A Mathematical-Algorithmic Approach to Sets: A Case Study

Judith Gal-Ezer[1] Orna Lichtenstein[2]

April 1996

Abstract

The aim of this paper is to show, by means of a mathematical example, how algorithmic thinking and mathematical thinking complement each other, and how an algorithmic approach leads to questions that deepen the understanding of mathematical material by students. Following influential scientists like Knuth, Ralston and Maurer, we think that the cultivation of algorithmic thinking - a main and natural component of the computer science curriculum, is also a central part of mathematical education. We believe that interweaving mathematical thinking and algorithmic thinking, will constitute a crucial contribution to the students' education.

Introduction

It is often difficult to draw precise boundaries between disciplines, and it is even harder when it comes to mathematics and computer science: Is computer science a branch of mathematics? Is algorithmics, the central part of computer science merely a branch of mathematics? Is mathematics perhaps a branch of computer science? We will not go into these issues, which have been thoroughly discussed by Knuth in [K1], for example. We will only mention that valid arguments can be made for either propositions. However, and despite what has been just said, artificial boundary lines are drawn between these two disciplines in school and university mathematics or computer science study programs. We want to try to remove these artificial boundary lines, and demonstrate how algorithmic thinking and mathematical thinking can be integrated either in the mathematics curriculum or in the computer science curriculum.

Algorithmic thinking and mathematical thinking have been discussed by mathematicians and computer scientists such as Knuth, Maurer and Ralston [K2,M,MR]. Knuth for example, sums up the common features shared by algorithmic thinking and mathematical thinking in a table. These include formula manipulation, representation of reality, reduction to simpler problems, abstract reasoning, information structures and detailed descriptions of algorithms. He also points out that while mathematical thinking deals with infinity - and algorithmic thinking does not - it ignores completely issues such as computational "cost", which is always one of the concerns of algorithmic thinking.

Algorithmic thinking encourages learners to construct a solution, prove its correctness, and analyze its complexity. This contributes to the understanding of problem solving, and therefore has pedagogical value, as emphasized by Knuth in [K1]: "... a person does not really understand

[1] The Open University of Israel, e-mail: galezer@cs.openu.ac.il
[2] Center for Technological Education, Israel, e-mail: orna_l@barley.cteh.ac.il

something until he teaches it to someone else. Actually a person does not really understand something until he can teach it to a computer, i.e. express it as an algorithm." This is something most of us have experienced, and it definitely strengthens the feeling that the cultivation of algorithmic thinking must be a central component of mathematical education. More about the different modes of thinking can also be found in [GZ].

In this paper we try to show that algorithmic thinking and mathematical thinking complement each other, and that an integrated mathematical-algorithmic approach can be taken when teaching and learning a subject. We chose an example - a case study - that fits into the mathematics curriculum as well as into the computer science curriculum. We introduce an abstract definition of a set, discuss the structure of the set, and present an algorithm for its construction, leading to questions of the kind: Is the algorithm correct? Can its correctness be proven mathematically? What is the mathematical generalization of the problem? We touch upon concepts such as algorithmic problems, decision problems, and other important and profound concepts like data structures, recursion, complexity of algorithms, and more.

However, we must make one point here: This paper is not supposed to be a study unit. This means that we present briefly fundamental mathematical and computer science concepts. We raise questions, and we lead to other questions, but, we don't necessarily give all the details of the solutions, nor the exact flow of the study. But, we try to provide some guidelines for the professionals who will write the study unit, and for the teachers who will teach this subject. In this paper we attempt to be clear and fluent at the expense of filling in all the details.

The set S – an abstract definition

We start with an abstract definition of a set S.
A set S is defined by the following three axioms:
A1. $0 \in S$,
A2. $x \in S \rightarrow 2x + 1 \in S$ and $3x + 1 \in S$,
A3. S is the minimal set satisfying A1 and A2.

