Little Mathematics Library

V. A. USPENSKY

POST'S MACHINE

Mir Publishers · Moscow


ИЗДАТЕЛЬСТВО
МИР

ПОПУЛЯРНЫЕ ЛЕКЦИИ ПО МАТЕМАТИКЕ

В. А. УСПЕНСКИЙ
МАШИНА ПОСТА

ИЗДАТЕЛЬСТВО «НАУКА» МОСКВА

Little Mathematics Library

V. A. Uspensky

POST'S MACHINE

Translated from the Russian
by
R. Alavina

MIR PUBLISHERS
MOSCOW

First published 1983
Revised from the 1979 Russian edition

На английском языке

CONTENTS

PREFACE

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—on 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 reader is not expected to have any knowledge of mathematics beyond the primary school curriculum.

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. The author's personal experience makes him confident that the schoolchildren of primary school and even children of pre-school age can easily cope with "computations"* on the Post machine by the preset program (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). This is precisely why the first chapter, easiest to grasp by junior schoolchildren, includes a special section "Methodological Notes".

Over forty years ago an article by Emil L. Post, an outstanding American mathematician, appeared in *The Journal of Symbolic Logic*. The article was called "Finite combinatory processes—formulation 1" (it is presented as a supplement to this booklet). In this article and in the article "On computable numbers with an application to the Entscheidungsproblem" by the British mathematician A. M. Turing, published in *Proceedings of London Mathematical Society* at the same time, there were given the earliest definitions of the concept of "algorithm", one

* The word "computations" is taken in quotes because 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.

of the central in mathematical logic and cybernetics. It is playing an increasingly important part in automation and so in the whole life of modern society.

The definitions of the concept of "algorithm" proposed by Post and Turing are of importance nowadays. The Turing machine is constantly employed as a working device in the modern theory of algorithms. The Post machine is less popular though, or because, it is simpler than the Turing machine*. The works mentioned are notable for one more reason: they preceded in an abstract form the basic principal features of computers a few years before large-scale (so-called universal) computers appeared (first not even electronic but electromechanic). The very devices proposed by Post and Turing were formulated as certain "abstract machines". It was done in an apparent form by Turing and more obliquely by Post who did not even use the term "machine". Turing's reasoning is often cited in literature on the theory of algorithms including popular science. As to Post's reasoning, despite its utmost simplicity compared with Turing's, it was for a long time not presented either in specialized or in popular science literature**, except for Post's original article. At the same time it is Post's reasoning, as the author's teaching experience shows, that can make as natural an introduction to the theory of algorithms as Turing's.

It is the concept of the algorithm suggested by Post that the present booklet is dedicated to. The version is somewhat modified and, in particular, takes on the form of the description of a certain abstract computing machine. That is why it seems appropriate to introduce terms differing from Post's.

The given issue of "Little Mathematics Library" is based on the lectures delivered by the author to schoolchildren as well as lectures delivered to the students of the mechanics and mathematics, and philology departments of Moscow University since the 1961-62 academic year.

---

* The Post machine is simpler than the Turing machine in that its elementary operations are simpler and the recording technique is less diverse. These are the reasons, however, why recording and processing of data on the Post machine demand, generally speaking, bigger storage capacity and more steps than on the Turing machine.

** In 1967 the author published a series of four articles underlying this booklet in *Mathematics in School*, 1-4 (in Russian).

1. HOW THE POST MACHINE WORKS

Sec. 1.1. "An Outward Appearance" of the Post Machine

First of all we must warn the reader that the Post machine is not a really existing device; that is why the words "an outward appearance" are taken in quotes. The Post machine as well as its near relative, the Turing machine, is an imaginary device, existing only in our imagination* (though it could in principle have been made "of metal"**). It is this fact that is meant when they say that the Post and the Turing machines are "abstract" calculating machines. That the Post machine does not practically exist will, however, be of no consequence to us. Just the opposite, we shall, for vividness, assume it "as if existing". And just as we can learn to calculate on a counting frame or on a slide rule without having them before us but only using their description and imagining them, so we will learn how to calculate on the Post machine summoning our imagination and using the description given below.

