X World Conference on Computers in Education
July 2-5, 2013; Toruń, Poland

Extreme Didactic Reduction in Computational Thinking Education

Gerald Futschek, *gerald.futschek@tuwien.ac.at*
Institute of Software Technology, Vienna University of Technology, Austria

Abstract
To master complex education scenarios, as for example learning algorithms, data structures, programming or data modelling, the educational technique of "didactic reduction" is very useful. For getting better educational results at the end the teacher may reduce details of the learning amount at the beginning. The goal of learning computational thinking is an understanding of principles and concepts of computer science. Depending on the learner and the learning situation the amount of details of the learning content or the size of learning objects or the amount of abstraction may be reduced. The purpose of reduction is to ease or even allow the understanding of concepts. We call a didactic reduction an extreme didactic reduction if the reduction is in some aspects so extreme that it cannot be reduced any more. An extreme didactic reduction may omit completely usually essential content or learning aspects. We give some examples of learning scenarios with different forms of extreme didactic reduction.

Keywords
didactic reduction, learning algorithms and data structures, learning programming, learning without text, learning by contest

INTRODUCTION
Teaching computational thinking (Wing, 2006) is a complex task. Especially for beginners, a wide variety of different aspects should be learned. Wing (2008) poses the "challenge: What are effective ways of learning (teaching) computational thinking by (to) children?" It is a real challenge because of the number and complexity of the concepts needed to teach/learn computational thinking. To avoid a mental overload of too many details, a strategy is needed to learn computational thinking in a way that allows easier learning without loss of essential learning content.

"Didactic reduction" is a term first used by (Grüner, 1967) to denote a specific teaching strategy where aspects of the learning content are simplified. Especially in teaching beginners, the amount of concepts can be reduced and the concepts itself can be simplified at the beginning. Later on by and by more details are added and so the simplifications are removed step by step. The simplification must be made in a way that the crucial aspects of the concepts are still valid and that the reduced aspects can be added and learned later. The Term "Didactic Reduction" is widely used in German pedagogical literature but seldom used in English literature. The synonymous terms "Simplification" or "Elementarization" express only a part of the full meaning of "Didactic Reduction". Didactic Reduction expresses also finding a way to teach a complex concept. Furthermore Didactic Reduction is widely used in natural sciences. This paper is also an attempt to use this term in the context of computer science education.

LEARNING PROGRAMMING AND ALGORITHMS

Understanding computer programming and learning algorithms and data structures needs a wide variety of different skills. Novices in programming are faced to learn at the same time:

- a new artificial language (the programming language)
- a new virtual programming environment

They have to apply a new thinking style to

- understand the problems to be solved
- create problem solving strategies
- use testing strategies
- apply a software development process

These are too many different aspects to learn all of them at the same time and in full extent. To avoid “mental overload” the teaching strategy of didactic reduction seems to be appropriate.

EXTREME DIDACTIC REDUCTION

There are different degrees of didactic reduction. The more details of learning contents are reduced, the easier for beginners to comprehend. But if too many essential details are removed the students may not understand the learning content any more. We define the maximal possible didactic reduction as extreme didactic reduction. In an extreme didactic reduction seemingly essential aspects of the content may be reduced too. Lehner (2012) uses in his book about didactic reduction the term “Extreme Reduction” and gives as examples a multiple choice task and a joke. He compares the extreme reduction with a coarse sieve, where only the largest pieces remain. In the following sections we give some examples of extreme didactic reduction in the field of computational thinking education.

LEARNING PRINCIPLES OF ALGORITHMS

Solving a problem with help of an algorithm needs a variety of competences:

- Know the problem domain to be solved
- Know and apply or even invent algorithmic problem solving strategies
- Know a set of basic algorithms and their properties
- Know and use of the basic actions that can be used by an algorithm
- Know and apply a language to describe algorithms
- Know the execution model (used memory, parallel/sequential execution, deterministic/nondeterministic execution)

To find an approach to start learning principles of algorithms we reduce aspects that are not necessary at the very beginning. The second competence “Know and apply or even invent algorithmic problem solving strategies” seems to be the most important one in respect to learn principles of algorithms. A strategy that concentrates on this competence seems to be the most promising because this competence is the only one that involves a creative aspect. The didactic reduction should be made in all the other competences of the given list. In (Futschek & Moschitz, 2010) a technique is described for “inventing and playing algorithms” that is very suitable for all beginners of all ages. A group of students should solve a given problem by finding (inventing) an algorithm that solves the problem. By playing their algorithms they can execute and test their algorithms.