This definition is taken from Engel's book: "Exploring Mathematics with Your Computer" ([E]). Engel introduces this mathematical example, and then uses the computer as a machine to help solve a specific membership problem: "Which of the numbers 511, 994, 995, 996, 997, 998, 999 belong to S?" Two different computer programs which provide a solution to the problem follow the presentation of the problem.

Engel incorporates the computer into a mathematics course, a set theory course. This, we believe, is a meaningful step in the right direction. However, we would like to take this approach a little further. We want to incorporate algorithmic thinking into mathematical thinking. Mathematical thinking and mathematical reasoning, play an essential part in all areas of computer science. So that mathematical thinking is almost always incorporated into the computer science curriculum. What we want to exhibit here is how to take an algorithmic approach while discussing a mathematical subject, or while trying to solve a problem from the mathematics curriculum. Let us proceed.

Considering the abstract definition of S, it seems quite natural to consider first, well known sets like N - the natural numbers including 0, or Z - the integers, and check whether they satisfy axioms A1 - A3. It can very easily be shown that N or Z satisfy the first two axioms but not the third one. We can take out the number 2 for example, and still the axioms A1 and A2 are satisfied by the new set $N - \{ 2 \}$. This has to be proven mathematically, of course. Teachers might prefer to choose a more formal way of teaching, in which case they can rephrase what has

just been said as an exercise in this way: Prove that the sets $\boldsymbol{N}$, $\boldsymbol{Z}$, and $\boldsymbol{N} - \{\ \mathbf{2}\ \}$ satisfy the axioms A1 and A2.

The students should be given time to "play around" a little, and get the feeling that there are many sets that satisfy axioms A1 and A2. It can be proven that actually there are an infinite number of such sets. At this point the teacher will explain the role of axiom A3. It should be made clear that axiom A3 is necessary in order to specify one and only one set from the infinite number of sets defined by the two axioms A1 and A2. This one set will be referred to hereafter as $\boldsymbol{S}$.

The teacher will elaborate a little on the concept of "the minimal set" - a somewhat abstract concept, which does not really help in "feeling" what the actual set looks like. As we will show in the next section, here is where the algorithmic approach comes in.

From an abstract definition to a concrete set

The students will now use the three axioms trying to construct $\boldsymbol{S}$. For the objective of this paper it is important to point out that by thinking of the construction of the set $\boldsymbol{S}$ we have already invoked the algorithmic mode of thought. We were not satisfied with only an understanding of the abstract set definition, we wanted to actually construct the set.

It looks simple enough to use the first two axioms as an algorithm for producing a list of numbers, satisfying them. The first axiom provides the first element **0**. We mark the **0**, and add the element **1**, by means of the second axiom. We then move to the next unmarked element **-1** , which by means of A2, yields **3** and **4**. The next unmarked element is **3**, and it yields **7** and **10**. When proceeding we finally get the list **L**:
0, 1, 3, 4, 7, 10, 9, 13, 15, 22, 21, 31, 19, 28, 27… Here is the algorithm in pseudo-code:

Algorithm Algor

```
write 0;
while you have time or patience do the following:

find the first unmarked element x;

mark the element x;

add 2x + 1 and 3x + 1 to the end of the list;
```

Clearly, this algorithm generates a "theoretically" infinite list of non-negative integers. It is worth noticing that this procedure does not generate all the natural numbers, i.e. the set $\boldsymbol{N}$; it generates a subset of $\boldsymbol{N}$. This again must be proven mathematically, and the teacher might prefer to rephrase this as an exercise, and ask the students to prove it.

Interestingly enough, this straightforward way of generating the set $\boldsymbol{S}$, using the first two axioms to produce a list of numbers, is not simple at all. Some understanding of basic concepts of computer science, like the concept of a queue as a data structure, is needed. We will however, not go into the details of the implementation here.

Before proceeding we should ask if this algorithm really generates the set $\boldsymbol{S}$. To be more specific, we should ask: If we had let ***Algor*** run *forever*, would it produce the infinite set $\boldsymbol{S}$?

