpreter, and its capabilities, that makes it possible to describe an algorithm *precisely.* On the other hand, the use of natural language to express algorithms clearly introduces the risk of ambiguity, which would instead require a formal syntax and semantics to be properly addressed. Thus, the distinction between algorithms and programs gets blurred when trying to define them formally [26]. Not surprisingly, then, [25] states that "any program that always halts is an algorithm".

Finally, the definition considers several other aspects including termination (*finite sequence*), input/output (*solution* or *unsolvability statement*), determinism (*unique initial action*, *unique successor*). Besides the definition, the entry in [25] also discusses other desirable properties of algorithms, like correctness, generality, efficiency, which however are not included into the definition as *required.*

As objected by Fant [15], it is worth noting that the above computer science notion of algorithm does not easily apply to many computer science contexts and tasks. For instance: operating systems or online servers do not halt, random algorithms are not deterministic (at least not at the level of abstraction of their description), a logic circuit is not a sequence of operations, operations in concurrent programs are carried out simultaneously. As a matter of fact, many revisions and redefinitions of algorithm have been proposed in the literature—reviewing them is beyond the scope of this paper, the interested reader could start for instance from [5, 29, 40]. Yet, we believe that the concept as defined above is adequate enough from an educational point of view, and that the related issues should be well understood by CS students and teachers.

4 DESCRIPTION OF THE WORKSHOP

The purpose of the three-hour workshop is to give participants an opportunity to think about the concept of 'algorithm' in an operational way. In the first phase of the workshop (around 40 minutes), participants are split into pairs, and each pair is given a sheet with a list of eight procedures described in natural language. We will refer to such procedures using the term *quasi-algorithms*; two of them are presented as examples in Fig. 2 (the full list of quasi-algorithms is available at [4]). Pairs are asked to discuss and decide, for each quasi-algorithm, whether it can actually be called algorithm or not, justifying their decisions (work in pairs encourages reflection and brings out discrepancies and doubts).

In the second phase (around 60 minutes), larger groups are formed by merging pairs; groups are asked to share and compare the decisions of the original pairs, and to search for a new consensus. Then, each group is asked to formulate in writing its own definition of 'algorithm' —based on the previous discussion—, specifying what properties an algorithm must have in order to be defined as such.

Finally, each group reads aloud and presents its definition to the rest of the class, while the workshop facilitator asks clarifying questions and highlights the similarities and differences between the various definitions. The activity concludes with the facilitator summarizing the main aspects of the concept of 'algorithm' and relating them to the definitions given by groups.

All quasi-algorithms are designed and formulated to foster discussion and raise doubts, as they all exhibit potential critical issues that are relevant for the notion of 'algorithm'. In the quasi-algorithm "Lasagna" (see Fig. 2), the last step is vague, as it does not say how long the cooking should take and what the oven temperature should be. Some other steps (*e.g.,* prepare the meat sauce or *béchamel*) are not elementary: you would need further instructions on how to

[1]Merriam-Webster online dictionary, https://www.merriam-webster.com/dictionary/algorithm visited 2023 July 31st.

Lasagna	**Mathematical operations**
(1) Prepare the meat sauce	(1) Read 2 numbers x and y
(2) Prepare the béchamel	(2) Sum x and the triple of y
(3) Boil the pasta	(3) Multiply the result by 3
(4) Layer meat sauce, béchamel, and pasta in a baking dish	(4) Add 73
(5) Bake in the oven	(5) Print 100

Figure 2: Two of the eight quasi-algorithms to be discussed during the workshop.

complete those tasks, unless you assume that the executor already has specific skills[2] (*e.g.,* is a cook with a predefined "library" of skills). Thus, for this quasi-algorithm (and others), it is not easy to give an ultimate answer as whether it can be called an algorithm or not, particularly because it is expressed using the natural language, which is intrinsically imprecise and ambiguous. The interpretation of the instructions may vary depending on the agent who executes them, who is purposely never indicated in the procedure (nor is there any indication of what their capabilities are).