An Example of this kind of learning activity

The task is to find an algorithm to distribute copies of a paper to the audience. Each person in an auditorium (lecture hall) should get a copy of a paper. There are

enough copies available in form of a stack. This procedure should be done very quickly. The usual solution: The teacher gives a stack of copies to the first student in a row and each student that gets such a stack takes a copy from top of stack and gives the next student the remaining stack. This seems to be an easy to comprehend algorithm that can easily be executed by the students and needs no algorithmic notation at all. The only disadvantage of this algorithm may be the long execution time if the number of students is high (it needs n steps for n students). The new task for the students is to find an algorithm that is more efficient so it fulfils the task in less time. The students may find out that they can work in parallel. They may also find out that they can split a stack of copies. The students sometimes find very efficient solutions where each student who gets a stack takes a copy, splits the remaining stack in two or more parts and gives the parts to students who had not got a copy so far. The order of this algorithm can reach log n, if finding a student that has not got a copy so far is an atomic action of order 1. When sitting in rows the algorithms found by students are usually not so efficient, because the students can pass stacks just to the students sitting next to them.

It is much better when students find themselves algorithmic solutions to given problems than just trying to understand algorithms that were invented elsewhere. It is also important that the students play the algorithms that were invented by them. So they see how far their algorithms are working and what problems are still to be solved. By inventing and playing algorithms beginners may learn essential concepts of algorithms in a very easy way with a high level of personal involvement that helps to create motivation to think about solutions of given problems. For example they learn that all steps of an algorithm must be exactly defined, so that each player knows exactly what to do. If they involve parallel algorithms, what is very natural with human players, they also learn concepts of parallel computation. For example they learn that each player can listen at most to one other player at the same time and that interaction with another player needs prior synchronization (for example by looking in each other's eyes).

A notation for an algorithmic language, a definition of basic actions of this language, an execution model and an extra study of the problem domain is not necessary in such a learning scenario. The algorithmic language is the natural language of the players, the basic actions are all actions that can be performed by humans, the execution model are the students that interact according the given algorithm, and the problem domain is the student's daily life and needs no special clarification. So the didactic reduction is in this case extreme because most of the usually needed competences are reduced to very easy understandable aspects. It is the choice of the problem and the algorithmic environment of the playing students that allows the reduction to the essential competence.

THE TASKS OF THE BEAVER CONTEST

As a second example of extreme didactic reduction in computational thinking we want to mention the tasks posed by the international Beaver Contest on informatics and computer fluency (Dagiene & Futschek, 2008). This contest intends to promote computer science concepts to secondary school pupils. Within 40 to 50 minutes the pupils have to solve 18 to 21 tasks online on a computer, usually at school under supervision of a teacher. The challenge for the international task committee is to create tasks that

- convey concepts of computer science
- can be solved within 2-4 minutes
- need no specific pre-knowledge of the pupils

Because of the different educational systems in the participating countries, there cannot be assumed a common curriculum or common knowledge of the internationally participating pupils. No specific computer science term can be used in all beaver tasks. Didactic reduction is necessary. The situation is quite extreme: computer science concepts should be conveyed in tasks of about 3 minutes solving time to pupils that may have no specific computer science knowledge at all. The solutions are quite amazing tasks that usually contain a short cover story of a given situation, a question and a possibility to give an answer (multiple choice or interactive).

The Beaver Contest is not used to test knowledge or skills of already learned content. This contest is aimed to pose interesting computer science related problems and so activate students to be more interested in computer science. By solving these problems the pupils may learn problem solving strategies that are usual in computer science. The focus of beaver tasks is on computational thinking that is necessary to solve the problems. The typical thinking style of computer science is important. Specific systems, notations or technical terms are not so important and omitted in this approach. There is also not enough time to do this during the contest. After the contest the teacher may explain some of the computer science concepts in detail.

LEARNING OF COMPUTATIONAL THINKING WITHOUT TEXT AND LANGUAGE?

A nice and challenging idea is to develop learning objects that do not contain text and written language. The advantage could be that all people of the world from all cultures independent from their language and education may use the same objects. Especially illiterate people may learn despite of their impairment. Kalinyaprak, Futschek, Blaha et al. (2005) show some examples of e-learning sequences that convey assembling and operating instructions that are completely independent from text and language. The way of learning is to mimic the actions that a person in a movie does. This kind of learning is especially adequate in learning practical skills that are independent from text and language, e.g. how to operate a machine, how to assemble objects, how to behave in special situations.

The question in our context here is: Can this kind of learning, that does not involve text and language, also be used to learn computational thinking? It seems to be an unsolvable task because computer science is heavily text based if we look at all the thick computer science text books.

An extreme example to learn computer science concepts without text and language may be a (not yet existent) Beaver Contest for the youngest children of age 5 to 7. At that age most of the children are not able to read and understand written problem statements. But they are able to do interactions with computers and they are able to perform simple computational thinking activities. Beaver tasks for the youngest children may be posed in the form of non-verbal instructions where interactivity is asked for solving a puzzle that needs some computational thinking skills to solve.

