Informatics in Education, 2026, Vol. 0, No. 0, 1–17

DOI: https://doi.org/10.15388/infedu.2512.009

Algorithmic Approach in Teaching Arithmetic or Going from Concrete to Abstractions in Education of Mathematics and Informatics

Juraj HROMKOVIČ* [0000-0001-9754-7042], Regula LACHER [0000-0002-7571-7298]

Universitätstrasse 6, 8092 Zürich, Department of ComputerScience, ETH Zürich, Switzerland
e-mail: juraj.hromkovic@inf.ethz.ch, regula.lacher@inf.ethz.ch

Abstract. Education is about supporting humans in their growth, with a special focus on exploring their intellectual potential. Learning to act following a given (even complex) pattern is losing its educational value very fast, because all well described activities can be automated. Education therefore should focus on developing those cognitive process dimensions of pupils where technology cannot compete with humans (Dagienė *et al.* (2020), Hromkovič and Lacher (2017), Hromkovič *et al.* (2020)).

The contribution of this paper is conceptual. In the paper we show that starting with the algorithmic view on the historical development of number representations and calculations offers a natural, more understandable way for teaching mathematics in primary schools. We show that going consequently from concrete to abstract empowers pupils to be able to design own representations of numbers, rediscover the execution of arithmetic operations on their own, and even develop elementary calculations in own designed number systems. We show here how a successful process of rediscovery of arithmetic algorithms can be designed by using classical algorithm design methods as "induction" and "divide and conquer". We show how that algorithmic thinking can essentially contribute to improving education in mathematics.

Keywords: teaching to abstract, problem solving, computational thinking, teaching elementary arithmetic operations, number representation, genetic Socrates method, constructivism, algorithmic symbol manipulation, algorithmics, arithmetics.

1. Introduction

> *"The brains of people should not be stuffed with facts, names, and formulae. To know all this, it is not necessary to have passed any school. The real purpose of education is to teach people to think." Albert Einstein.*

Automation has always been the part of human culture that made and makes our civilization more and more efficient (Hromkovič (2015), Hromkovič and Lacher (2017)). Since ever humans acquire knowledge and use it to develop procedures (algorithms) for differ-

*Corresponding author.

ent purposes. The original principle of automation was that many people could successfully execute such procedures without understanding why they work properly, i.e., without having the knowledge of their inventors. The oldest archaeological artifacts documenting algorithms as mathematically described exact procedures are about 4000 years old. In this sense, computer science as the discipline about automated information processing (i.e., about automated knowledge generation) has always been an integral part of human culture (Hromkovič (2015)). Today we are living in the era of information technology that enables us to automate all activities we understand to some extent and execute them faster and more reliable than humans. The 200 years old model of schools striving to educate experts able to correctly act following a complex pattern needs to be updated because the educational value of acting by following given procedure descriptions (does not matter how complex they are) is decreasing fast. Since education is about supporting humans to grow (especially to explore their intellectual potential), nowadays one has to force the development of those dimensions of pupils (creativity, fantasy, critical thinking) in which technology cannot compete with humans (Dagienė *et al.* (2021), Dagienė *et al.* (2020), Hromkovič and Lacher (2023)).

The contribution of this paper is on the conceptual level. We design a novel approach for teaching elementary arithmetics by combining constructivism (Piaget (1926), Piaget and Harel (1950),Wagenschein (1966)), algorithmic thinking (Knuth (1985)) and genetic Socrates method (Hromkovič and Lacher (2025)). This concept essentially deepens the understanding of arithmetic by pupils and simultaneously makes arithmetic algorithms much easier to understand.

We claim that the current style of teaching numbers and elementary arithmetic operations in primary schools does not fit our above formulated requirements. Introducing calculation starts on an abstraction level that is too high. The abstract decimal positional number representation is assumed as given. Still worse, it is ignored that the written algorithms for multiplication and division are products of thousands of years of fine-tuning to minimize the amount of work for their execution as well to minimize the space of their symbolic (abstract) execution. Moreover, all these optimizations have been done without taking care of the understandability of the executed algorithms. And of course, the optimization of the amount of calculation work and of space of calculation execution, and the search for an appropriate number representation impacted each other and were done simultaneously. Teaching these final products as one without being familiar with the process of their development aggravates considerably the understanding of basic arithmetic. Each abstraction arises as a generalization of experiences with concrete, and this experience with concrete is the only well-understood way to introduce abstractions and work with them (Piaget (1926), Piaget and Harel (1950), Wagenschein (1966), Wittmann (1981)).

It is a folklore that many institutions educating teachers decided not to aim to teach the execution of arithmetic operations as division because this typically results in learning the corresponding symbol manipulations without understanding why these algorithms work. But this is a wrong decision because avoiding abstractions restricts the intellectual growth of pupils a lot. Without learning to create and verify abstract models (descriptions) of

objects and processes we essentially restrict the ability of learners to approach more complex topics (a more involved discussion can be found in Delic and Senad (2016) and in the text book series "Mathematik entdecken und entwickeln", (in German) Hromkovič *et al.* (2025a), Hromkovič *et al.* (2025b), Hromkovič *et al.* (2025c), Hromkovič *et al.* (to ap- pear in 2026a), Hromkovič *et al.* (to appear in 2026b)). This is the reason why we start to teach calculation with integers by developing various and more concrete representations of numbers and designing algorithms for the basic arithmetic operations in these less abstract number representations in such a way that pupils are involved in designing number representation as well as in developing descriptions of calculation processes.

In fact, our proposal is based on the daily job of computer scientists. We design symbolic, abstract representations of objects in such a way that we can work with these representations efficiently. From this point of view, the simultaneous development of number representations and algorithmics for execution of arithmetic operations is a representative pattern of the work in algorithmics (Dagienė *et al.* (2020)). This is not allowed to be a surprise, because informatics was since ever an integral part of human culture (Hromkovič and Lacher (2017)).

