Post's Machine.

Liesbeth De Mol
Center for Logic and Philosophy of Science
Blandijnberg 2, 9000 Gent, Belgium
elizabeth.demol@ugent.be

Abstract

In 1936 Turing gave his answer to the question "What is a computable number?" by constructing his now well-known Turing machines as formalisations of the actions of a human computor. Less well-known is the almost synchronously published result by Emil Leon Post, in which a quasi-identical mechanism was developed for similar purposes. In 1979 these Post "toy" machines were described in a little booklet, called "Post's machine" by the Russian mathematician Uspensky. The purpose of this text was to advance abstract concepts as algorithm and programming for school children. In discussing this booklet in relation to the historical text it is based on, the author wants to show how this kind of ideas cannot only help to teach school children some of the basics of computer science, but furthermore contribute to a training in formal thinking.

1 Introduction

If one would ask a man in the street: what is a computation?, he would probably refer to some of the basic operations of arithmetics like addition, multiplication, substraction, or he would point to that which is performed by his computer. Intuitively clear though as this concept might be, it has been far from trivial to find a mathematically satisfying characterization of this notion. In 1936, the seminal paper by Turing *On computable numbers with an application to the Entscheidungsproblem* [23] was published. While hardly read at that time (See [13]), it is now one of the most famous mathematical

answers to the question “What is a computation?”.
This mathematical answer showed itself important not only on the theoretical level, but also on the level of physical computability through the rise of the computer. While the influence of Turing’s paper on the development of the first computers should not be overestimated,[1] the formalization of the notion of a computation has played a significant role in their further development, especially on the level of theoretical computer science.
Nowadays the computer has become omnipresent in our society, having applications on all its levels going from science to warfare. Despite its significance, many people have no idea about how their computer works let alone know what a computer program is. The computational stuff underlying the computer remains hidden away behind the GUI. In this respect the question concerning the significance of making school children familiar with some of the basics of computing becomes an important one.
In 1979 the Russian mathematician Uspensky published a little booklet called *Post’s machine* [26] in which he describes how he introduced the abstract notion of e.g. a computer program to little school children. However, he did not start from real computers, nor programming languages, but from a theoretical construction, almost identical to Turing’s, called a Post machine. The original description of this “toy machine” is due to Emil Post [18] and was published in the same year as Turing’s paper.
The purpose of the present paper is to offer an analysis of the teaching methods proposed by Uspensky in relation to the historical text it is based on. In this way, it will be shown how these ideas cannot only be used to teach school children some of the basics of computing, but furthermore contribute to a training in formal thinking.

2 Emil Post’s Formulation 1

In 1936 Church, Post and Turing each described a formalism to be identified with certain related intuitive notions. Turing proposed the identification between computability and Turing machines [23], Church identified effective

[1]While this paper must of course have played its role in Turing’s own design of one of the first computers, one should be careful with respect to its influence on the ENIAC: while it was important for Von Neumann, its influence on Mauchly and Eckert’s contributions to the design of the Eniac is negligible. For more information on this theme see e.g. [2, 9, 11, 15, 16]

calculability with general recursiveness and λ-definability [3], while Post formalized the notion of solvability in terms of his formulation 1 [18].[2] Since all three formalisms were shown to be equivalent, the respective intuitive notions should also be regarded equivalent, at least, on acceptance of these identifications. While Church's and Turing's identifications are now well-known under the general heading 'the Church-Turing thesis', Post's formulation is less famous.

Post's suggestion of identifying the intuitive notion of solvability with formulation 1 however, was not his first proposal in this direction. Already in 1921, he had thought about another such thesis, identifying the notion of a generated set with a certain class of formal systems called systems in normal form, systems which are now known to be equivalent to Turing machines.[3] Furthermore he had already convinced himself of the unsolvability of the Enstcheidungsproblem,[4] and proven the unsolvability of certain other decision problems relative to normal form.[5] While Post was convinced of the computational power of systems in normal form, he was aware of the fact that:[20]

> [...] for full generality [for this thesis to be valid] a complete analysis would have to be given of all the possible ways in which the human mind could set up finite processes for generating sequences.

When he described formulation 1 about 15 years later, it might have been this kind of analysis he hoped to offer, however now not in terms of generated sets but in terms of solvability.

[2]In [10] an analysis is given of this 'confluence of ideas'.

[3]This identification was called Post's thesis by Martin Davis [6]. This is why the author will use the term "Post's second thesis" with regard to the identification proposed in his 1936 paper.

[4]Although an exact formal proof was lacking

[5]Post however did not submit these results to a journal. Only in 1941 did he submit a paper summarizing these results, which was not accepted – only a small portion of the paper was published as [19]. The complete paper was posthumously published in 1965 by Martin Davis [20]. For more information on this earlier work by Post in 1921, see [7, 17, 22].

2.1 Solvability of Decision problems

Already from the beginning of his 1936 paper it is clear what Post intended with formulation 1 ([18], p. 103):

> We have in mind a *general problem* consisting of a class of *specific problems*. A solution of the general problem will then be one which furnishes an answer to each specific problem.

