Developing algorithmic thinking without programming by designing instructions for encryption

Carina Zindel

Developing algorithmic thinking without programming by designing instructions for encryption

Carina Zindel

University of Cologne, Germany; carina.zindel@uni-koeln.de

Algorithmic thinking is an important part of digital literacy, enabling learners to navigate the digital world with confidence. As part of a Design Research study with 22 learners in grades 5 and 6 so far, it was investigated to what extent the children learn to think algorithmically when they discover an encryption algorithm without use of a programming language. It was found that some learners can intuitively and independently develop certain algorithmic thinking ideas (such as sequencing and loops) when writing instructions (=developing an algorithm). Hurdles show up in the degree of precision and the degree of generality of the instructions.

Keywords: Algorithmic thinking, algorithms, encryption, design research.

Introduction

Digital literacy is important in order to be able to navigate the digital world in a responsible manner. Many IT systems in the world around us can only be used in a responsible manner if the basic ideas behind them are understood. Wing (2006) coined the term "computational thinking" in this context, "that represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use" (Wing, 2006, p. 33). Computational thinking includes processes such as "solving problems", "using abstraction and decomposition when attacking a large complex task", or "thinking recursively" (ibid, p. 33). Hence, computational thinking is a different skill than programming, as "programming" relates to concepts regarding the syntax and semantics of programming languages. As one example, the "CS Unplugged" project (Bell, Rosamond, & Casey, 2012) presents several resources developed to promote computational thinking without a computer.

Up to now, there have been very few learning opportunities on this topic for children in Germany, as computer science is not a compulsory subject throughout the country or even offered at all (cf. Pasternak, Hellmig & Röhner, 2018). This is due, among other things, to the way education is promoted. Computer science is often only offered as an elective subject from grade 9 onwards. Digital literacy is politically regarded as a task for all subjects, and thus especially for mathematics education. Therefore, it is also due to the general conditions that, due to the general educational aspects, algorithmic thinking should be promoted in mathematics classes as well. Besides, the opportunities should be used that exist due to the topics that are covered anyway.

Theoretical background

Although Wing (2006) proposes the term *computational* thinking, this type of thinking should not to be considered as restricted to dealing with computers. Instead, Wing (2006) stresses that

computational thinking is an activity of humans, not of computers. It is a foundational type of thinking that allows humans to attack problems in a way so that they can ultimately be solved using a computer.

Such considerations are not unknown to mathematics. Here, the idea of algorithm proves to be much older than that of computer (Möller & Collignon, 2019). Many algorithms are part of the mathematics curriculum, although their algorithmic nature often remains implicit. That is, certain algorithms are made a topic of discussion (e.g., written arithmetic procedures or solving equations), but it is rarely made explicit that this is an algorithm. This also leads to the fact what constitutes an algorithm is not defined and ideas such as "explicit sequence of steps to solve a problem" or similar have not been an explicit learning object in mathematics education so far.

In order to reduce the complexity of computational thinking as a learning subject, and at the same time to emphasize more the role of the algorithm, this study proposes to use the term *algorithmic thinking* (AT) for capturing the kind of thinking that is necessary when designing and reflecting algorithms (in contrast to technical programming). There are different conceptualizations for specifying what is required for students to possess algorithmic thinking. Following Harlow et al. (2016, p. 340), five algorithmic thinking ideas are distinguished in this paper:

- "**Sequencing**: Creating an ordered list of instructions to complete a task"
- "**Breaking down actions**: breaking an event into smaller parts"
- "**Looping**: Repeating a set of instructions multiple times"
- "**Event-driven programming**: identifying how one circumstance triggers another"
- "**Message passing**: coordinating actions across code or characters"