In this paper we present our concept and illustrate some parts of our design of a teaching sequence devoted to the understandable development of algorithms for multiplication and division. The main didactic strategy is based on splitting the learning path in such small steps (sequences of questions, tasks, and activities see Delic and Senad (2016),Wagenschein (1966),Wittmann (1981), Hromkovič *et al.* (2025a), Hromkovič *et al.* (2025b), Hromkovič *et al.* (2025c), Hromkovič *et al.* (to appear in 2026a), Hromkovič *et al.* (to appear in 2026b)), that pupils are able to make the necessary discoveries for solving particular steps (problems) to a high extent on their own. Because of that we call our approach genetic Socrates method. Note that this was the method how Socrates used to teach. The basic idea is not to present the products of scientific work, but to teach the processes of their discoveries and their development.

To learn how to describe objects (numbers, for instance) and calculation processes in abstract ways is more important than to learn executing concrete calculation algorithms. The ability to abstract and solve problems in their abstract descriptions have the key role in the processes of investigating, understanding, and shaping the world around us, and must be the main issue in education in mathematics and informatics. It corresponds to the proper idea of teaching algorithmic thinking (Knuth (1985), Wing (2006), Denning (2009), Aho (2011), Tedre and Denning (2016), Bollin and Micheuz (2019), Denning and Tedre (2021)).

This paper presents a novel strategy how to teach elementary arithmetic by combining the development of number representations with the design of algorithms for executing elementary arithmetic operations.We illustrate the fruitfulness of their mix of abstraction development, problem solving and algorithmic thinking by showing how easy and systematically the algorithms for addition, multiplication, and division can be designed. The division algorithm in section 4 we already presented in extended abstract (Hromkovič and Lacher (2025)).

This paper is structured as follows. In the second section we shortly discuss the development of number representations with the focus on so called coin-representations. Since our number systems are based on addition of units of number systems, we outline here also how to calculate addition in presented number representations. In section 3 we show how to use the number representations introduced to transparently develop algorithms for multiplication. In section 4, we show how to develop a very well understandable approach for a division algorithm based on two simple modules – exchange of a big coin for several smaller coins of the same value, and equally partitioning a set of equally-valued coins (splitting a pirate treasure). We discuss how to move from this transparent tangible algorithm to an understandable script form (abstract written execution). In conclusion we discuss the importance of our approach of teaching elementary arithmetic.

2. Number Representations and Addition

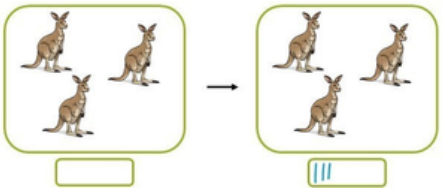
The history of developing number representations is at least 68.000 years old. The first attempts resulted in the unary number representation which is suitable for small numbers only. Nevertheless, historically this was one of the greatest breakthroughs in human history enabling the storage of information outside of the human brain (Hromkovič (2015), Hromkovič and Lacher (2025)). The true digital revolution started more than 5300 years ago with the development of the first scripts in Mesopotamia and then in Egypt as the consequences of the first Big-Data crises in human history. Using alphabets in order to represent shortly (concisely) information as sequences of symbols (today called data as digital representations of information) and store them on media has changed the management of and the living in the old cultures more essentially than the IT does recently. For the first time in the history of mankind, humans were able to store and process information externally outside of their brains on some medias. This allowed humans to store informa- tion more objectively and for an unrestricted amount of time, to communicate information to arbitrary long distances, to build databases for information processing, and to do busi- ness with information.

From the very beginning the number representations of most of the old cultures have been based on addition. One has chosen basic values (units) of the number system used (called coins here), for instance the decimal ones (1, 10, 100, 1000,. . .) in Egypt, the hexadecimal ones (1, 60, 3600, . . .) in Mesopotamia, or the vigesimal system (1, 20, 400, 8000, . . .) by Maya in South America. The origin of the Roman number representation is the same by taking roman coins of values 1, 5, 10, 50, 100, 500, and 1000 and not using the subtraction (not allowing to place smaller values to the left of the larger ones, i.e. IV, IX, CM were not allowed). In what follows we shall call the numbers represented by the addition of roman coins “old roman numbers”. In all these cultures a number was expressed by a collection of coins (basic values) in such a way as the sum of the values of the coins in the collection corresponds to the value of the number. Obviously, based on this principle, one can express a larger number by many different coin collections. To avoid ambiguity in their number representations all old cultures have decided (without any communication between the cultures) that the collection with the smallest number of coins expressing a

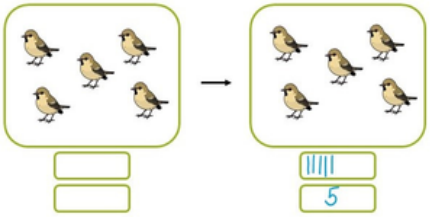
given number is its right representation. For the coins (basic values of the number systems) used by old cultures this representation strategy guarantees the unambiguous representation of all numbers.

From the didactic point of view one of the main advantages of the coin-number representations is that one can work with numbers in a physical representation as collections of coins. This number representation is so transparent that pupils in the second class of the primary school do not have any problem to count and to sum numbers up to 1 million. Addition is especially easy to execute. One takes two numbers in their coin representations as coin collections and unite these two collections into one collection. The sum of the values of the coins in this unified collection offers already the right value of the addition. The pupils are now asked to minimize the number of coins by exchanging smaller coins for equal value of larger coins. Training this exchange of coins without changing the value of a coin collection is the only ability needed to execute addition.

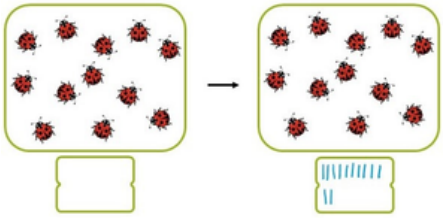
We introduce the number representations and train them by executing addition with them in the order as shown in Figures 1a through 1f and as follows:



(a) Unary representation



(b) Unary representation and the meaning of digits.



(c) Two dimensional unary representation.

(d) Coin representation of numbers.

100	10	1
IIIII	III	IIIIII

(e) Table representation of numbers with lines.

100	10	1
IIIII	III	III III
5	3	6

(f) Table representation of numbers with lines and digits.