Contrary to Church's and Turing's 1936 papers, Post's did not contain the proof of the unsolvability of certain decision problems. However, he did understand his formulation as being able to solve any decision problem which is considered intuitively solvable, just as Turing machines were understood as being able to compute anything which is considered intuitively computable.
A decision problem is a general problem consisting of a class of specific problems. It is considered solvable if and only if there is a general procedure which, when applied to each of the specific cases, results in a solution for each of these cases. An example of a solvable decision problem is the problem to determine for two arbitrary integers x and y whether x is a divisor of y. An example of an unsolvable decision problem is the problem to determine for a given Turing machine whether it will yes or no halt.
However how can one know whether a problem is solvable? If one already has a method that always leads to a solution, as is the case with the above mentioned problem of divisibility, one of course knows that the problem is solvable. But what about problems for which there is no such answer? How will one know whether one should give up on the problem, concluding that it is not solvable? It is exactly this general *mathematical* problem which makes the proposal of a formalization capable to solve anything considered solvable intuitively, more than significant. Indeed, only if one accepts such a formalization, one can prove that certain problems are not solvable. Not accepting any such 'limit' of solvability implies that one can never answer the question of whether a certain problem is solvable, except when a method is found to effectively it.

2.2 Finite-1 Processes: From workers in boxes to solvability.

How can one formally capture the notion of solvability? Post's answer to this question was originally called *formulation 1*. In this general formulation

of a solution of a decision problem, there are two basic concepts involved: the *symbol space* (tape) in which the work leading to a solution is to be carried out and the *set of directions* (instructions) which direct operations in the symbol space and determine the order in which the directions have to be performed.[6] The symbol space is a two way infinite sequence of *boxes* (cells). The *worker* or problem solver (carriage/ reading or recording head) can move and work in the symbol space, and is capable of being in, and operating in but one box at a time. A box can have two possible conditions: empty (blank) or marked (labelled) with one symbol, e.g. "|". One of the boxes is called the starting point.
Now what can our worker do, what is he capable of? The worker's work is limited to the following *primitive acts*:

(1) Marking the box he is in (assumed empty).

(2) Erasing the mark in the box he is in (assumed marked).

(3) Moving to the box on his right.

(4) Moving to the box on his left.

(5) Determining whether the box he is in, is or is not marked.

The worker's acts are ordered through a set of directions, to remain unaltered for every specific case of the decision problem one wants to solve. If one would for example translate a solution of the problem of divisibility into formulation 1, the set of directions will always be the same for every specific pair of integers. Every set of directions is headed by:

- Start at the starting point and follow direction 1.

The set always consists of a finite number of directions numbered $1, 2, 3, ..., n$. The i-th direction ($i \in \mathbb{N}^+$) always has one of the three following forms:

(A) Perform operation O_i ($O_i = (1), (2), (3)$ or (4)) and then follow direction j_i.

(B) Perform operation 5, and according as the box is marked or not marked follow direction $j_{i'}$ or $j_{i''}$.

[6] Every concept used by Post will be followed by the equivalent concept used by Uspensky in describing a Post machine (Cfr. 3.2.1)

(C) Stop.

Simple as the description of formulation 1 might seem, Post understood it as being capable to solve anything considered intuitively solvable. Any general decision problem for which one cannot find a fixed set of directions which, when applied to an input, generates a solution for each of the specific cases of the problem, should thus be considered as an unsolvable problem.
After the description of formulation 1, Post further explicates the identification between solvability and his formalism by adding several definitions.

Applicability A set of directions is called applicable to a general problem, if in applying it to any specific case of the problem, instruction (1) is never ordered when the box the worker is in is already marked, and (2) is never ordered when the box is unmarked.

Finite 1-process A set of directions is considered as setting up a finite 1-process relative to a given general problem if it is applicable to the problem and if the process it sets up terminates for each specific case of the problem.

1-solution A finite 1-process is a 1-solution of a general problem if the answer it gives to each specific case of the problem is always correct.

1-given A problem is *1-given* if a finite 1-process can be set-up which, when applied to the class of positive integers symbolized in a certain way in the symbol space, yields in a one-to-one fashion the class of specific problems constituting the general problem. This 1-givenness can be compared to Gödelnumbering: the possibility to translate a problem to numbers and vice versa.

We can now give Post's answer to the question of how to formally capture the notion of solvability: a problem is considered solvable iff. one can set up a finite 1-process relative to the problem (1-given) which is a 1-solution. For example, the above mentioned divisibility problem is solvable in this sense. The Entscheidungsproblem however is a 1-given problem for which there is no finite 1-process which is a 1-solution.