The quasi-algorithm "Mathematical operation" (see Fig. 2) meets all the properties in the definition proposed in §3, and it can sensibly be considered an algorithm that solves the problem of printing '100'. However, the instructions have no internal "teleological" consistency: the results of the intermediate operations are not used in the final part of the procedure, and the intermediate operations are in fact useless w.r.t. the output produced. Furthermore, the procedure always produces the same effect independently of the input numbers, so the procedure's aim is not to provide the result of mathematical operations in spite of what its title suggests. One can also debate the meaningfulness of the solved problem (*i.e.,* printing 100).

Among the remaining six quasi-algorithms, one is built according to the logical programming paradigm, and all the others present features that could be easily perceived as flaws, *e.g.*, one contains unfeasible instructions, another presents logical errors w.r.t. the stated purpose, another is incomplete w.r.t. the possible inputs/initial conditions. Only a few of the quasi-algorithms are general and allow for a range of different inputs; in the other quasi-algorithms, the input is fixed, or irrelevant (see "Mathematical operations" which always prints 100 no matter the input).

5 ANALYSIS OF PARTICIPANTS' WORKS

The workshop has been offered since 2012 in different contexts: (1) as a part of a CSE course targeted to graduate CS students interested in education; occasionally the course was attended also by undergraduate CS students, graduate math students, in-service teachers (from different school levels, from primary to secondary), or perspective teachers from pedagogical schools; (2) as an outreach activity for high school students [3]; (3) as a professional development opportunity for in-service teachers. Some workshops were conducted online due to the pandemic. All participants were informed and agreed that their works would be collected for teaching and research purposes. No further personal information were collected; in particular we did not collect information about gender. Anecdotally, the participants were in majority males, but with a significant portion of females (higher than usual CS courses in our university). Overall, for this study we considered the definitions produced during nine workshops organized since 2017. More precisely, we qualitatively analyzed the definitions of 'algorithm' produced by 27 groups.

As we examined the material already produced over the years by the workshop participants, (and hence we did not design the data collection process beforehand), the analyzed works are not homogeneous. Some groups wrote only their definition of 'algorithm', whereas other explicitly added and defined the fundamental properties of algorithms. Most definitions were textual, however some groups proposed diagrams or conceptual maps. The composition of groups varied as well (teachers only, CSE students only, high school students only, teachers and CSE students mixed) and also the size of groups was not fixed (from four to eight participants, depending on the number of participants in their workshop); the groups' works were collected without keeping track of these details. Nevertheless, we believe that the collected material gives us an interesting overview of teachers' and students' perspective.

We analyzed the definitions of 'algorithm' written by the workshop participants by using qualitative content analysis (as described in [36]). Each definition was split into *segments*, where a segment is either one single word, or an expression formed by 2–3 words. Each segment was then assigned a unique *code*. We followed a deductive

Table 1: Initial set of codes. Top rows are fundamental properties, bottom ones are optional but desirable.

Code	Description
PRECISE	The algorithm must be precise (detailed, complete, clear, self-explanatory, univocal)
METHOD	The algorithm provides a method, a procedure.
PROB.SOLV.	The algorithm solves a problem.
NOTATION	The algorithm is expressed in some notation.
INTERPRETER	There is an interpreter who/which interprets and executes the algorithm.
INPUT	Algorithms are applied to input data.
SEQUENCE	The actions are arranged in an ordered sequence.
ACTIONS	Algorithms involve actions (steps, moves).
HALT	The algorithm must always halt.
DETERMINISM	The next action is uniquely determined by the previous ones, to any given input corresponds the same output.
OUTPUT	The algorithm outputs a solution, an answer.
CORRECT	The algorithm provides the expected output for any input.
GENERAL	The algorithm solves the problem for a class of problems of the same type.
EFFICIENT	The algorithm solves the problem efficiently, *e.g.*, with a careful use of resources.

[2]Notice that our working definition does not require that the actions be "elementary", but it does require that the algorithm be written in a "notation interpretable by the device", which was exactly the point we wanted to cover by this quasi-algorithm.

approach, defining the initial set of codes based on the literature. However, we allowed the flexibility to refine or add codes through the coding process, as emerging from the data (see §6).