Figure 1. Number representations are trained executing additions in the representations. The order of representations is as follows: (a) Unary representation. (b) Unary representation and the meaning of digits. (c) Two dimensional unary representation. (d) Coin representation of numbers. (e) Table representation of numbers with lines. (f) Table representation of numbers with lines and digits.

What are the main advantages of developing number representations in this order?

1. We start with very concrete representations of numbers and move finally to the abstract decimal positional number representation;
2. Moving from one number representation to the next one is transparent and very well understandable;
3. If pupils have troubles with some new, more abstract representation, they can always move back and verify their perception of the representation concept by the already well understood, more concrete representation;
4. As we will show in the rest of the paper, pupils get a much more understandable way to manage the execution of elementary arithmetic operations;
5. Pupils can train to create abstract number representations of other number systems and so learn to abstract (develop abstract descriptions). In the following Figure 2 we see a table (positional) representation of old roman numbers.

M	D	C	L	X	V	I
III	I	IIII		II	I	II
3	1	4	0	2	1	2

Figure 2. Table representation of old roman numbers.

Immediately after introducing a new number representation, it is recommended training to execute addition in this representation. For unary number representation, addition is so easy that it cannot be easier. One joins the representation of both summands, i.e. estimating the value of the results means to keep on counting (Figure 3).

Figure 3. Addition in unary representation.

As a good preparation for working with the coin representation of numbers, it is helpful to train also addition in two-dimensional unary representation (Figures 1c and 4). Addition in the coin representation consists of two actions, the pupils are already familiar with both. The first one is to put both coin representations (collections of coins) of the summands together, and the second one is minimizing the number of coins used (Figure 5). It is only a small step to move from addition in the coin representation of numbers to the table representation. Instead of minimizing the number of coins in the second action of the addition, one executes carryover. The first action corresponds to the unary addition for each coin

value (unit of the number system) see Figure 6a. Here is also advantageous that pupils exercised the addition in the unary two-dimensional representation of numbers.

After mastering addition in all number representations above, pupils understand the operation of addition so well that they can start to execute addition in other number systems (Maya numbers, binary numbers, old roman numbers, etc.) as well as in own number systems proposed by them.

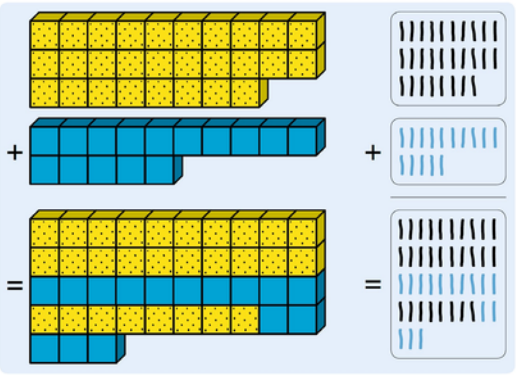


Figure 4. Addition in unary representation with cubes.

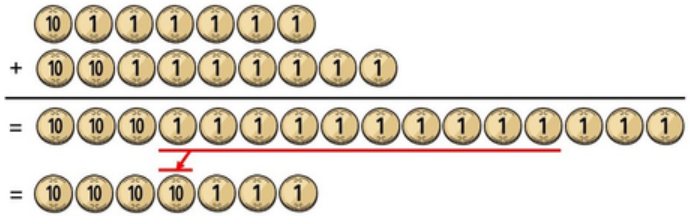


Figure 5. Addition in unary representation consists of two actions: combining and minimizing the number of coins used.

3. Modular Design of Teaching Multiplication

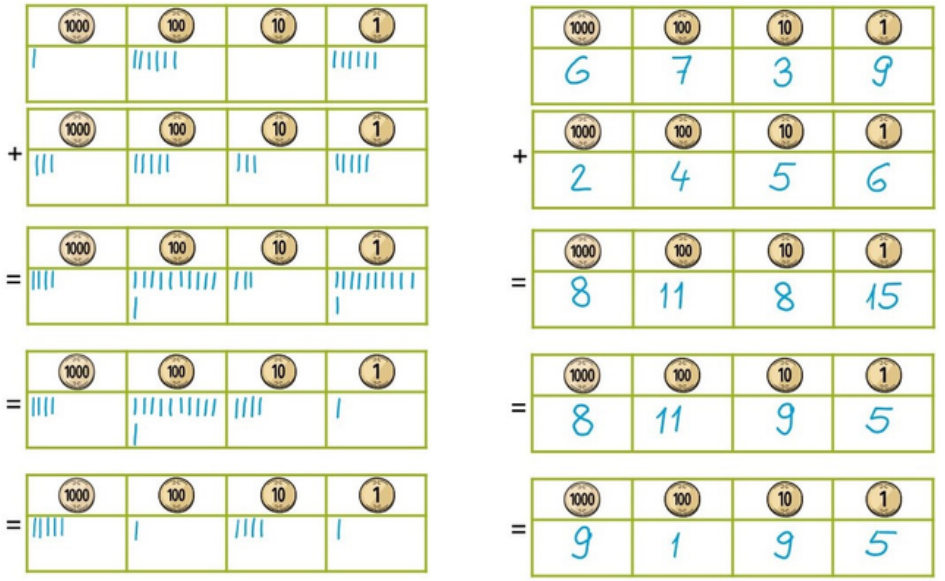
Multiplication is the first complex arithmetic operation pupils have to learn to execute. To develop a multiplication algorithm with the class, it is important, but not sufficient, to move from concrete to abstract number representation as designed in section 2. In addi- tion, one has to work with modular design. This means developing algorithms for simple tasks and using them as building blocks to compose the multiplication algorithm.

For sure, the most fundamental building block is addition, and one has to define multiplication as the sum of several equal values.

$$a \cdot b = \underbrace{b + b + \ldots + b}_{a \text{ times}}$$

This definition offers the first algorithm for multiplication by executing $a-1$ additions (Figure 7).

The next step is again based on algorithmic thinking, focusing on efficient executions of all activities that we want to automate. The motivation not to be satisfied with this simple multiplication algorithm is the big amount of work (number of additions executed) if the multiplier is large. Commutativity can help to increase the efficiency only if one of the numbers in the product is small.