2.3 From solvability to computers

As was explained at the beginning of this section, the idea of finding a formal equivalent for intuitive notions such as solvability and computability

was rooted in the mathematical problem to determine for a general problem whether it is solvable (computable) or not. Since 1936 hundreds of mathematical problems have been shown to be unsolvable in a variety of domains, going from group theory to theoretical physics. Theses such as those proposed by Church, Post and Turing are the conditions sine qua non for these results to be valid.
Despite the significance of finding a formal equivalent for intuitive notions such as solvability for those mathematicians who want to determine whether there exists a general solution to certain problems, these developments were rather abstract at that time, linked as they were to the research on the foundations of mathematics. Indeed, e.g. Post's formulation 1 seems far away from our everyday life, describing workers operating and moving around in an infinite sequence of boxes, putting or erasing marks depending on the instruction he/she has to follow at a given moment. However, in connecting this formalism to the computer it becomes clear how concrete these theoretical ideas actually are.
With the rise of the computer, the notion of computability was given a physical interpretation. When Von Neumann was called upon by Goldstine to enforce the team which was already working on the construction (and had almost finished the design) of one of the first computers, the ENIAC, he wanted to change its design inspired by Turing's notion of a universal computing machine. Also Turing got involved in the design of one of the first computing machines called the ACE and explicitly stated the connection between real and abstract computing machines: "*Machines such as the ACE may be regarded as practical versions of this same type of machine.* [i.e. a universal Turing machine]"[24]. The theoretical ideas put forward by Church, Turing and Post however would prove even more significant in theoretical computer science. In rereading for example the paper by Post from a programming perspective, it becomes clear how closely connected these ideas are to modern concepts. Indeed, the similarity between on the one hand a finite set of instructions directing which operations have to be performed when, and on the other hand algorithms used on our computer, can hardly be ignored. Even the definition of a 1-given process has a close connection to modern computing. Translating a problem in a given language to another language is basic to any modern computer, both in writing a program as well as in moving and clicking a mouse to e.g. find and open files.
Given the simplicity of the formalisms such as those described by Turing and Post, together with their connections with computer science, it seems rea-

sonable to use them as tools to teach computer science. Indeed, the Turing machine is often used as a way to introduce the more abstract concepts of computer science on a graduate level. However, notwithstanding this simplicity together with the more general connections between computers and e.g. formulation 1, it is not immediately clear how the inner workings of a Post machine (formulation 1) can help one to advance basic aspects and problems of computer science *to schoolchildren*. Furthermore, why would one start from such abstract machines? Why not start from an existing computer or a programming language such as Basic?

3 Uspensky's little booklet

In 1979 Uspensky published a Russian booklet called *Post's machine*,[7] in which formulation 1 is used as a way to make school children familiar with computer science. As Uspensky states in the preface ([26], p. 7):

> This booklet is intended first of all for schoolchildren. The first two chapters are comprehensible even for junior schoolchildren. The book deals with a certain "toy" ("abstract" in scientific terms) computing machine – the so-called Post machine – in which calculations involve many important features inherent in the computations on real electronic computers. By means of the simplest examples the students are taught the fundamentals of programming for the Post machine, and the machine, though extremely simple, is found to possess quite high potentialities. [...] The author hopes that the present booklet can to a certain extent advance such concepts as "algorithm", "universal computing machine", "programming" in the secondary school, even in its earlier grades.

Furthermore, in drawing from his own experience, Uspensky is convinced that children from primary school and even pre-school should be able to cope with the basic notion of a "computation" on a Post machine ([26], p. 7):

> The author's personal experience makes him confident that the schoolchildren of primary school and even children of pre-school

[7]Published in English as [26].

> age can easily cope with "computations" on the Post machine (for instance, with the aid of paper tape, ruled in square sections, and the clips or buttons that are used as labels) and prepare the simplest programs (containing no transfer-of-control instructions)[8]

Two questions pose themselves here. First of all, why is a Post machine, given its abstract or "toy" character, an ideal instrument to teach school children some of the basics of computer science? Secondly, why would one start from an abstract machine and not a real machine? In order to answer these questions it is interesting to start with Uspensky's methodological notes ([26], pp. 19–22), "*addressed to those who are going to instruct schoolchildren of earlier grades on the Post machine*".

3.1 Methodological Notes

In the first part of the methodological notes Uspensky adds some small advices the teachers can follow which can be summarized as follows:

- Do not use concepts and technical terms which might be too difficult too grasp or lead to unnecessary confusion for a 7-year old school child. For example, it could be better to assume a finite instead of an infinite tape, and in presenting the material to junior school children, it is no good mentioning words such as "algorithm" or "Post machine". One can introduce these concepts later on.

- Do not use numbers, but rather some symbolic representations which can then be manipulated through the operations of a Post machine.

- Introduce new classes of instructions not all at a time but step by step, each instruction being followed and explained by visual examples and exercises.

The idea of not using difficult and/or abstract concepts initially and explaining the inner workings of a Post machine step by step are indeed important

[8]As is clear from this quote, while even children of pre-school age should be able to cope with the calculations of a Post machine, the preparation of more complicated programs, together with the more theoretical consideration on universal machines and Post's thesis are not meant for children of this age.

guidelines in introducing a Post machine to school children. Of more significance here however, is the idea of replacing numbers by another representation.

On the first page of the preface, Uspensky mentions in a footnote that he has put the word "computation" between double quotes because ([26], p. 7):

> [...] it is not in the least necessary that the initial data and the results of conversions executed on the machine be numbers. Operations with combinations of symbols having no numerical values are in a number of cases much more visual.