In particular, we derived our set of codes from the ACM Encyclopedia entry 'algorithm' [25], see §3. Such codes are presented in Table 1: the first group of codes covers fundamental components or properties of the notion of algorithm; the second group covers additional (desirable) properties of algorithms mentioned in [25]. The coding was conducted in two iterations. In the first iteration eight definitions were analyzed and discussed together by three authors, resulting in a revision of the set of codes. Then, the remaining 19 definitions were coded consensually by two of the three authors. Codes were assigned considering the context of the segment, both to single word segments equal to the code, and to synonyms or terms and expressions that the group of coders considered equivalent (*e.g.,* the same code was assigned both to the single-word segment 'general' and to the term 'class' occurring in the sentence "it must solve a class of problems"). In some cases, the intended meaning was debatable, *e.g.,* "it is finite" may refer either to the description of the"sequence of steps" or to the execution of those steps (implying halting); in those cases the code was assigned as uncertain. Finally, we counted the frequency of codes through the definitions set.

6 FINDINGS

The analysis of definitions resulted in a revision of the initial set of codes. In particular:

- We articulated code PRECISE into three sub-codes:
  - UNAMBIGUOUS: the steps are univocally described, so that they can be interpreted in a unique way.
  - DETAILED: the method and the steps are defined in details, nothing is left unexplained.
  - FINITE: the description of the method is finite (*e.g.,* "a finite set of instructions").
- We added a specification of code CORRECT with a further sub-code:
  - COMPLETE: The algorithm is complete, it covers all cases.
- We added three new codes:
  - INSTRUCTIONS: The algorithm is made of instructions/orders/commands. The focus here is on the fact that the algorithm mandates directions, whereas the code ACTIONS focuses on the actions resulting from the application of the algorithm.
  - BASIC: The instructions are elementary (atomic, simple).
  - EXECUTABLE: The instructions are executable, feasible.

Table 2 shows the outcome of the coding process. The green cells denote the presence (P) of coded segments within a definition, yellow is used to identify uncertain (U) occurrences (*i.e.,* the coders were not sure of the meaning to attribute to the segment). In some definitions we found explicit reference to some dubious property, *i.e.,* the group explicitly wrote that they were not sure or they did not agree upon some property or aspect in their definition; we annotated such doubts (D) in blue. The 'count' column reports the frequency of the green occurrences, the more occurring the darker the background.

All definitions mention an *ordered sequence* of steps or instructions. Most definitions state that algorithms must be unambiguous; those that are not coded as such mention that the instructions must be atomic/elementary. Most definitions state that algorithms *solve a problem.* Only a few definitions mentions *actions* (or steps, or moves); all the remaining ones mention *instructions* (or orders, or commands), often with specification like *atomic* or *executable. Termination* is a required property for about half of the groups and is debated or uncertain in others. Additional properties—like *correctness*, *generality*, and *efficiency*— are seldom included in the definition, and when they occur, they are often debated. Very few definitions explicitly mention an *interpreter*/executer; only two definitions describe the algorithm as a *method* or a procedure; none includes any references to *notation.*

To show what kind of reflections the activity can trigger, we also report some of the issues raised in the groups in the last two editions of the workshop, as annotated by one of the authors at the end of the activities. We do not claim that these issues are representative of all group discussions, but we believe they suggest some interesting insights:

- Must an algorithm always halt or can it go on forever? Is a procedure with a possibly infinite loop an algorithm?
- Must an algorithm solve a problem? And must the problem be meaningful? Is it ok for an algorithm to have useless instructions?
- Must it produce a result? For all possible cases? Always a correct one? Always the same one, given the same set of inputs?
- Instructions must be precise. But what does it mean to be precise? The way out found by a group was that it must be precise enough for the interpreter to precisely know what to do and obtain the expected result. (However, this was not included in their definition.)

7 DISCUSSION

Generally, the definitions contain the basic elements covered in the one given by the Merriam-Webster dictionary (see §3):

- there are steps, instructions, commands, operations (sometimes it is specified that these steps are elementary, simple, basic, repeatable, executable);
- the algorithm solves a problem or performs a task; in each case it must have a purpose, it serves something;
- the instructions are arranged in order.