This paper examines the extent to which AT ideas can be developed. One promising way to promote the development of algorithmic thinking is to engage students in writing instructions for self-discovered algorithms. After all, writing is not only a learning goal in its own right, but also an important thinking tool (learning to write and writing to learn, cf. Morgan, 1998). Austin and Howson (1979, p. 167 f.) also already dealt with the connection between language and concept formation. Heller and Morek (2015) distinguish three different functions of academic language (AL) both oral and written: (1) "AL as a medium of knowledge transmission (communicative function)"; (2) "AL as a tool for thinking (epistemic function)"; (3) "AL as a ticket and visiting card (socio-symbolic function)" (ibid., p. 175). This suggests that writing promotes deeper engagement with the content and thus fosters deeper understanding. At the same time, instructions for algorithms can be conceived as a separate text genre that students need to learn (similar to geometric construction texts, cf. Rezat & Rezat 2017).

In this paper, the idea is to develop algorithmic thinking by inventing an encryption algorithm without directly focusing on programming (similar to the examples of Bell et al., 2012). Encryptions are suitable because, on the one hand, they are an important topic in everyday life. On the other hand, they are suitable because even without programming, the algorithm must be described very precisely and accurately so that someone else can understand or reverse it. The basic idea of a symmetric encryption procedure can be experienced and learned by reinventing the Caesar Cipher, which is described in more detail in the design section below. In reality, of course, this is

currently even more complex. However, since the teaching-learning arrangement should be suitable for grades 5/6, the Caesar Cipher was chosen as an example.

This leads this study to the following research question:

> RQ: To what extent do children in grades 5/6 show AT ideas when writing down instructions (algorithms) for encryption and decryption? What hurdles are encountered?

Methods

The methodological framework of this study is design research (Gravemeijer & Cobb, 2006), whose aim is on the one hand the development of a teaching-learning arrangement in several cycles and on the other hand the analysis of the learning processes initiated by the teaching-learning arrangement, i.e., in particular typical learning pathways and hurdles. In this paper the first results of the first cycle are presented.

So far, 22 learners in grades 5 and 6 have participated (a total of about 1600 minutes of video material). The design experiments were videotaped and passages relevant for the analysis were transcribed.

To select relevant passages, those involving writing the algorithm were identified. To show the range of written products, four cases, some of which are very different, are contrasted below. For data analysis, a qualitative content analysis (Kuckartz, 2019) of these products was conducted and the reconstructed AT ideas were marked by ||...||. For this purpose, a deductive-inductive procedure drawing on the categories from above (||sequencing||, ||breaking down actions||, ||looping||, ||event-driven programming||, ||message-passing||) was chosen to reconstruct which AT ideas learners address when it comes to the elements of instruction (algorithms) and which learning trajectories and hurdles occur.

Design

The learners have been working with the Caesar Cipher (cf. Allen, 2017). In this process, letters are shifted forward in the alphabet by a fixed key (Table 1). For example, the plaintext "hello" and key 3 provides the ciphertext "khoor", as every character is shifted by 3 characters.

Table 1: Clear- and walking alphabet to illustrate the Caesar Cipher (here with key k=3)

plaintext	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
ciphertext	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c

Table 2: Letter number table

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12
n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

When numbers are assigned to the letters of the alphabet as in Table 2, the encodings can also be expressed mathematically by the basic arithmetic operations or modulo calculations. The encoding of a plaintext letter can thus be represented by the calculation $y = (x + t)\ mod\ 26$ where $x, y \in \{0, \ldots ,25\}$ is the number of the plaintext and ciphertext letters, respectively, and $t \in \{1, \ldots ,25\}$ is the key number. As an alternative to the modulo computation, a bifurcation could also be considered: if $x + t < 26$, then set $y = x + t$, otherwise set $y = x + t - 26$. Formulas for decryption using this method are obtained analogously. The plaintext letter is obtained from the ciphertext letter using the calculation $x = (y - t)\ mod\ 26$, which can alternatively be described by the branching "if $y - t > 0$, then set $x = y - s$, otherwise set $x = y - t + 26$".