Since the word "computation" is *intuitively* closely connected to the idea of performing operations on numbers, Uspensky is careful in using this concept. In replacing numbers by another symbolic representation, the word "computation" itself exceeds our intuitive understanding of it – doing operations on numbers – and thus becomes far more general. While Uspensky gives a rather pragmatic reason to do this – operations with symbols different from numbers are more visual – these remarks should also be connected to what is considered the more significant part of the methodological notes here. To Uspensky it is very important that in introducing the Post machine one should never explain what it can be used for, before the student has clearly understood how it works ([26], p. 21):

> [...] any comment on what the given device is is used for should be delayed until it is clearly understood and the execution of programs becomes free and easy to grasp.[...] "I will tell you first what I do; I will tell you the reason afterwards." The capability of perceiving any system of concepts or any reasoning, in general (and regardless of) the purpose of the knowledge is obtained, i.e., before (and regardless of) any application seems one of the most important qualities which are trained by mathematical studies. Giving an idea of the goal you are after in presenting material [...] should not affect understanding which can and must proceed regardless of the goal. The ability to think formally is a special ability developing like every ability through training. This training can begin from an early age. The summation of multiple digit numbers and the simplest exercises with the Post machine can serve as the elements of such training, easy to grasp for primary school children.

One of the reasons to start from a "toy" machine such as Post's and not real computers is rooted in its abstractness. In this way, this kind of teaching cannot only help to advance certain aspects of computer science to school children, but can furthermore contribute to a training in the ability to think formally. It is in this respect that Uspensky's notes on using certain symbolic representations instead of numbers, become more significant. In first teaching the inner workings of a Post machine, using a symbol such as "V" – the one introduced by Uspensky – without making any reference to computing machines or numbers, a child will have a more formal and general understanding of the notion of a "computation". Only after this more general understanding, one can make explicit reference to the notion of a computation, by showing how one can program a calculation on a Post machine.
Uspensky thus differentiates between two stages in the Post machine teaching: the formal stage and a more practically oriented stage. After the completion of the formal stage – which is considered the more important stage by Uspensky – one can pass to the applications the Post machine can be used for.

3.2 Post machine teaching. The formal stage or how the Post machine works

In teaching how a Post machine works without making any reference to concrete applications of the machine, thus e.g. not even mentioning that it is an abstract computing machine, there are of course several aspects to be taught. In explaining the basics of a Post machine, Uspensky first describes its components, then the instructions followed by the possible classes of operations. After this more general description, he discusses some more examples and exercises.

3.2.1 "An outward appearance of the Post machine"

In order to explain how a Post machine works, it is of course best to start with a description of the components it exists of. However, where Post talked about a symbol space, a worker and boxes Uspensky uses a slightly different terminology. The symbol space is called "tape", the worker becomes "carriage" or "reading or recording head" and the boxes are now called "cells". For each new component introduced, it is important to explain the concept through a visualization, e.g. drawing a tape divided into cells on the blackboard, or using a paper tape divided into square sections. After the introduction of the components one can start to illustrate the so-called primitive acts of a Post machine: it can move left or right; it can "examine" a cell and it can label or erase a cell. All these actions should again be visualized through examples. The labelling can be done by using a certain symbol – Uspensky uses $\bigvee$ – or can be illustrated by putting and removing clips or buttons or any other kind of object in cells of a paper tape.

3.2.2 The program for the Post machine

After having explained the outward appearance of a Post machine one can start to teach how the different acts of the carriage are controlled through instructions. In the following table an overview is given of the different kinds of directions as described by Post compared to the description given by Uspensky.

Post	**Uspensky**
The set consists of a finite number n of directions to be numbered 1, 2, 3,..., n.	Idem
The machine always starts at a starting point, to follow instruction 1.	Idem
An instruction i is always of the following form ($i, j \in \{1, ..., n\}$):	
i. Perform operation (1) and then follow direction j (mark = |)	Instructions for printing the label: $i.\ \bigvee\ j.$
i. Perform operation (2) and then follow direction j	Instructions for erasing the label: $i.\ \xi\ j.$
i. Perform operation (3) and then follow direction j	Move-to-the-right instruction: $i.\ \Rightarrow j.$

$i.$ Perform operation (4) and then follow direction j	Move-to-the-left instruction: $i. \Leftarrow j.$
$i.$ Perform operation (5) and according as the box is marked or not, follow direction j or j'	Transfer-of-control instruction: $i.? \begin{array}{l} \diagup j \\ \diagdown j' \end{array}$
$i.$ Stop	$i.$ Stop

While Post never gave a kind of shorthand for writing a program in formulation 1, Uspensky does provide for such a notation which is necessary if one really wants to give examples of programs in a Post machine.
After having described the abstract form of the several instructions, he gives several more specific examples like e.g.:

$$25.? \begin{array}{l} \diagup 32 \\ \diagdown 25 \end{array}$$

Uspensky adds two further conventions for a program to be a program for a Post machine which were not taken into consideration by Post, but are basic in this kind of teaching: an instruction with number k occupies the k-th place in the list of instructions and to every jump of an instruction in the list, there corresponds an instruction whose number equals the number of the jump under consideration. The definition of a program can then be (and is) further explained by giving examples of lists of instructions which are and which are not programs for a Post machine.
After having defined the length of a program as the number of instructions a program exists of, Uspensky adds the exercise to first write down all the possible programs of length 1, then determine how many programs there are of length 2 and length 3 and consequently how many programs there are of length n. This exercise beautifully illustrates how the 'simple' Post machine can indeed be used to train the ability to think formally. Starting from specific cases in order to solve the more general case is one of the typical methods of mathematics.