In addition, groups highlight that algorithms must be precise, unambiguous. We interpret this as a result of the lively discussions initiated by the quasi-algorithms, that were often, purposely, scarcely detailed or vaguely described. However, very few groups came up with the idea of making the interpreter explicit to resolve the issue of the inherent ambiguity of the natural language. Consistently, none was concerned about the need for a notation to express any algorithm. This is somehow surprising for CS students, who are used to code in programming languages or to use other formal notations, and should be familiar with the notion of 'interpreter'. Thus, using the terminology of [32] we would categorize the definitions of workshop participants as "responses that comprise elements of scientific thought, but are incomplete".

Table 2: Coding of algorithm definitions given by groups of teachers and students. Green cells (P) denote the presence of coded segments within a definition, blue (D) means that the group expressed doubts, yellow (U) means that the coders were uncertain about the coding.

				2017			2018					2019				2020				2021				2022				2023		
			#	**1**	**2**	**3**	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**1a**	**2a**	**1b**	**2b**	**1**	**2**	**3**	**4**	**1a**	**2a**	**3a**	**1b**	**1**	**2**	**3**
Definition	PRECISE	UNAMBIGUOUS	**26**	P	P	P	P	P	P	P	P	P		P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
		DETAILED	**8**	U	P		P			U	P				P	P	P	U			P						P		U	
		FINITE	**14**		U	P		P	P	P		U	P						P		P	P	P	P	P	P		P		P
	METHOD		**2**					P		P																				
	PROB.SOLV.		**21**	P	P	P		P	P	P		P	P	P	D	P	P	P	P	P	P	P	D	P	P	P	P	P		
	NOTATION		**0**																											
	INTERPRETER		**3**				P			D															P		P			
	INPUT		**8**			P	P		P		P								P							P	P			P
	SEQUENCE		**27**	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
	ACTIONS		**8**				P	P	P	P		P		P	P	P					U									
	HALT		**14**		U	P	P	P	P	P	P	U	P	P	P		P		U	P	U	U	D	D	P	P				P
	DETERMINISM		**6**	U		P			P		P			D		P							D			P			U	P
	OUTPUT		**10**		P	P			P		P						P					P		U		P		P	P	P
Properties	CORRECT		**3**	P	P																P									
		COMPLETE	**2**	U									D					U									P			P
	GENERAL		**3**							P						P						D	D				P			
	EFFICIENT		**1**	P									D								U									
Emerging codes	INSTRUCTIONS		**20**	P	P	P				P	P		P				P	P	P	P	P	P	P	P	P	P	P	P	P	P
	BASIC		**12**	P	P	P	P	P	P			P	P			P					P		P						P	
	EXECUTABLE		**7**	P		P			P	P			P			P											P			

Another relevant finding is that the participants were often concerned with features of the quasi-algorithms that, as a matter of fact, do not pertain specifically to the algorithm concept. For instance, they lively debated over the fact that the output computed by the quasi-algorithm "Mathematical operations" is not consistent with the purpose declared in its title. We interpret this as if they embedded the problem specification into the given algorithm, instead of seeing the two as distinct. Thus, they seem to consider the algorithm's *correctness* as a property of the algorithm itself instead of a property of the relationship between an independent problem specification and its algorithmic solution [19]. In the same way, they seem to consider the algorithm's *usefulness* as a property of the algorithm itself instead of a property of the relationship between the algorithm's behavior and the user's requirements [22]. A similar argument holds for *efficiency*, especially when this shows up in the form of "unnecessary" operations, inconsistent with a presumed goal. Using the hierarchy proposed by [33], we would place the definitions stated by the workshop participants at level 3 (the algorithm is not connected to any specific programming language), but the discussions often shift towards the "problem level".

Finally, it is interesting to note that most definitions mention instructions and commands, thus exhibiting the (implicit) adoption, by the workshop participants, of the imperative paradigm; this is consistent with the fact that the quasi-algorithm written according to the logical paradigm is seldom accepted by the groups as a true algorithm. In many contexts (*e.g.,* in primary school education) it is indeed appropriate to think of algorithms as finite sequences of instructions. However, and consistently with [10, 17], we argue that CS teachers' and students' conceptualization of algorithms should not leave out other paradigms such as logic or functional programming, and more theoretical models of computation such as the Turing machine, in which algorithms are defined by finite sets of transition rules.