(a) Addition in table representation **with lines** (b) Addition in table representation **with digits**

Figure 6. Addition in table representation of decimal numbers (represented as coins). First unary with lines in the table (a) and with digits (b). Note that we suggest to include to train carryover in both representations.

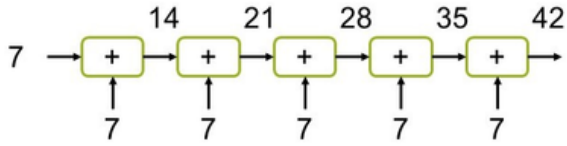


Figure 7. Multiplication 6 7 as the repeated addition of the same value.

Here we recommend to introduce the original Egyptian multiplication that strives to execute multiplication in any number system with well-defined addition by executing as few additions as possible. The goal here is not to present the algorithmic procedure that may be too complex for pupils. The idea is to show pupils examples like the trees (Figure 8) and to give them a sequence of challenges. Each challenge is the execution of a concrete product $a \quad b$ with a minimal number of additions by using the idea of summing the value

b in a tree (Figure 8). Observe that this way you can always execute *a* *b* with less than $2 \log_2 a$ additions.

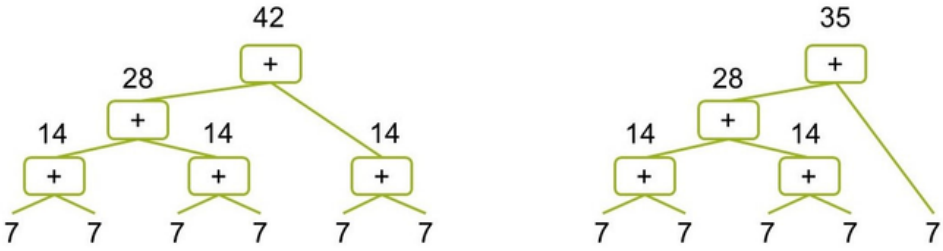
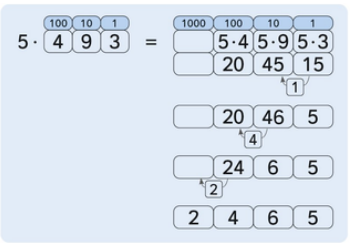


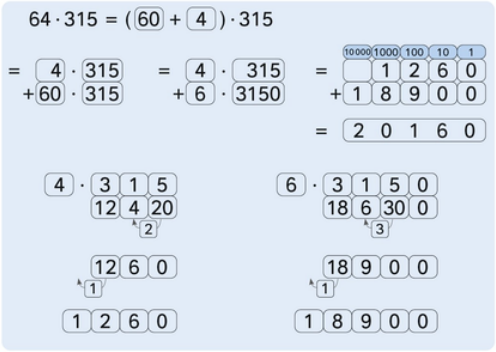
Figure 8. Using a tree to reduce the number of additions to three by calculating 6 7 and 5 7.

After this we recommend to approach our common multiplication algorithm based on the positional number representation. The key point in understanding this multiplication algorithm is the distribute law *(a + b)* *c = ac + bc*, and we recommend to train it by learning the multiplication table by task like 7 8 = (4 + 3) 8 = 4 8 + 3 8 = 32 + 24 = 56. The task formulation can be as follows: "I know 4 8 and 3 8. Can I calculate 7 8 by executing one addition only?

Next, we develop the multiplication algorithm for one-digit multiplier. We start with the coin representation and multiplying with *a* we exchange every coin in the representation of the multiplicand by *a* coins of the same value. After that we minimize the number of coins in the representation as usual. This should be executed only for small *a*'s because we have to avoid handling too many coins. After understanding the principle of multiples of coins, we move to the table representation of multiplicand as in Figure 9a and 9b.



(a) Multiplication with a one-digit multiplier in the table representation.



(b) Multiplication with a two-digit multiplier showing the separation of the decimal units.

Figure 9. Note that we suggest to not minimize the space. In (b) we even suggest to use two supporting calculation tables at the lower part.

When all pupils mastered it, you can shorten the description of the execution of multiplication with one digit multiplier to the common one row written execution in positional decimal representation.

The next module is the multiplication by units 1, 10, 100, ... of the decimal number system. This module and the module of multiplication by one-digit multiplier can be combined to a module for multiplying by multiples of units.

600 3721 = 6 372100 = 2232600. Now, all modules needed are ready to compose the multiplication for arbitrary multipliers.We start with two-digit multipliers as depicted in Figure 9b. Note, that the fundamental idea is again based on the distributive law, because we have to express the multiplier as the sum of tens and ones (64 = 60 + 4 in Figure 9b).

It is important not to strive too quickly executing the multiplication on the smallest possible space as in the common multiplication algorithm. We recommend the larger schema as it enables the pupils to see any time during the execution of multiplication what they are doing and why. It also helps the pupils to train verifying the correct execution of multiplication, and searching for and correcting errors. All this brings more sense of achievement which is the most efficient teaching method. Finally, the pupils become more self-confident and open for experimenting by problem solving.

To extend this schema for multiplying by larger multipliers is quite natural and is no obstacle for pupils in mastering general multiplication. A crucial issue is not approaching too quickly the written common execution of multiplication in the minimal space. The right time for moving in this direction is when pupils find it boring to fill the schema introduced and start to suggest to make it faster by jumping over some parts of the schema.

4. Division by Genetic Socratic Method

Teaching the division, one starts with the unary number representation, which in tangible representation means with a collection of equal-valued coins. The fair distribution of a coin collection is running in rounds and in each round each party gets one coin. The transparent procedure is illustrated in Figure 10. This corresponds to the definition of division *a : b*, where one subtracts b from a as many times as doable.

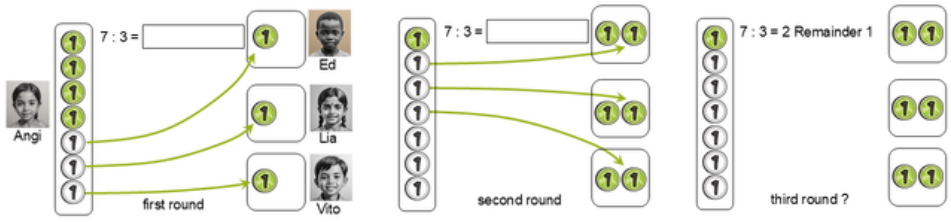


Figure 10. Distributing equal-valued coins to three people.