3.2.3 The operation of the Post machine

After the description of a Post machine one can start to make it work. The machine starts from a certain configuration (the initial state): the carriage is facing one of the cells, some of the cells on the tape are labelled others not, and the machine can start to run a program by following the first instruction in the list. In analyzing several different short and simple programs, Uspensky illustrates how one can easily show that there are several possible operations a Post machine can accomplish in following a list of instructions, starting in a certain initial state. There are three different operations a Post machine can complete:

O_1 The machine gets into a non-executable state: it has to label a cell already labelled, or to erase a non-labelled cell. Then the execution is stopped and a no-result halt takes place.[9]

O_2 The machine comes to a halt instruction. The machine has accomplished the program.

O_3 The machine never halts nor comes to a non-executable instruction: it gets into an infinite loop.

Of course the school child can only get a grasp on these three possible outcomes if he/she has went through several examples of programs.

3.2.4 Examples of performing programs

In the last section on the formal stage, Uspensky illustrates why examples are not only important with respect to the general understanding of how a Post machine works, but are, maybe even more, significant in coming to an understanding of certain features of a Post machine which would be quite inaccessible if one would restrict oneself to pure description. He not only gives further examples of several programs leading to O_1, O_2 or O_3 but also makes clear, through the examples, that different programs applied to the same input might lead to different outcomes O_1, O_2 or O_3, but not necessary, and that, vice versa, the same program applied to different inputs can also lead to different outputs, but not necessary. Simple though as these two

[9]In the methodological notes, Uspensky gives the advice not to use "a no-result halt" but rather to speak about the machine getting out of order in the class room.

insights might seem, it will be shown later that these two features are basic to programming (Cfr. 3.3.1).
At the end of this section, two exercises are added. In the first exercise the following questions have to be solved: Can there be a program that always halts, never halts or results in a non-executable instruction, no matter what the input is? What is the minimal length for each of these programs? The second exercise asks for a program which can perform O_1, O_2 and O_3 by feeding it different inputs. The student is furthermore asked to prove that the minimal number of instructions for such a program must be 4, and to write down all such programs of length 4.
These exercises are important for several reasons. Again Uspensky teaches through the exercises how in starting from specific cases, one can find a way to answer a more general case of the problem at hand. In this respect such exercises attribute to a training in formal thinking. Furthermore questions concerning minimal lengths of programs are fundamental in computer science. The child is made familiar with these problems without being aware of their practical significance and thus possibly gaining a more abstract grasp on this problem.
After this formal stage of the Post machine teaching, having gone through the details of a Post machine, making extensive use of visual examples and some more theoretical exercises, one can start to explain what the Post machine can actually be used for. However, in not having explained this before, the child will now have a more general understanding of a Post machine – and thus a computing machine. Since no specific meaning was attached to the operations and algorithms performed by such a machine, the introduction of the idea that such a machine is able to perform computations leads to a more general understanding of the notion of a computation itself.

3.3 Post machine teaching. The applicative stage or how to add 1 to a number.

Uspensky only discusses one application of the Post machine in all its details, namely unary addition or the computation of "+ 1". Trivial as this example might seem, its (lengthy) analysis shows how in starting from a seemingly simple machine, an arithmetical operation intuitively considered as the most basic step possible, becomes far from trivial.

3.3.1 "+1" and the fundamental problems of programming

The analysis and description of a Post machine in the formal stage, as well as its above given original formulation by Post, makes it not immediately obvious how this abstract machine can be linked to more practical concerns of programming. Exposing this close connection between an abstract formalism and real computers, is one of the main goals behind Uspensky's analysis of the problem of unary addition, since it shows how a Post machine can indeed be applied to real programming:

> Analysing the addition of unity on the Post machine (though as simple as it is) enables us to acquaint the reader (in a very simplified form, of course) with problems arising in real high-speed computers. The point is that the principal mathematical problem that faces us in operation on the computers remains the same both for physically existing and "abstract" machines. This problem is *preparing a program for the machine leading to a given goal.*

Some of the typical features (problems and techniques) of programming which become apparent in the analysis of the problem of unary addition, and thus illustrate how a Post machine can effectively be used to teach school children some of the basics of programming, are:

F1 Different programs lead to the same goal.

F2 In making an input more general, the programming becomes much more difficult, and the program itself can become more complex i.e. generalization leads to complication.