The teaching-learning arrangement consists of three sessions of 45 minutes each. Here, the focus lies on the learners' instructions they wrote down at the end of the first and the beginning of the second session. In the first session, the learners are first confronted with an encrypted message that they need to decrypt. However, they are not given explicit instructions on the Caesar Cipher, but rather discover the procedure by themselves. Afterwards, the learners are tasked with writing down instructions for their discovered procedure, for encrypting as well as for decrypting messages. The writing assignment demands a lot of precision, or even the ability to commit. This is different from oral language, where it is sometimes rather fleeting and children may contradict each other in the next sentence. By writing it down, a commitment is demanded, and the AT ideas in the learners' instructions are made explicit.

In addition, two design principles were implemented to initiate AT ideas. The design principle "Using your own instructions again" was implemented in such a way that at the beginning of the second sessions the children were asked to encode and decode again according to their own instructions from the first design experiment session and then to reflect on whether they had really only used their instructions. The design principle "confrontation with algorithms (instructions) of other children" was implemented in such a way that they were then confronted with another instruction from another group that addressed other aspects in order to sensitize them to this and to encourage the learners to sharpen their instructions again.

Empirical insights

Four of the instructions written down at the end of the first session are presented below (Table 3). They were chosen because their differences prove instructive for illustrating the range of written products.

Florian and Aaron ||sequence|| their instructions by making several steps explicit ("pick a number," "move each letter in your text by its respective place," "as soon as the alphabet ends, start over"). Moreover, they perform ||breaking down actions|| in the process, as they sequence into several substeps. It should also be emphasized that they are one of the few who also intuitively implement ||looping|| directly ("as soon as the alphabet ends, you start again from the beginning", "then continue with all the letters like this until they make a meaningful sentence"). Finally, they also show the idea of ||event-driven programming|| by formulating conditions ("when you have figured out the number of shifted digits").

Karin and John also show the idea of ||breaking down actions|| by aptly describing the shifting pattern of encryption and decryption ("In encryption you go letters forward", In decryption you always go letters back"). Thus, they also address a step of ||sequencing|| at the same time, but fold the steps less widely than Florian and Aaron. No ||loops|| or ||conditions|| are explicitly formulated. The instructions of Anna and Rebekka are very similar. They too focus on the central shift pattern and thus address ||sequencing|| and ||breaking down actions||, respectively. Unlike Karin and John, however, they do not formulate the instructions in general terms, but for a specific key, namely key 2. Marcel and Jonas, on the other hand, do not address the algorithm per se, but rather focus on the input-output idea by stating that "you have to give the other person the key and then you can write the text". This most closely addresses the AT idea of ||message-passing||, although the statement remains very general.

Table 3: written products for instructions on encoding and decoding of three learner pairs (literally translated from German) and reconstructed AT ideas.

	Encryption	Decryption	AT ideas
Florian & Aaron	Encryption: Pick a number from 1-25 and move each letter in your text by that number. *As soon as the alphabet ends, start again from the beginning.	Decryption: Look first how the letter is called if you shift it by all possible places. When you have found out the number of shifted digits (then continue with all letters) until they form a meaningful sentence.	||sequencing||, ||breaking down actions||, ||looping||, ||event-driven programming||
Karin & John	*In the case of encryption, you move letters forward.*	*When decrypting, you always go back letters.*	||sequencing||, ||breaking down actions||
Anna & Rebekka	*You get the encrypted letter by moving two letters to the right.*	*You get the decoded letter by moving two letters to the left in the alphabet.*	||sequencing||, ||breaking down actions||
Marcel & Jonas	You give the key number to the person you want to write to and then you can write the text.	You should drag the first word through all 25 possibilities and then use the correct principle to decode the other words.	||message-passing||

In summary, algorithmizing works quite well for two written products (||sequencing||, ||breaking down actions||). However, only one instruction also contains the idea of looping and ||event-driven programming||. The third treatment is more on the surface or usage level and, apart from the ||message-passing|| aspect, does not really address the algorithm itself. So, there are some ideas available as resources among the learners. However, not all AT ideas are activated by all learners by themselves. In addition, with ||sequencing|| also - when it occurs - the degree of precision needed is partly unclear.