We see that the fair partitioning of coins in this way can be mastered by pupils in a few minutes and one can easily learn to write the result of the division following the common convention. This simple procedure is the basic module for the general division algorithm. This module has to be applied as many times as the number of positions of the positional number representation of the dividend. In the Figures 11, 12, 13 and 14 we see that one begins with distributing the coins of the largest value on the corresponding number of par- ties given by the divisor. If some rest coins cannot be distributed (because their number is smaller than the divisor), one exchanges them for the coins of the next smaller value. You do not need to explain this action, you have to ask pupils to propose it. At latest after this action pupils are able to design the rest of the calculation on their own. Note that this pro- cedure for division works also if pupils do not start with distributing coins with the largest value. The only disadvantage of not starting with the largest coins is that it can happen, that some coin value will be distributed more than once and finally one has to exchange smaller coins for a larger one in the calculated result.

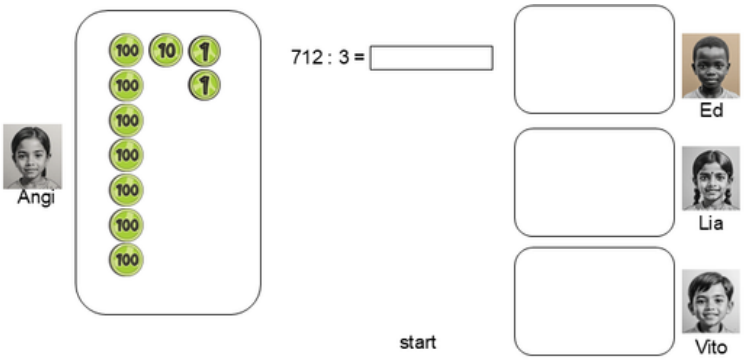


Figure 11. Formulation of a division task with the coin representation of the dividend.

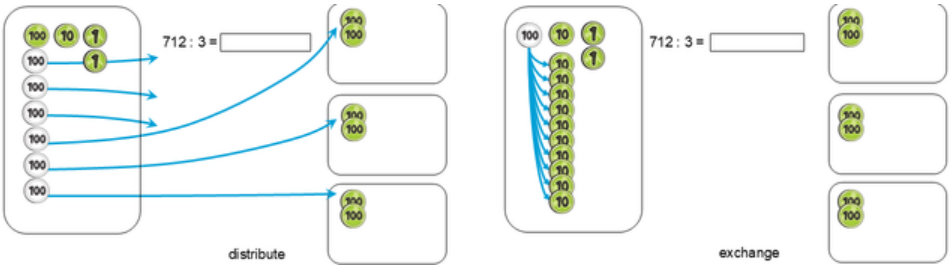


Figure 12. Distributing coins of the largest value 100 and exchanging the rest for coins with smaller value.

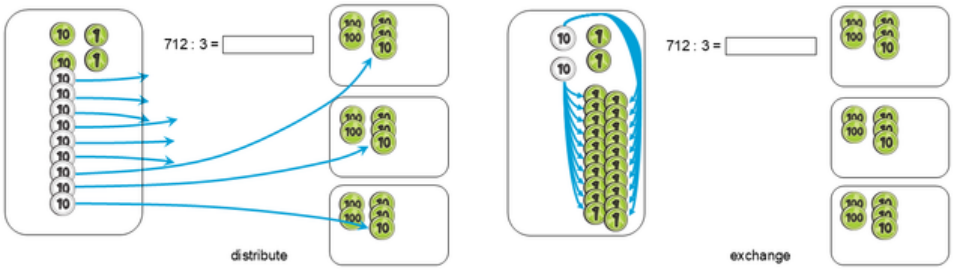


Figure 13. Distributing the coins of value 10 and exchanging the rest for coins of smaller value 1.

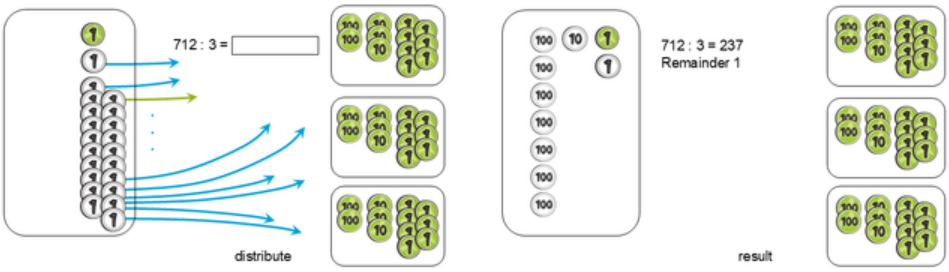


Figure 14. Distributing coins of value 1.

This transparent, concrete way of executing division leads to a better understanding of the operation of division because of the following reason. The repetition of distributing equal-valued coins and exchanging the rest for smaller valued coins can be transparently executed in any number system, not only in the presented decimal system. We encourage teachers to apply this tangible division strategy also for other number systems, because this contributes a lot to the understanding of the process of division.

As we already noted the calculation of the division works also if pupils do not start with the distribution of the largest value. The coins can be distributed in any order. The only risk is that one will distribute coins of the same value more than once. But if pupils propose to start with distributing coins different from the largest one, let them do so and discover the advantage of starting with the most-valued one.