The Post machine consists of a *tape* and a *carriage* (that is also called a *reading and recording head*). The tape is infinite and marked off into square sections (later referred to as *cells*) of equal size; to make it more graphic we will arrange the tape horizontally (Fig. 1).

The infiniteness of the tape contradicts the above-made assertion that the Post machine could have been built in principle. The thing is that we only call the tape *infinite* for the sake of simplicity of the presentation. The tape

---

* That is why recommendations on getting acquainted with like machines such as: "...in order to get a better notion of a really operating machine, those concerned should turn to technical literature" sound absurd (A. I. Popov, *Introduction to Mathematical Logic*, LGU Press, Leningrad, 1959, p. 91 (in Russian)).

** A device, that makes it possible to simulate the operation of the Post machine with short-length programs and nonbulky calculations, was made in Simferopol State University in 1970 (see V. N. Kasatkin, *Seven Problems on Cybernetics*, Kiev, 1975, p. 26 (in Russian)).

could equally have been assumed *growing unlimitedly* in both directions rather than infinite; for instance, we could believe the tape to link a new cell as soon as the carriage reaches the end of the tape and must move further (for the carriage motion, see below) or we could believe

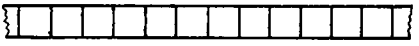

Fig. 1. The tape of the Post machine is squared into cells and grows infinitely to the left and to the right.

a new cell to be linked on the left and on the right per unit time. We think, however, more convenient to regard all the cells on the left and on the right to have been linked and thereby to assume the tape infinite in both directions ignoring the real state of things

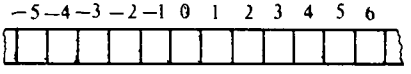

Fig. 2.

The order the cells of the tape are arranged in is similar to that of integers. It is natural, therefore, to use on the tape a whole-number system of coordinates, i.e. to give the cells integral numbers: —3, —2, —1, 0, 1, 2, 3, . . . (Fig. 2).

We will hold that the coordinate system is rigidly linked to the tape and this will enable us to denote a certain cell on the tape by its number or coordinate. (Sometimes, though, it is helpful to introduce, alongside the principal "constant" coordinate system, another, auxiliary, "additional", coordinate system shifted in respect to the initial one.)

Each cell of the tape either may contain nothing (such cell is called *blank*) or may contain a *label* (such cell is called *labelled*) (Fig. 3).

The information on which cells are blank and which are labelled characterizes the *condition of the tape*. In other words, the condition of the tape depends on how the labels are distributed over the cells.* As we will see later, the condition of the tape changes in the process of machine operation.

The carriage can move along the tape to the left or to the right. When it is at a standstill, it only faces precisely

Fig. 3. Each cell of the tape either contains nothing or contains a label.

one cell (Fig. 4*a*; this drawing and those that follow depict the carriage as a solid square); the carriage is said to *examine* this cell or to *keep it in view*.

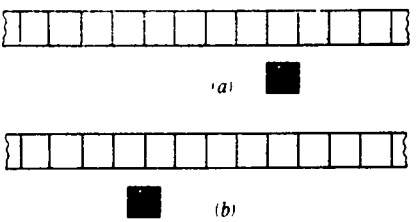

Fig. 4. When the carriage is at a standstill it examines one of the cells of the tape as shown in (*a*) rather than as it is shown in (*b*). The situation depicted in (*b*) can arise only during the carriage movement.

The information on which cells are blank and which are labelled and which cell is being examined by the carriage characterizes the *internal state of the Post machine*. So the state of the machine is composed of the condition of the tape and the number of the cell being examined by the carriage. The carriage can move one cell to the left or to the right in a unit time (that will be referred to as a *step*). Besides, the carriage can put (print) or cancel

* The condition of the tape is in an orthodox mathematical language a function that correlates a number (a cell number) either with the label or, say, with the word "blank".