The lack of precision also becomes clear when looking at the learners' oral statements about it. When asked by the teacher how John and Karin proceeded with an encoding task, John answers "We have always gone one further" (#2) and, when asked again by the teacher (#3), justifies this with "Because when you encrypt you always go forward" (#4).

1	Teacher	Can you briefly describe how you proceeded?
2	John	We have always gone one further.
3	Teacher	Ok, can you still explain why you always went one further?
4	John	Because when you encrypt you always go forward.

With regard to ||sequencing||, John remains as general as they have formulated it in their instructions by only mentioning the step of going one character forward. So, he does not name any further steps verbally either. Afterwards, the teacher gives the order to work on the decoding task “EQTQPC” (plaintext is “CORONA”). John suggests in #10 to try everything from 1 to 25 and starts directly with the first letter of the ciphertext ("E"), which he moves to a "D".

10	John	Ok. Then we would actually have to try out everything from 1 to 25. Then one back from E the D.
11	Teacher	You can write in there again and take notes.
12	John	Ah, yes. D. Ah I didn't click on it. D. Then back from Q, that's P. Then back from T one, that's S. Then back from Q one, that's P again. But that doesn't give a real word (laughs) You can see that now.
13	Teacher	Mhm. What could you do now?
14	John	Then two back, three back and then four back and then 5 back.
15	Teacher	Karin, what would you do now? What is your idea?
16	Karin	I've already tried a few combinations and I'm just about to try A, so. (long drawn out) A. (long drawn out) M. N.
[…]		
22	Teacher	So describe what you're doing right now. What exactly are you really doing now?
23	John	The letter back and try that until it makes a correct word.

In #12, John further unfolds this process using the example. By performing the shift in steps for each individual element, he basically performs a ||loop||. This is also reflected in #14 when, after shifting one place in the alphabet, he now suggests as a further procedure "then two back, three back, and then four back, and then 5 back" (#14). This is a change from the written product, as ||loops|| were not made explicit in the instructions before. This shows that he uses the idea of ||loops|| intuitively as a structured procedure, but that he does not yet describe this process in general. Possibly he lacks appropriate linguistic means for this. A possible linguistic device would be "do … until …". He uses this in #23 to indicate the termination condition for the algorithm as a whole: "until it results in a correct word".

The following is another brief look at what situational potential the design principle of "confronting instructions from other children" can have. Anna and Rebekka formulated their own instructions in an example-based manner. After Anna is confronted with Florian and Aaron's instruction, she notes that the instruction is "much more general" (#101).

101	Anna	This instruction is also much more general. So, I think we understood it last time in such a way that we should really formulate instructions for this example. So, with the right instruction you really come from each task also really to the solution. Both encryption and decryption. And ours is more related to a specific example.

In summary, it can be stated that hurdles can be reconstructed in the *degree of precision* of the instructions or in the *degree of generality*. Under certain circumstances, as seen in the two scenes, these can be resolved by a renewed reflection on one's own instructions by executing them on another example or by a contrasting comparison with instructions of other children.

Discussion

Algorithmic thinking is important for mathematics education and first ideas can be initiated in grades 5/6, as shown here. The initial results of this Design Research study show how diverse the starting points for possible learning processes for algorithmic thinking are. Learners intuitively activate different AT ideas. It should be emphasized that the learners develop AT ideas in a setting that does not draw on any ideas of programming, but rather on the general discovery and description of algorithms. The empirical insights illustrate possible working mechanisms of the design principles investigated in this study, namely “Using your own instructions again” and “confrontation with algorithms (instructions) of other children”, which are suitable for eliciting further AT ideas

The results presented here also make a first contribution to the discussion whether algorithmic thinking or programming should be learned first. It is possible that learners develop AT ideas even without programming, which are thus starting points for further learning processes. It would therefore be wasted potential to simply prescribe the programming language and so without first giving the learners the opportunity to develop the AT ideas themselves. This could provide a promising starting point for the developing of algorithmic thinking, although it is likely that some aspects of algorithmic thinking require programming to become fully developed. The results show that general principles (such as looping) could possibly first be learned in a general way, and then be applied to different programming languages or programming environments such as Scratch. The idea behind this is that the children learn and understand not only the specific handling of a tool, but the principles behind it.