8 CONCLUSIONS

Despite feeling familiar with the concept, the workshop participants often found themselves disagreeing with each other on whether to classify the procedures as algorithms or not; in particular, the issue of ambiguity and the elicitation of the interpreter of the algorithm emerged as crucial ones. Such lively discussions brought about reflections around the idea of algorithm and its interpreter, going beyond stereotyped definitions, and leading participants to deepen their comprehension. These findings suggest that this kind of activities should be offered more often both to students during their study career, and to teachers in professional development opportunities, in order to prompt them to reflect about computing foundations also in a non-technical, more holistic way.

Depending on the different background of participants, the workshop can provide the stimulus to explore the concept of algorithm in several directions. Based on the aspects that turn out to be most debated during the group work, the workshop facilitator can choose which elements to emphasize in the concluding phase of the workshop, when summarizing the main aspects of the concept of 'algorithm' and relating them to the definitions given by groups. It is by no means necessary that all the aspects discussed in this work are touched upon, let alone explored in depth; however, it is important for the workshop facilitator to be aware of them in order to be able to respond appropriately to any doubts, questions, or observations.

The results of this study also provide some insights as to how to improve the quasi-algorithms to propose in the workshop. In particular, we noticed that when a quasi-algorithm exhibits more than one flaw, some aspects can go unnoticed or neglected. For instance the quasi-algorithms could be simplified so that: the quasi-algorithm that contains an infinite loop is precisely described (possibily in pseudocode); the quasi-algorithm that contains useless instruction nevertheless outputs a result consistent with its title; the quasi-algorithm that is inconsistent with its title is precise and logical. It is also worth noticing that our workshop was designed before the current hype on AI tools for programming: these are likely to change the general perception of what it means to instruct a machine and will probably be reflected by any informal notion of algorithms.

REFERENCES

[1] Dan Aharoni. 2000. Cogito, Ergo Sum! Cognitive Processes of Students Dealing with Data Structures. In *Proc. of the 31st SIGCSE Technical Symposium* (Austin, Texas, USA) *(SIGCSE'00)*. ACM, New York, NY, USA, 26–30. https://doi.org/10.1145/330908.331804

[2] Parvaneh Babari, Michael Hielscher, Peter Adriaan Edelsbrunner, Martina Conti, Beat Döbeli Honegger, and Eva Marinus. 2023. A literature review of children's and youth's conceptions of the internet. *International Journal of Child-Computer Interaction* 37 (2023), 18 pages. https://doi.org/10.1016/j.ijcci.2023.100595

[3] Carlo Bellettini, Violetta Lonati, Dario Malchiodi, Mattia Monga, Anna Morpurgo, Mauro Torelli, and Luisa Zecca. 2014. Extracurricular activities for improving the perception of Informatics in Secondary schools. In *Informatics in Schools. Teaching and Learning Perspectives (Lecture Notes in Computer Science, Vol. 8730)*, Yasemin Gülbahar and Erinç Karataş (Eds.). Springer International Publishing, Springer, Cham, 161–172. https://doi.org/10.1007/978-3-319-09958-3_15

[4] Carlo Bellettini, Violetta Lonati, Mattia Monga, and Anna Morpurgo. 2023. *Replication Data for: To be or not to be... an algorithm: the notion according to students and teachers.* Università degli Studi di Milano. https://doi.org/10.13130/RD_UNIMI/JM0R05

[5] Andreas Blass and Yuri Gurevich. 2003. Algorithms: A Quest for Absolute Definitions. *Bulletin of the EATCS* 81 (01 2003), 195–225.