Proposal of a Beaver task for 5 to 7 years old children: A version of this task was originally posed in Beaver Contest 2011 for grade 5-6 (age 10-11). We removed here all text of the original task and added just the question mark to indicate the position where we want to fill the missing colours. We hope the task is still understandable and an example for a learning sequence without use of text and language.

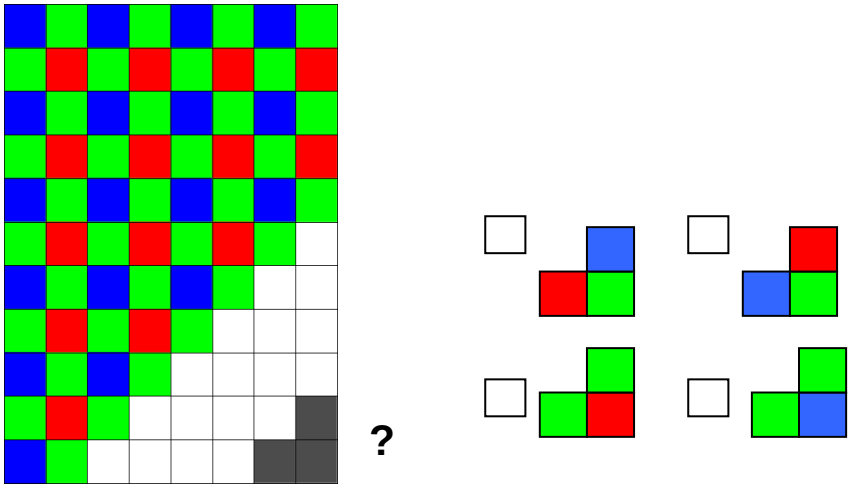

Figure 1: A proposal of a beaver task without use of text and language

The use of symbols like the question mark (?) in our task proposal are normally understandable by pre-school children, see for example the learning scenarios of Mioduser, Kuperman & Levy (2012), where even letters and digits are widely used in kindergarten education. The omission of all text may contribute to confusion, when pupils do not know what is expected from them. In our example it may be unclear that only the 3 tiles in the lower right corner may be filled with one of the offered tiles. To improve this situation the computer may read loud the problem statement in the language of the pupil. Then language is used to help understanding the problem statement but the youngsters still must not read a given text. The next challenge will be to construct a reasonable large set of interesting beaver tasks for that age group that involve also some aspects of computer science concepts that can be interesting for such young learners.

CONCLUSION

Extreme didactic reduction differs extremely from case to case. It highly depends on topic, target group and learning objective. The examples mentioned above show this variety. Extreme didactic reduction focus on specific aspects of the learning topic, it may give beginners quick access to essential aspects of the topic while other aspects are not necessary to be mentioned, at least at the beginning. In relation to computational thinking we found a series of possibilities to apply extreme didactic reduction. In our examples we showed that also aspects that are often seen as essential may be dropped in order to draw the attention of the learners to the most important aspect from the very beginning. By trying to avoid text and language in learning materials we trespass the limits of didactic reduction for most cases in computational thinking learning. But in the special case of a fictive Beaver Contest for pre-school and primary school children we need examples of learning without text and language.

REFERENCES

Dagienė, V., & Futschek, G. (2008). Bebras International Contest on Informatics and Computer Literacy: Criteria for Good Tasks. In *Informatics Education-Supporting Computational Thinking: Third International Conference on Informatics in Secondary Schools-Evolution and Perspectives ISSEP,* LNCS 5090, Springer Verlag, Berlin, Heidelberg, pp. 19-30.

Futschek, G. & Moschitz, J. (2010). Developing Algorithmic Thinking by Inventing and Playing Algorithms, *Constructionism 2010*, Paris.

Grüner, G. (1967). Die Didaktische Reduktion als Kernstück der Didaktik, *Die Deutsche Schule*, Münster: Waxmann Verlag.
Kalinyaprak, H., Futschek, G., Blaha, G. & Weippl, E. (2005). E-Learning without Text and Language: A Language-Free Learning Model, *Proceedings of EDMEDIA*.
Lehner, M. (2012). *Didaktische Reduktion*, UTB 3715, Bern Stuttgart Wien: Haupt Verlag.
Mioduser, D., Kuperman, A. & Levy, S.T. (2012). Design and Learning in the Kindergarten, *Proceedings Constructionism 2012*, Athens, pp. 620-624.
Wing, J. M. (2006). Computational Thinking, *Comm. of the ACM 49(3)*, pp. 33-35.
Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 366(1881)*, pp. 3717-3725.

Biography

Gerald Futschek is associate professor at the Faculty of Informatics at Vienna University of Technology. His main research areas are software engineering and didactics of informatics with special interest in teaching/learning programming and algorithms. He is highly engaged in the international educational initiatives ECDL and Bebras Contest. He is past president of the Austrian Computer Society OCG and Austria's representative of IFIP TC 3.