What is challenging is to find an appropriate level of precision and generality when writing instructions for algorithms. Algorithms could indeed be thematized without programming. However, until learners understand how a computer works, they lack a kind of alphabet of what is allowed as ingredients for creating an algorithm. So, it is not trivial to find or name the smallest building blocks that are considered valid for describing an algorithm. Here, learning opportunities are needed. The design principle "confronting with algorithms of other children" has revealed situational potential in this respect. Based on this, the children developed AT ideas on their own.

In the next step, it will be examined to what extent implementing the instructions with Scratch building blocks promotes the degree of precision and to what extent further AT ideas are sharpened or developed. In addition, it is conceivable to check for other (mathematical) topics to what extent AT ideas can be developed in the respective area.

Acknowledgment

I thank my Master student Mr. Nima Khazaei, whose work on his master thesis has supplied the data investigated here.

References

Allen, G. D. (2017). The Caesar Cypher. In G. D. Allen & A. Ross (Eds.), *Pedagogy and Content in Middle and High School Mathematics* (pp. 11–12). Sense Publishers.

Austin, J. L., & Howson, A. G. (1979). Language and mathematical education. *Educational Studies in Mathematics, 10*, 161–197.

Bell, T., Rosamond, F., & Casey, N. (2012). Computer science unplugged and related projects in math and computer science popularization. In H. Bodlaender, R. Downey, F. Fomin, & D. Marx (Eds.), *The multivariate algorithmic revolution and beyond* (pp. 398–456). Springer.

Gravemeijer, K. & Cobb, P. (2006). Design research from the learning design perspective. In K. van den Akker, S. Gravemeijer, N. McKenney & N. Nieveen (Eds.), *Educational Design research: The design, development and evaluation of programs, processes and products* (pp. 45–85). Routledge.

Harlow, D. B., Dwyer, H., Hansen, A. K., Hill, C., Iveland, A., Leak, A. E., & Franklin, D. M. (2016). Computer programming in elementary and middle school: Connections across content. In M. J. Urban & David A. Falvo, *Improving K-12 STEM education outcomes through technological integration* (pp. 337–361). IGI Global.

Heller, V., & Morek, M. (2015). Academic discourse as situated practice: An introduction. *Linguistics and Education, 31*, 174–186.

Kuckartz, U. (2019). Qualitative Text Analysis: A Systematic Approach. In G. Kaiser & N. Presmeg (Eds.), *Compendium for Early Career Researchers in Mathematics Education* (pp. 181–197). ICME-13 Monographs. Springer.

Morgan, C. (1998). *Writing Mathematically: The Discourse of 'Investigation'*. Falmer, Taylor & Francis.

Möller, R. & Collignon, P. (2019). On the historical development of algorithms – hidden in technical devices? In U. T. Jankvist, M. van den Heuvel-Panhuizen & M. Veldhuis (Eds.), *Eleventh Congress of the European Society for Research in Mathematics Education*. Freudenthal Group & ERME. hal-02421871

Rezat, S. & Rezat, S. (2017). Subject-Specific Genres and Genre Awareness in Integrated Mathematics and Language Teaching. *Eurasia Journal of Mathematics, Science and Technology Education, 13*(7b), 4189–4210.

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

Pasternak, A., Hellmig, L., & Röhner, G. (2018). Standards for Higher Secondary Education for Computer Science in Germany. In S. N. Pozdniakov & V. Dagienė (Eds.), *Proceedings of the 11th*

International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (pp. 117–128). Springer.