As we have seen above, one can learn to understand division, and divide numbers by small divisors very quickly in a transparent and tangible way. The main difficulty in teaching division is not in understanding its execution in the proposed coin representation; this can be mastered in one lesson. The hardest task is to learn to execute the division in an abstract representation as an algorithm manipulating symbols with which the numbers are represented. We do not recommend to directly in one step go from the presented tangible division execution in coin representations to a written representation of the division procedure although the written representation is only an exact description of the tangible representation introduced. To learn to divide in a written form we developed a sequence

of steps moving carefully from concrete manipulation to its written form. Here we only show the most important step of moving from concrete to abstract in the description of the division procedure. The idea is to avoid the optimization that strives to execute division in the smallest possible place with the smallest number of symbol manipulations. In the representation below we use two columns and show the calculation to be executed on the left side. The left column always involves the coins that still have to be distributed (la- beled “to distribute”), and the right column contains the already distributed coins (labeled “distributed”). The time of the execution is running top down and so the sum of the coins in each row correspond to the value of the dividend. In this transparent description pupils can reconstruct any particular step of division in the concrete and tangible execution (Fig- ures 15, 16, 17, 18, 19 and 20). This enables also to train pupils to search for errors and to correct them, and so to make the kids more selfconfident. First, one starts with divisions using only a small number of coins during the division execution. In this way one can work only with symbols for coins drawing the corresponding coins. Later one can continue as in Figures 15 – 20 where we already use digits in order to write the number of coins used instead of drawing the corresponding number of coins. Note that using cards with digits, one can execute the division in the illustration above in a tangible way and therefore keep the relation between the execution and its description transparent. Some more modules and steps are needed to develop the written division for large divisors, but this is not the issue for the first years of the primary school.

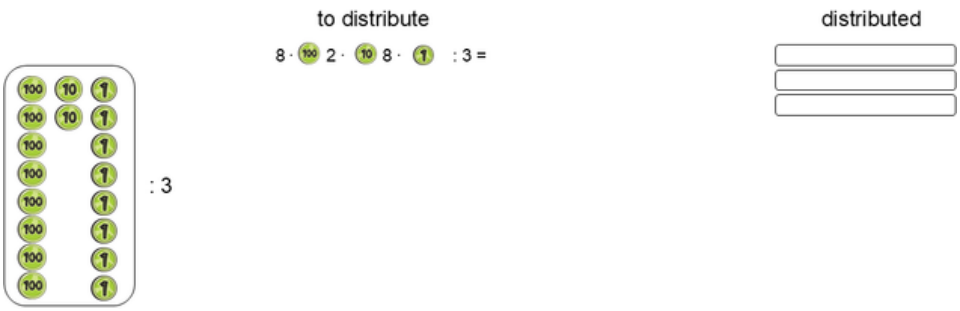


Figure 15. Semi-written representatin in two columns: left is what remains to be distributed and right is what is already distributed. For full transparency, the calculation to be executed is shown in coin representation on the left.

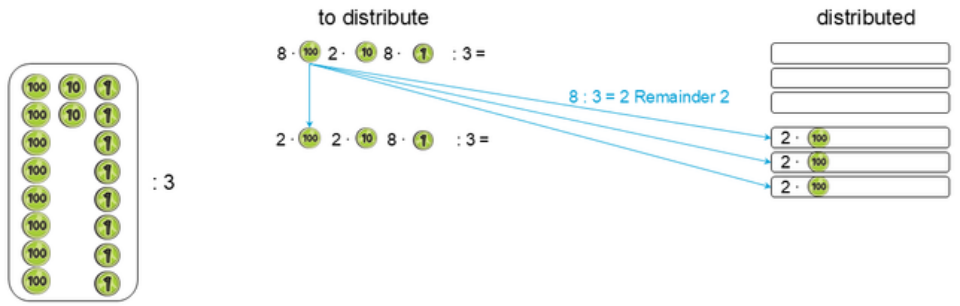


Figure 16. Semi-written representation after the first distribution.

The details of an implementation of teaching division with large divisions can be found in Hromkovič *et al.* (to appear in 2026b).

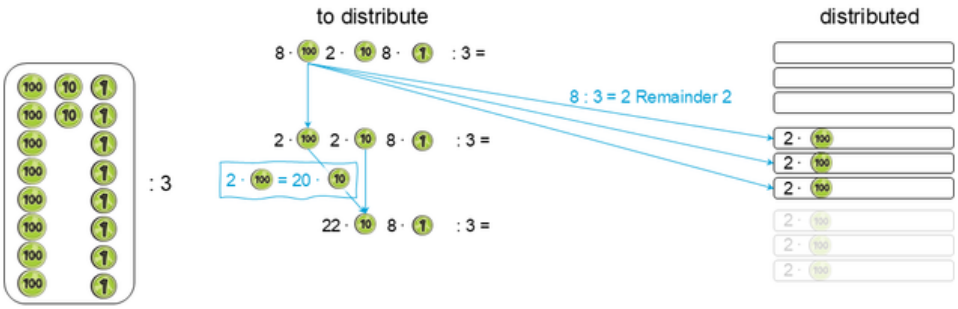


Figure 17. Semi-written representation after the first exchange.

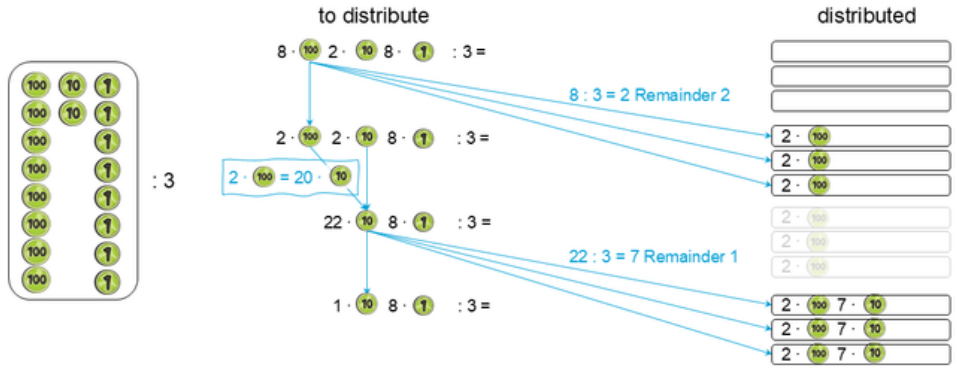


Figure 18. Semi-written representation after the second distribution.