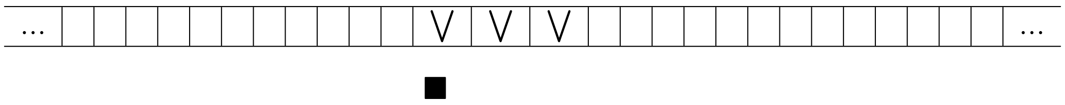
F3 In writing programs for more general or complex problems, it can be very useful to recycle programs already written.

F4 The significance of the place the input is stored in the memory.

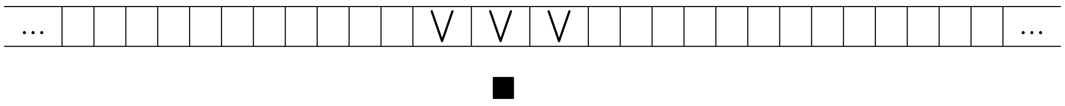
F5 To find the shortest program for a certain problem. This can be decisive in the context of real computers – a program *can* be made more efficient in this way.

Uspensky shows how all these features indeed pop-up in the context of programming the simple arithmetical operation of adding 1 to a number on a

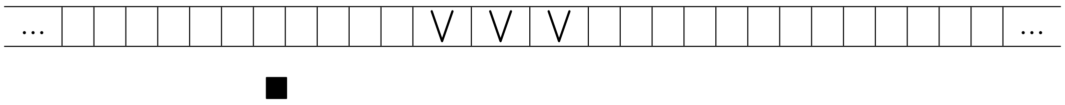
Post machine, in considering more and more general initial configurations.
A number n is represented by $n+1$ consecutive labelled cells. The number 1 e.g. is represented as: $\bigvee\bigvee$. In starting e.g. from the number 1 we now have to find a program which, when halted, results in: $\bigvee\bigvee\bigvee$. There are 5 consecutive generalizations on the initial configuration for which Uspensky shows how one can find an algorithm and how programming these several cases is closely connected to F1-5. The several cases are determined by the position of the carriage relative to the input.
In the simplest case, the carriage is positioned at the most left or most right labelled cell of a sequence of labelled cells representing the number to which 1 has to be added:



In the more complicated case (case 2), the carriage cannot only be positioned at the extreme left or right of the input, but can be in any one of the labelled cells:



In the yet more complicated case (Case 3) the carriage examines a blank cell in the initial state, although it is "known" in advance to which side of the labelled cells the carriage is: we can write a program for the case the carriage is to the left of the labelled cells, and one for which it is to the right of the labelled cells:



In the fourth even more complicated case, the carriage can be examining a cell somewhere to the left of the initial input, or examining one of the labelled cells (a solution for the case where the carriage can be examining a cell somewhere to the right of the initial input, or examining one of the labelled cells is considered equivalent) The most general, and thus most complicated case, is the case for which it is not known at all in advance which cell the

carriage is scanning in the initial state. It can be examining a cell to the left or to the right of the input or a labelled cell part of the input.
These cases clearly illustrate how a seemingly simple operation to be programmed on a Post machine is connected to features F2 and F4. However, in teaching how one can prepare a program to solve each of these cases, not only all other features mentioned can be made familiar to a school child, but the ability to think formally can be further trained. First of all, in making the input more and more general it is shown that the consequent solution to the problem becomes more complex – in this way a general feature not only of programming but of many other mathematical problems becomes apparent. Secondly, in discussing the first three cases Uspensky always first gives a lengthy statement of the problem and then a short statement. In this way he shows how one can make the transition from descriptions in normal language to a formal language, a language which clearly becomes more and more important if not necessary (given the space it would take to give a lengthy description) the more general the problem becomes. Learning to understand the formal description of a problem and its solution is basic to the ability to think formally.
In discussing case 1, Uspensky prepares several algorithms for solving the case, the shortest possible program included, and as an exercise asks to prove that there are in fact infinitely many programs to solve this problem. In solving case 2 and 3, the significance of F1 and F5 is again taken into focus through the exercises, one of them being to prove that there are exactly 12 programs of length 4 which contain a move-to-the-left instruction that solve case 2, and another one asking for a proof of the fact that the shortest solution to case 3 is an algorithm of length 5. Again these exercises not only help to teach some of the more general features of programming, but furthermore train the ability to think formally, always starting from a specific case or example, and then abstract from this case to find a more general solution. Given the description of case 4, the case for which initial positions from case 1, 2 and 3 are combined, it is clear that F3 plays an important role here. Indeed, Uspensky shows how one can recycle the algorithms from the simpler cases to solve case 4. He even gives a kind of algorithm to combine algorithms and to shorten the resulting algorithm by showing how an instruction "absorbs" another one. In this way the first combined algorithm is reduced from 10 to 8, to 7 to 6 instructions.
While finding a solution for the previous cases is rather straightforward if one went through the formal stage, notwithstanding the problem of finding

shortest algorithms, the most general case of the problem of unary addition is surprisingly complicated, needing 23 instructions, and even asks for another kind of notation. One of the possible algorithms to solve case 1 e.g. looks like this:

1. $\Longleftarrow 3$
2. stop
3. $\bigvee 2$.