Here a mathematical proof is needed. This is the first opportunity to introduce the notion of correctness. The teacher will decide how much effort (time) to devote to this notion here. It should be pointed out that this is not just an exercise in writing down the algorithm and later hopefully running it on a machine; we want, rather, to be sure that the algorithm is correct - that it actually does what it is expected to. We formulate this as a claim:

Claim 1:
The list generated by the algorithm ***Algor***, $\mathbf{L}$, equals the mathematically defined set $\mathcal{S}$.

Proof: $\mathbf{L}$ satisfies the axioms A1 and A2 (this is how it was constructed), and since by definition $\mathcal{S}$ is the minimal set satisfying A1 and A2, hence $\mathcal{S} \subseteq \mathbf{L}$.

For convenience we now denote the elements of $\mathbf{L}$ by $a_0, a_1, a_2, \ldots, a_i, \ldots$. Let l be an element in $\mathbf{L}$. For some i, $i \geq 0$, $l = a_i$. It can now be proven by induction that for every i, $i \geq 0$, $a_i \in \mathcal{S}$. Hence $l \in \mathcal{S}$, hence $\mathbf{L} \subseteq \mathcal{S}$, and $\mathbf{L} = \mathcal{S}$. Q.E.D.

Having proven the correctness of the algorithm, we can now refer to our set as $\mathcal{S}$ or as $\mathbf{L}$, since we have seen that both consist of actually the same elements. As a consequence we have another important claim with the aid of which our set becomes much more concrete.

Claim 2:
Let x be a non-negative integer, $x \in \mathcal{S}$ iff
$x = 0$ or there exists an element y, $y \in \mathcal{S}$, such that $x = 2y + 1$ or $x = 3y + 1$.

Now, the algorithm *Algor* can be used to show that there are many more sets that satisfy axioms A1 and A2, but do not satisfy A3. As a matter of fact there are infinitely many such sets. For example, if we add the number **2** to the list generated above, and all the numbers "generated" from **2**, by means of A2, that is by means of *Algor*, we will get the additional numbers: 2, 5, 7, 11, 16, 15, 22, 23, 34, 33, 49.... The union of the new generated sequence and $\mathbf{L}$ satisfies axioms A1 and A2, but it does not satisfy axiom A3.

Following the example above, the teacher can now raise a more general question: Given an element a, $a \notin \mathcal{S}$, will replacing A1 with $a \in \mathcal{S}$ yield a different set? We will probably get a new and different list. To find out how many different lists we can produce, we ask how many different such elements a are there? The students will be guided to find out that there is an infinite number of such elements, by observing that the complement of $\mathcal{S}$, $N - \mathcal{S}$ is an infinite set.

What we have done by now is defining a set, and trying to get the feeling of it. We have done this by writing an algorithm which generates the elements of this set. Now, we would like to ask where does the computer come in? What is its role? Let us turn to the next section.

From an algorithm to a computer program

"The automatic computer really *forces* the precision of thinking which is alleged to be a product of any study of mathematics," said George Forsythe in [F]. Indeed, before proceeding, a very crucial question has to be asked: Can a computer run our algorithm at all? Obviously, we have to translate the algorithm into a programming language. But this will not suffice, and it is not the main point here. We have already hinted that we may have some problems "executing" this algorithm. "If we would have let the algorithm run *forever*..." we said; obviously we cannot. If an algorithm or a computer program runs forever, we unfortunately, will not be around to see the results.... We don't want it to run forever. We want it to get some input, run for a finite amount of time, and produce some output, hopefully the desirable output.

If we want our algorithm to be implemented on a computer, we have to define better the algorithmic problem we want to solve. The set $\mathcal{S}$ is an infinite set, but, our algorithm will obviously not produce an infinite set. This means that we have to define the *finite* subset of $\mathcal{S}$ that we want the algorithm to generate. This point must be very well clarified in class before proceeding. Here is the right place to introduce the notion of *algorithmic problem*, as we have to define the specific *algorithmic problems* we want to solve.