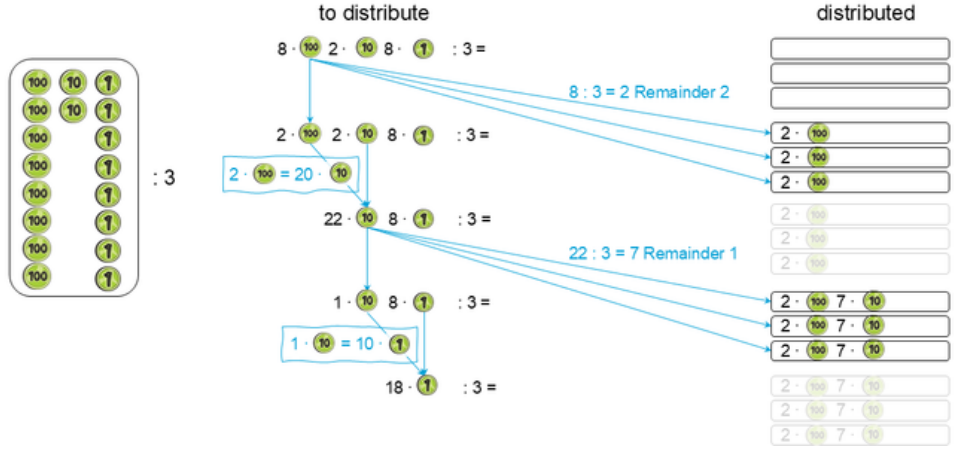


Figure 19. Semi-written representation after the second exchange. Note that in every horizontal line, the total remains correctly represented. This allows the children to check their calculation at all times on their own.

5. Conclusion

For pupils as novices with a very low experience in abstraction there is no other way than starting with concrete and learn to develop the abstraction as a generalization of their experience with concrete by active handling (Piaget (1926), Piaget and Harel (1950)). The hardest part of teaching the execution of arithmetic operations is not in understanding their meaning and in their execution in a transparent and tangible representation, but in their common, abstract, size-optimized written representation. If we omit to work in concrete representations, we risk that pupils will learn to calculate without understanding the meaning of what they do. For preparing the lessons and writing textbooks we recommend us- ing the genetic Socratic method (Wagenschein (1966), Delic and Senad (2016), Wittmann (1981)). The Motto is:

> *"Do not focus to teach the products of science and technology (facts, theorems, models, methods, etc.) and to work with them but teach the processes of their discoveries and their development."*

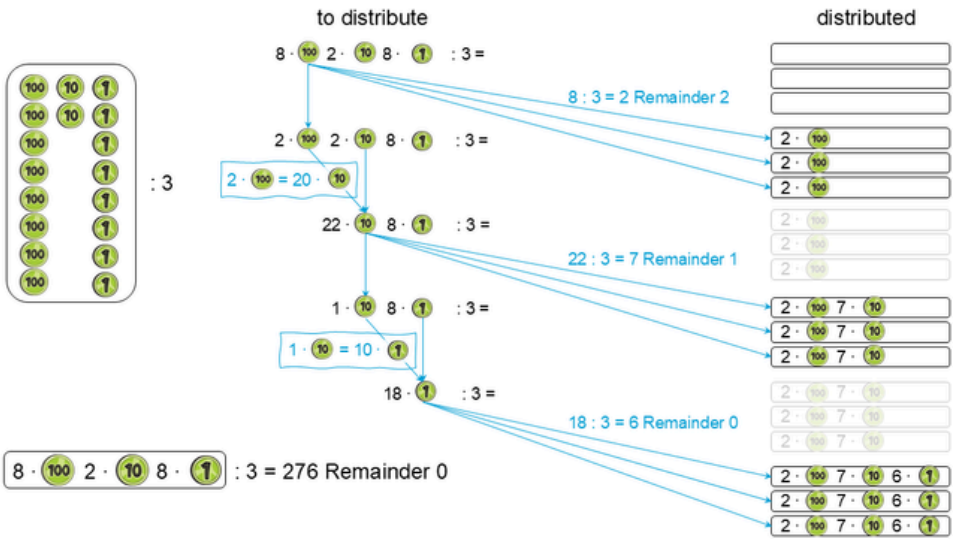


Figure 20. Semi-written representation; last distribution including the result.

Especially this means to study the genesis of the products of science and to also recognize and learn from the failures that helped to achieve breakthroughs. According to this one can divide the processes of inventions in such small steps (sequence of questions, tasks, challenges, and activities (Delic and Senad (2016),Wagenschein (1966),Wittmann (1981), Dagienė *et al.* (2021), Dagienė *et al.* (2020), Hromkovič *et al.* (2020), Hromkovič *et al.* (2025a), Hromkovič *et al.* (2025b), Hromkovič *et al.* (2025c), Hromkovič *et al.* (to appear in 2026a), Hromkovič *et al.* (to appear in 2026b)) that pupils can discover particular steps with high probability on their own. We illustrated this approach by teaching elementary arithmetic operations. The most impressive advantage of using genetic Socrates method is in designing algorithms for division. Observe that after discovering the distribution of

equal-valued coins in the first part of section 4 presented above, the pupils to a high extent are able to participate in the development of the tangible division algorithm which makes the acquired knowledge much more sustainable. The same is true if pupils are involved in the development of the written description of the already mastered tangible division procedure.

To develop teaching concepts presented here authors worked with classes as well with small groups of pupils, but did not execute any empirical study. Some large empirical studies are planned for 2027-2028 with more than 300 project schools.

Acknowledgements

An extended abstract of section 4 was published by Hromkovič and Lacher (2024) under the title "Teaching Tangible Division Algorithms or Going from Concrete to Abstractions in Math Education by the Genetic Socratic Method".

The authors thank Vaidotas Kinčius for the drawing of coins used in several figures and-Westermann Schweiz for re-use of Figure 4 (published in Hromkovič *et al.* (2025b)) and Figure 9 (published in Hromkovič *et al.* (2025c)). Artificial intelligence was used for some of the symbols used: the children in Figure 10 and 11 (OpenArt AI) as well as the animals and pencils in Figure 1 and 3 (Canva).

References

Aho, A.V. (2011). Computation and computational thinking. In: Proceedings of the Ubiquity Symposium (Vol. 1). Association for Computing Machinery, New York, NY, United States. https://dl.acm.org/doi/10.1145/1922681.1922682.