If one however needs 23 instructions instead of 3 a representation such as that for case 1, becomes very complicated and difficult to follow. In order to analyze and understand how such a long program works, without losing an overview of the general solution, Uspensky introduces a more structured way of writing down programs, using diagrams and block diagrams (comparable to flow charts). In figure 1, one can see the diagram for the solution of unary addition in the most general case, for which each number corresponds to an instruction.
The instructions are now grouped, each group corresponding to small subroutines in the program. This kind of notation leads to a more structured code more easy to understand. In further introducing block diagrams, each group of instructions is given a more general name, indicating why each specific "subroutine" is needed in the program. E.g. instructions 1-4 taken together form the "start block". In this subroutine the machine is arranged such that every initial state the carriage is in, is changed to the same state: the carriage is positioned such that the cell examined and the cell to the right of this cell are both blank.
After a detailed analysis of the generalized unary addition algorithm, Uspensky goes on to consider algorithms not to add 1 to a number but to add an arbitrary number to a number. Again, starting from the simple case, it is shown how an algorithm becomes more complex, the more general the problem becomes. The most general case of addition of numbers, is addition of an arbitrary number of numbers at an arbitrary distance from each other. Uspensky explains that this problem cannot be solved by a Post machine. Indeed "*however long the carriage has travelled along the tape, it "will never know" whether it has already bypassed the records of all the addends.*" ([26], p. 62).[10] In discussing this problem, he again emphasizes the significance of

[10]This of course does not mean that the problem to add an arbitrary number of numbers is not computable by a Post machine. It only means that one needs a specific configuration to start from if one wants to solve this problem.

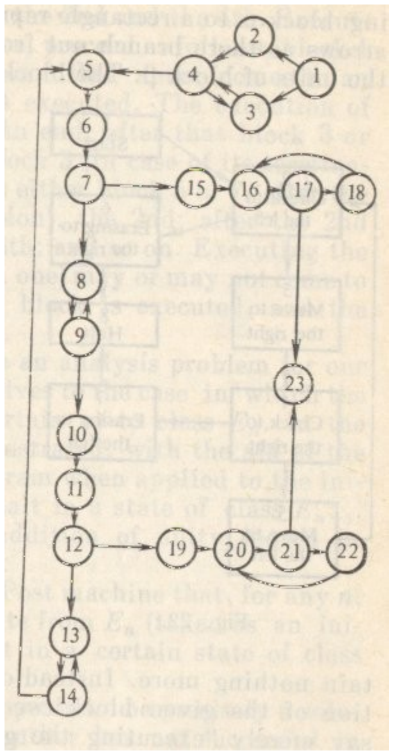


Figure 1: Figure 1: Block Diagram Unary addition

F4: depending on the place where the input is stored relative to the carriage, certain problems are solvable or not by a Post machine.
This further exploration of the seemingly simple problem of adding numbers is followed by a discussion of the idea of identifying the notion of an algorithm with a Post machine program, a description of the notion of a universal machine and a note on the differences and similarities between a Post machine and a real computer. Interesting as these further remarks might be, they fall beyond the scope of this paper. Of more significance here is the fact that in starting from a very simple machine, Uspensky has shown how one can indeed advance the basics of programming and computing to school children and ultimately of theoretical computer science.
However, as was asked at the beginning of this section why is an abstract machine such as a Post machine the ideal instrument to reach this goal? As was shown, basic to Uspensky's proposal is the differentiation between the

formal stage and the more practical stage. In starting from a Post machine without making any reference to what it is used for, this kind of teaching can contribute to a training in formal thinking. In making abstraction from the applications, school children might get a broader conception of the notion of a computation. Through the detailed analysis of unary addition, Uspensky's booklet not only beautifully illustrates how such abstract machines are closely connected to some basic features of programming, but furthermore shows how that which is intuitively considered as the most basic step of computing, is far from basic. In this way, this training in formal thinking is further pursued in teaching some of the applications of a Post machine. Indeed, through the analysis of the unary addition case, a child learns that there is nothing fundamental about the way one usually calculates, it all depends on the formal framework one is starting from. Understanding that there are several different ways to do something, can make it more easy to make abstraction from the specific meanings and interpretations involved when working with one specific kind of system and can thus further attribute to a training in formal thinking – an ability basic to programming and computing.

4 Conclusion

Since its first calculations for the A-bomb, the computer has become an object on which whole societies rely on many different levels: from typing and saving a simple text to controlling flights of airplanes to avoid collision. Despite its omnipresence however, many people have no clue about what lies beyond their interface. Not knowing what one is using, while heavily depending on it, can at least be called a disquieting thought. This is exactly why Uspensky's suggestion should be taken into account again.
But why should one start from abstract machines rather than from real ones? It is a well-known fact that abstracting from real computers is basic to a better understanding of what programming is. As is stated in one of the early books on the mathematical character of programming ([8], p. 5):

> It is abundantly clear that significant progress in programming, in reasoning about programs, and in the design of programming languages will only materialize, provided we learn how to do this, while temporarily *fully* ignoring that our program texts also admit the interpretation of executable code, because it is precisely

> that interpretation that clutters up our minds with all the computational processes, which truly baffle the imagination. In short: for the effective understanding of programs we must learn to abstract from the existence of the computer.