[6] Torsten Brinda and Friederike Braun. 2017. Which Computing-Related Conceptions Do Learners Have About the Design and Operation of Smartphones? Results of an Interview Study. In *Proc. of the 12th Workshop on Primary and Secondary Computing Education* (Nijmegen, Netherlands) *(WiPSCE '17)*. ACM, New York, NY, USA, 73–81. https://doi.org/10.1145/3137065.3137075

[7] Torsten Brinda, Matthias Kramer, and Yannick Beeck. 2018. Middle School Learners' Conceptions of Social Networks: Results of an Interview Study. In *Proc. of the 18th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '18)*. ACM, New York, NY, USA, Article 3, 8 pages. https://doi.org/10.1145/3279720.3279723

[8] Cyril Brom, Tereza Hannemann, Pavel Jezek, Anna Drobná, Kristina Volná, and Katerina Kacerovská. 2023. Principles of Computers and the Internet - Model Lessons for Primary School Children: Experience Report. In *Proc. of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. ACM, New York, NY, USA, 215–221. https://doi.org/10.1145/3587102.3588861

[9] Susan Carey. 2000. Science Education as Conceptual Change. *Journal of Applied Developmental Psychology* 21 (02 2000), 13–19. https://doi.org/10.1016/S0193-3973(99)00046-5

[10] CC2020 Task Force. 2020. *Computing Curricula 2020: Paradigms for Global Computing Education.* ACM, New York, NY, USA.

[11] Alonzo Church. 1936. An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics* 58, 2 (1936), 345–363. http://www.jstor.org/stable/2371045

[12] Ricardo Coelho. 2010. On the Concept of Force: How Understanding its History can Improve Physics Teaching. *Science and Education* 19 (01 2010), 91–113. https://doi.org/10.1007/s11191-008-9183-1

[13] Holger Danielsiek, Wolfgang Paul, and Jan Vahrenhold. 2012. Detecting and Understanding Students' Misconceptions Related to Algorithms and Data Structures. In *Proc. of the 43rd SIGCSE Technical Symposium* (Raleigh, North Carolina, USA) *(SIGCSE '12)*. ACM, New York, NY, USA, 21–26. https://doi.org/10.1145/2157136.2157148

[14] Anna Eckerdal and Michael Thuné. 2005. Novice Java Programmers' Conceptions of "Object" and "Class", and Variation Theory. In *Proc. of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Caparica, Portugal) *(ITiCSE'05)*. ACM, New York, NY, USA, 89–93. https://doi.org/10.1145/1067445.1067473

[15] Karl M. Fant. 1993. A Critical Review of the Notion of Algorithm in Computer Science. In *Proc. of the 1993 ACM Conference on Computer Science* (Indianapolis, Indiana, USA) *(CSC '93)*. ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/170791.170794

[16] Mohammed F. Farghally, Kyu Han Koh, Jeremy V. Ernst, and Clifford A. Shaffer. 2017. Towards a Concept Inventory for Algorithm Analysis Topics. In *Proc. of the 48th SIGCSE Technical Symposium* (Seattle, Washington, USA) *(SIGCSE '17)*. ACM, New York, NY, USA, 207–212. https://doi.org/10.1145/3017680.3017756

[17] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. 2018. *How to Design Programs: An Introduction to Programming and Computing.* The MIT Press, Boston, MA, USA.

[18] Judith Gal-Ezer and Ela Zur. 2004. The efficiency of algorithms—misconceptions. *Computers & Education* 42, 3 (2004), 215–226.

[19] Bruria Haberman, Haim Averbuch, and David Ginat. 2005. Is It Really an Algorithm: The Need for Explicit Discourse. In *Proc. of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Caparica, Portugal) *(ITiCSE '05)*. ACM, New York, NY, USA, 74–78. https://doi.org/10.1145/1067445.1067469

[20] David Harel and Yishai A Feldman. 2004. *Algorithmics: The spirit of computing.* Pearson Education, Harlow, UK.

[21] David Hilbert and Wilhelm Ackermann. 1928. *Grundzüge der Theoretischen Logik.* Julius Springer, Berlin, Germany.

[22] IEEE Computer Society. 2012. *IEEE Standard for System and Software Verification and Validation.* Technical Report IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004). IEEE. https://doi.org/10.1109/IEEESTD.2012.6204026

[23] Philipp Kather and Jan Vahrenhold. 2021. Exploring Algorithm Comprehension: Linking proof and program code.. In *Proc. of the 21st Koli Calling International Conference on Computing Education Research* (Joensuu, Finland) *(Koli Calling '21)*. ACM, New York, NY, USA, Article 28, 10 pages. https://doi.org/10.1145/3488042.3488061

[24] Donald E. Knuth. 1972. Ancient Babylonian Algorithms. *Commun. ACM* 15, 7 (jul 1972), 671–677. https://doi.org/10.1145/361454.361514

[25] Robert R. Korfhage. 2003. *Algorithm.* John Wiley and Sons Ltd., GBR, 36–38.