Bollin, A., Micheuz, P. (2019). Computational Thinking on the Way to a Cultural Technique. In: Passey, D., Bottino, R., Lewin, C., Sanchez, E. (Eds.), Empowering Learners for Life in the Digital Age. Springer International Publishing, Cham, ZG, Switzerland, pp. 3–13. 978-3-030-23513-0. https://doi.org/10.1007/978-3-030-23513-0_1.

Dagienė, V., Hromkovič, J., Lacher, R. (2020). A Two-Dimensional Classification Model for the Bebras Tasks on Informatics Based Simultaneously on Subfields and Competencies. In: Kori, K., Laanpere, M. (Eds.), Informatics in Schools. Engaging Learners in Computational Thinking - 13th International Conference, ISSEP 2020, Tallinn, Estonia, November 16-18, 2020, Proceedings. Lecture Notes in Computer Science: Vol. 12518. Springer, pp. 42–54. https://doi.org/10.1007/978-3-030-63212-0_4.

Dagienė, V., Hromkovič, J., Lacher, R. (2021). Designing informatics curriculum for K-12 education: From Concepts to Implementations. Informatics Educ., 20(3), 333–360. https://doi.org/10.15388/infedu.2021.22.

Delic, H., Senad, B. (2016). Socratic method as an approach to teaching. European Researcher, 111, Is. 10, 511–517, https://doi.org/10.13187/er.2016.111.511.

Denning, P.J. (2009). The profession of IT - Beyond computational thinking. Commun. ACM, 52(6), 28–30. https://doi.org/10.1145/1516046.1516054.

Denning, P.J., Tedre, M. (2021). Computational Thinking: A Disciplinary Perspective. Informatics Educ., 20(3), 361–390. https://doi.org/10.15388/INFEDU.2021.21.

Hromkovič, J. (2015). Homo Informaticus. Bull. EATCS, 115. http://eatcs.org/beatcs/index.php/beatcs/article/view/327.

Hromkovič, J., Lacher, R. (2017). The Computer Science Way of Thinking in Human History and Consequences for the Design of Computer Science Curricula. In: Dagienė, V., Hellas, A. (Eds.), Informatics in Schools: Focus on Learning Programming - 10th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2017, Helsinki, Finland, November 13-15, 2017, Proceedings. Lecture Notes in Computer Science: Vol. 10696. Springer, pp. 3–11. https://doi.org/10.1007/978-3-319-71483-7_1.

Hromkovič, J., Lacher, R. (2023). How Teaching Informatics Can Contribute to Improving Education in General. Bull. EATCS, 139. http://eatcs.org/beatcs/index.php/beatcs/article/view/752.

Hromkovic, J., Lacher, R. (2024). Teaching Tangible Division Algorithms or Going from Concrete to Abstractions in Math Education by the Genetic Socratic Method. CMSC 2024, LNCS 15229, pp. 136–144, 2025. https://doi.org/10.1007/978-3-031-73257-7_11.

Hromkovič, J., Lacher, R. (2025). How to Teach Problem Solving and Algorithm Design in High Schools by Constructive Induction or How to Reach True Competences in Informatics Education. Informatics in Education, 24(1), 99–144. https://doi.org/10.15388/infedu.2025.05.

Hromkovič, J., Komm, D., Lacher, R., Staub, J. (2020). Teaching with LOGO Philosophy. In: Tatnall, A. (Ed.), Encyclopedia of Education and Information Technologies. Springer International Publishing, pp. 1–9. https://doi.org/10.1007/978-3-319-60013-0_76-1.

Hromkovič, J., Lacher, R., Marty, J. (2025a). Mathematik entdecken und entwickeln, Praxisbuch 1, Vom Zählen zur Zahldarstellung. Westermann Schweiz AG, Schaffhausen, Switzerland.

Hromkovič, J., Lacher, R., Marty, J. (2025b). Mathematik entdecken und entwickeln, Praxisbuch 2, Von der Zahldarstellung zur Addition. Westermann Schweiz AG, Schaffhausen, Switzerland.

Hromkovič, J., Lacher, R., Marty, J. (2025c). Mathematik entdecken und entwickeln, Praxisbuch 3, Von der Addition zur Multiplikation. Westermann Schweiz AG, Schaffhausen, Switzerland.

Hromkovič, J., Lacher, R., Marty, J. (to appear in 2026a). Mathematik entdecken und entwickeln, Praxisbuch 4, Von der Addition zur Subtraktion. Westermann Schweiz AG, Schaffhausen, Switzerland.

Hromkovič, J., Lacher, R., Marty, J. (to appear in 2026b). Mathematik entdecken und entwickeln, Praxisbuch 5, Von der Subtraktion zur Division. Westermann Schweiz AG, Schaffhausen, Switzerland.

Knuth, D.E. (1985). Algorithmic Thinking and Mathematical Thinking. American Mathematical Monthly, 92, 170–181. https://api.semanticscholar.org/CorpusID:124493215.

Piaget, J. (1926). La représentation du monde chez l'enfant. Translation to German: Bernard, L. Das Weltbild des Kindes, Klett-Cotta, Stuttgart, Germany (1978). F.Alcan, Paris, France.

Piaget, J., Harel, I. (1950). The Psychology of Intelligence. Harward University Press, Cambridge.

Tedre, M., Denning, P.J. (2016). The long quest for computational thinking. In: Sheard, J., Montero, C.S. (Eds.), Proceedings of the 16th Koli Calling International Conference on Computing Education Research, Koli, Finland, November 24-27, 2016. ACM, pp. 120–129. https://doi.org/10.1145/2999541.2999542.

Wagenschein, M. (1966). Zum Problem des Genetischen Lehrens. Zeitschrift für Pädagogik.

Wing, J.M. (2006). Computational thinking. Communications of the ACM, 49(3), 33–35. https://doi.org/10.1145/1118178.1118215.

Wittmann, E. (1981). Grundfragen des Mathematikunterrichts (6th edition). Vieweg, Braunschweig/Wiesbaden, Germany.