In a recent paper [4] Post's machine was even used as the basis for the development of a "microscopic" software environment to be used as an introduction to formal methods of programming in the framework of an introductory computer science course. The motivation behind the paper is that despite the significance of "formal programming" for programming itself, it is often not included in the curriculum due to its high mental cost.
The fact that "formal programming" is indeed considered very difficult, such that one even has to urge for its teaching beyond high school level, makes Uspensky's suggestion, based on his experiences as a teacher, even more apparent. In following his booklet it indeed becomes very clear how a "toy" machine can not only help to create a better understanding of computing as such, but also contributes to a training in formal thinking. While often considered irrelevant for practical life, this ability is in fact fundamental to it and should thus not remain restricted to the graduate level. Teaching with a Post machine, or any other similar machine, seems to be an ideal instrument not only to create a better understanding of the object that affects so many aspects of our lives, but also to contribute to this important ability. As an abstract counterpart of the modern computer, it forms a direct link between certain aspects of our everyday life and mathematics.

References

[1] R.S. Boyer and J Strother Moore (eds.), *The correctness problem in computer science;*, International Lecture Series in Computer Science, London, Academic Press, 1981.

[2] Arthur W. Burks, *From eniac to stored-program computer: Two revolutions in computers.*, [14], 1980, pp. 311–344.

[3] Alonzo Church, *An unsolvable problem of elementary number theory*, American Journal of Mathematics (1936), no. 58, 345–363, Also published in [5], 108–109.

[4] V. Dagdilelis and M. Satratzemi, *Post's machine: A didactic microworld as an introduction to formal programming*, Education and Information technology **6** (2001), no. 2, 123–141.

[5] Martin Davis, *The undecidable. Basic papers on undecidable propositions, unsolvable problems and computable functions*, Raven Press, New York, 1965, Corrected republication (2004), Dover publications, New York.

[6] ______, *Why Gödel didn't have church's thesis*, Information and Control **54** (1982), 3–24.

[7] ______, *Emil L. Post. His life and work.*, Solvability, Provability, Definability: The collected works of Emil L. Post [21], 1994, pp. xi–xviii.

[8] E.W. Dijkstra, *Why correctness must be a mathematical concern*, [1], 1981, pp. 1–8.

[9] J. Presper Eckert, *The eniac*, [14], 1980, pp. 525–540.

[10] Robin Gandy, *The confluence of ideas in 1936*, Published in [12], pp. 55–111.

[11] Herman H. Goldstine, *The computer from pascal to von neumann*, Princeton University Press, Princeton, 1972.

[12] Rolf Herken (ed.), *The Universal Turing machine*, Oxford University Press, Oxford, 1988, Republication (1994), Springer Verlag, New York.

[13] Andrew Hodges, *Alan turing. the enigma*, Burnett Books, London, 1983, Republication (1992), 2nd edition, Vintage, London.

[14] J. Howlett, Nicolas Metropolis, and Gian-Carlo Rota (eds.), *History of computing in the twentieth century*, Academia Press, New York, 1980.

[15] John M. Mauchly, *The eniac*, [14], 1980, pp. 541–550.

[16] Scott McCartney, *Eniac: The triumphs and tragedies of the world's first computer*, Walker & Co, New York, 1999.

[17] Liesbeth De Mol, *Closing the circle: An analysis of emil post's early work.*, The Bulletin of Symbolic Logic **12**, no. 2, 267–289.

[18] Emil Leon Post, *Finite combinatory processes - Formulation 1*, The Journal of Symbolic Logic **1** (1936), no. 3, 103–105, Also published in [5], 289–291.

[19] ———, *Formal reductions of the general combinatorial decision problem*, American Journal of Mathematics (1943), no. 65, 197–215.

[20] ———, *Absolutely unsolvable problems and relatively undecidable propositions - account of an anticipation*, [5], 1965, pp. 340–433.

[21] ———, *Solvability, provability, definability: The collected works of Emil L. Post*, Birkhauser, Boston, 1994, edited by Martin Davis.

[22] John Stillwell, *Emil Post and his anticipation of Gödel and Turing*, Mathematics Magazine **77** (2004), no. 1, 3–14.

[23] Alan Turing, *On computable numbers with an application to the entscheidungsproblem*, Proceedings of the London Mathematical Society (1936), no. 42, 230–265, Also published in [5], 116–151.

[24] ———, *Lecture to the london mathematical society on 20 february 1947.*, A.M.Turing's ACE Report of 1946 and Other papers (Cambridge, Massachusettes) (B.E. Carpenter and R.N. Doran, eds.), MIT Press, 1986, Also published in [25], pp. 87–105, pp. 106–124.

[25] ———, *Collected works of A.M.Turing. mechanical intelligence.*, North-Holland, Amsterdam, 1992.

[26] Vladimir A. Uspensky, *Post's machine*, Mir Publishers (Little Mathematics Libarary), Moscow, 1983, Originally published in 1979. Translated from Russian by R. Alavina.