The notion of algorithmic problems, a term coined by Harel [H], is a very important one in the field of computer science. If the students are not yet familiar with this notion, it is recommended to devote some time and effort on this issue. We only very briefly mention that an algorithmic problem consists of: A characterization of a legal, possibly infinite, collection of potential input sets, and a specification of the desired outputs as a function of the inputs. The solution to an algorithmic problem is an algorithm, which when executed on a legal input set, is expected to produce the output as required.

Back to our problem then. We can easily find out that there are several appropriate algorithmic problems that we can formulate. Here are two examples:

Example 1: Given an integer n, generate all the elements of $\mathcal{S}$ defined by axioms A1-A3, that are smaller than n.
Example 2: Given an integer n, generate the first n elements of $\mathbf{L}$ defined by axioms A1-A3.

One way of solving the problem given in the first example, is generating the first elements a_0, $a_1 \ldots$, $a_{n-i}, \ldots$, noticing that $a_i \geq i$; then throwing away elements that are greater than $n - 1$. This is a *generation problem* which leads to *Example 2*.

Another way of solving the problem of the first example is asking if each integer i, $0 \leq i < n$, belongs to $\mathcal{S}$. This leads to the *membership problem* which we will discuss in the next section.

The membership problem, or is 1001 in $\mathcal{S}$?

Before discussing the membership problem, the teacher may want to take advantage of this opportunity to introduce another concept from algorithmics, the concept of *decision problems*. Decision problems are the algorithmic problems that do not produce any "real" outputs, but only a "yes" or a "no". In the world of algorithmic problems there are many well known problems that belong to the category of decision problems. The teacher is advised to point out such algorithmic problems, and to encourage the students to find more problems of this kind.

The problem presented here is a decision problem, as can easily be observed. The answer we are looking for is a "yes" or "no". How to produce this one-word answer is a different story and not necessarily a short one. In Engel's book two different algorithms are given, both of which produce a table of a subset of the first n elements of the set $\mathcal{S}$. We can change these algorithms to produce a table of say, 1200 elements, and then look up the table to see if 1001 is included.

The first algorithm given in Engel's book, ***inset***, is a recursive algorithm. It is assumed that the students are familiar with the concept of recursion, and that they have written recursive algorithms before. However, if this is not the case, the teacher has to devote time and effort to teach the problematic topic of recursion. The teacher may not be interested in doing so at this point, and may therefore skip the presentation of this algorithm. This is not a problem and the teacher can proceed to the next algorithm.

The algorithm we present here as a Pascal function, which determines if a given non-negative integer x is in S, differs only slightly from the one given by Engel. The algorithm is based on the following claim – *Claim 3*, which is a straightforward result of *Claim 2*.

Claim 3:
Let x be a non-negative integer, $x \in S$ iff
$x = 0$ or
$x - 1$ can be divided by **2** and $(x - 1)div2 \in S$ or
$x - 1$ can be divided by **3** and $(x - 1)div3 \in S$.

```
function  InS(x: integer): boolean;
var y: integer;
begin
if x = 0 then InS:=true
else begin
y:=x-1;
if (y mod 2<>0) and (y mod 3<>0)
then inS:=false
else if y mod 2<>0
then inS:=inS(y div 3)
else if y mod 3<>0
then inS:=inS(y div 2)
else inS:=inS(y div 2) or  inS(y div 3)
end
end{InS}
```

The proof of *Claim 3* paves the way to proving the correctness of the algorithm, which the teacher may want to emphasize. It is recommended that the students solve a few membership problems with the aid of the algorithm, either in the laboratory or in class.

The generation problem – revisited

To conclude we would like to present an algorithm for the generation of the first n elements of **L**, which is based on a recursive definition of **L**. The algorithm itself is not recursive. The teacher should not fail to use this opportunity and point out the difference between a recursive definition of a concept and a recursive algorithm. *Claim 4*, which has to be proven mathematically (in class or as a home exercise), defines **L** recursively.