[26] Violetta Lonati, Andrej Brodnik, Tim Bell, Andrew Paul Csizmadia, Liesbeth De Mol, Henry Hickman, Therese Keane, Claudio Mirolo, and Mattia Monga. 2022. What We Talk About When We Talk About Programs. In *Proc. of the 2022 Working Group Reports on Innovation and Technology in Computer Science Education* (Dublin, Ireland) *(ITiCSE-WGR '22)*. ACM, New York, NY, USA, 117–164. https://doi.org/10.1145/3571785.3574125

[27] Andrei Andreevich Markov. 1954. The theory of algorithms. *Trudy Matematicheskogo Instituta Imeni VA Steklova* 42 (1954), 3–375.

[28] Berta Martini, Marisa Michelini, Alberto Stefanel, and Monica Tombolato. 2021. Prospective Teachers' Representations on the Concept of Force. *Education Sciences* 11, 10 (2021), 614.

[29] Yiannis Moschovakis. 2001. *What Is an Algorithm?* Springer, Berlin, Germany, 919–936. https://doi.org/10.1007/978-3-642-56478-9_46

[30] Andreas Mühling and Gregor Große-Bölting. 2023. Novices' conceptions of machine learning. *Computers and Education: Artificial Intelligence* 4 (2023), 11 pages. https://doi.org/10.1016/j.caeai.2023.100142

[31] Roger Osborne and Peter Freyberg. 1985. *Learning in Science. The Implications of Children's Science.* Heinemann Educational Books, Portsmouth, NH, USA.

[32] Marina Papastergiou. 2005. Students' Mental Models of the Internet and Their Didactical Exploitation in Informatics Education. *Education and Information Technologies* 10 (10 2005), 341–360. https://doi.org/10.1007/s10639-005-3431-7

[33] Jacob Perrenet, Jan Friso Groote, and Eric Kaasenbrood. 2005. Exploring Students' Understanding of the Concept of Algorithm: Levels of Abstraction. In *Proc. of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Caparica, Portugal) *(ITiCSE '05)*. ACM, New York, NY, USA, 64–68. https://doi.org/10.1145/1067445.1067467

[34] Emil L. Post. 1936. Finite Combinatory Processes-Formulation 1. *The Journal of Symbolic Logic* 1, 3 (1936), 103–105. http://www.jstor.org/stable/2269031

[35] Michael T. Rücker and Niels Pinkwart. 2016. Review and Discussion of Children's Conceptions of Computers. *Journal of Science Education and Technology* 25, 2 (2016), 274–283. http://www.jstor.org/stable/43867796

[36] Margrit Schreier. 2014. *Qualitative Content Analysis.* SAGE Publications, London, Chapter 12, 170–183. https://doi.org/10.4135/9781446282243

[37] Anna Sfard. 1991. On the dual nature of mathematical conceptions: Reflections on processes and objects as different sides of the same coin. *Educational Studies in Mathematics* 22, 1 (1991), 1–36. https://doi.org/10.1007/BF00302715

[38] A. M. Turing. 1937. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proc. of the London Mathematical Society* s2-42, 1 (1937), 230–265. https://doi.org/10.1112/plms/s2-42.1.230 arXiv:https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/plms/s2-42.1.230

[39] Jessica Vandenberg and Bradford Mott. 2023. "AI Teaches Itself": Exploring Young Learners' Perspectives on Artificial Intelligence for Instrument Development. In *Proc. of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. ACM, New York, NY, USA, 485–490. https://doi.org/10.1145/3587102.3588778

[40] Moshe Y. Vardi. 2012. What is an Algorithm? *Commun. ACM* 55, 3 (mar 2012), 5. https://doi.org/10.1145/2093548.2093549