Claim 4:
L can be defined as follows:
$a_0 = 0$, $a_1 = 1$, and for every i, $i \geq 1$,
$a_{2i} = 2a_i + 1$, $a_{2i+1} = 3a_i + 1$.

Below is the algorithm given as a Pascal program which outputs all the elements of **L** from a_0 to a_{n-1}:

```
program generate (input,output);
const  n = 10000;
var i: integer;

a: array[0..n] of integer;
begin

a[0]:=0; a[1]:=1;
for i:=1 to (n-1) div 2  do
begin

a[2*i]:=2*a[i]+1;

a[2*i+1]:=3*a[i]+1
end;
for i:=0 to n-1 do write(a[i])
end.
```

As mentioned before we can now solve the problem of *Example 1* formulated in the previous section. To solve this problem it is sufficient to generate the elements $a[0]$ to $a[n-1]$, and ignore the elements that are greater than n.

Summary

The aim of this work was to show by means of a case study how algorithmic thinking and mathematical thinking can be integrated in high school or college curriculum. The idea is to introduce a subject into the mathematics or the computer science curricula, and to then discuss it in a way that leads to small research questions, and encourages the algorithmic approach.

From the computer science and mathematics curricula we discussed topics like algorithmic problems, and algorithms, decision problems and recursion, and touched upon issues like the correctness of algorithms, and data structures. However, we did not expect to cover or complete in any way all the aspects of the given example, or the issues we mentioned or pointed out for discussion. There are many more questions to be discussed, which may lead to interesting "mathematical discoveries" made by the students, and may serve as an opening to more profound theoretical issues.

We can ask for example: What happens if we change the three axioms to:
A1. $\mathbf{0} \in \mathcal{S}$,
A2. $x \in \mathcal{S} \rightarrow f(x) \in \mathcal{S}$ and $g(x) \in \mathcal{S}$,
A3. $\mathcal{S}$ is the minimal set satisfying A1 and A2.

This will lead to the more complicated topics of computer science and mathematics, asking what if $f(x)$ and $g(x)$ are computable, and what if they are not.

There are more profound issues that can be discussed. We mentioned the complexity and correctness of algorithms, but we did not go into detail, not even analyzing the complexity of

the algorithms we presented. However as we mentioned above, we definitely prepared the ground for further study of these subjects, in a later stage of the students learning.

It should be emphasized, however, that even presenting in detail only what we have in this paper, might constitute a crucial contribution to the students' mathematical education as a well-established basis for further studies. It might not be suitable for the entire student population of high schools and colleges; it was not meant to. It was designed for students with a greater and deeper interest in mathematics and computer science, or in logical thinking. We are sure that the students who have a strong interest in mathematics will find this example most attractive.

Acknowledgment
We wish to thank our colleague A. Arcavi with whom we had very stimulating and fruitful discussions.

Bibliography

[E] Engel, A., (1993), *Exploring Mathematics with Your Computer*, Mathematical Association of America, pp. 50-51.

[F] Forsythe George E., (1959), The Role of Numerical Analysis in an Undergraduate Program, *American Mathematical Monthly*, **66**, pp. 651-662.

[GZ Gal-Ezer, J. and G. Zwas, (1996), A Note on Algorithmic vs. Instrumental Thinking in Mathematics Education, in preparation.

[H] Harel, D., (1987), *Algorithmics: The Spirit of Computing*, Addison-Wesley.

[K1] Knuth, D. E., (1974), Computer Science and its Relation to Mathematics, *American Mathematical Monthly*, **81**, pp. 323-343.

[K2] Knuth, D. E., (1985), Algorithmic Thinking and Mathematical Thinking, *American Mathematical Monthly*, **92**, pp. 170-181.

[M] Maurer, S. B., (1985), The Algorithmic Way of Life is Best, *College Mathematical Journal*, pp. 2-18.

[MR] Maurer S. B. and A. Ralston, (1991), *Discrete Algorithmic Mathematics*, Addison-Wesley.