(erase) the label in the cell being examined and recognize whether or not the cell being examined is labelled. What governs the carriage behaviour and what it means "to recognize" as applied to the carriage will be cleared up in Sec. 1.3.

Sec. 1.2. The Program for the Post Machine

The Post machine operation consists in that the carriage moves along the tape and either prints or erases labels. The operation proceeds by a certain instruction, that is called a program. Various programs can be prepared for the Post machine. Let us see how a program is devised.

Each program for the Post machine consists of instructions. We will call an expression of one of the following six forms an *instruction* to the Post machine (i, j, j_1, j_2 denote everywhere natural numbers 1, 2, 3, 4, 5, ...):

1st form. Move-to-the-right instructions

$$i. \Rightarrow j.$$

2nd form. Move-to-the-left instructions

$$i. \Leftarrow j.$$

3rd form. Instructions for printing the label

$$i. \vee j.$$

4th form. Instructions for erasing the label

$$i. \xi j.$$

5th form. Transfer-of-control instructions

$$i. \ ? \begin{matrix} \nearrow j \\ \searrow j_2 \end{matrix} 1.$$

6th form. Halt instructions

$$i. \text{ stop.}$$

For example,

$$137. \Rightarrow 1$$

is a move-to-the-right instruction,

$$25. \ ? \begin{matrix} \nearrow 32 \\ \searrow 25 \end{matrix}$$

is a transfer-of-control instruction, and

6386. stop

is a halt instruction.

The number i before an instruction is called the *number of the instruction*. So, the instructions just mentioned have numbers 137, 25 and 6386, respectively. The number j at the end of an instruction (j_1 and j_2 for transfer-of-control instructions) will be called a *jump* (j_1 being the upper jump and j_2 the lower jump for the transfer-of-control instruction). Halt instructions have no jump. Numbers 1, 32, 25 denote the jumps in transfer-of-control instructions given above as examples, 32 being the upper jump and 25, the lower jump.

We will call a finite nonblank (i.e. containing at least one instruction) list of the Post machine instructions possessing the following two features the *program for the Post machine*:

(1) The instruction with number 1 occupies the first place, the one with number 2, the second place (if at all), and so on; the instruction with number k, generally, occupies the kth place.

(2) A jump of any instruction in the list coincides with the number of a certain instruction (the same or any other) in the list (more precisely: to every jump of every instruction in the list there corresponds an instruction whose number equals the jump under consideration).

The following list, for example, will serve as a program for the Post machine:

1. stop 3. ξ 3

2. ? 4 / 1 4. stop

whereas these two lists cannot be programs for the Post machine:

2. ? 4 / 1 3. ξ 3

1. stop 4. stop

(the first restriction is not satisfied),

1. stop	3. ξ 3
2. ? 4 / 5	4. stop

(the second restriction is not satisfied).

We will write the programs for the Post machine in columns to make them more graphic. The number of instructions in the program is called the length of the program.

Exercise. Write all the programs of length 1 for the Post machine. How many programs are there of length 2, length 3, length n?

Sec. 1.3. The Operation of the Post Machine

To start the Post machine it is necessary: (a) to preset a program, (b) to set the machine in a certain internal state, i.e. to arrange somehow labels in the tape cells (for example, all the cells can be left blank) and to place a carriage facing one of the cells. As a rule, we will suppose that in the initial internal state (i.e. in the state given in the beginning) the carriage of the machine is always placed facing the cell with the zero number (coordinate). With such a stipulation the initial internal state of the machine is completely determined by the condition of the tape.

As already stated, the program is an instruction which is the basis for the machine's operation. The machine operates on the basis of the preset program (and at the given initial internal state) as follows: it is driven into the initial internal state and starts executing the first instruction of the program (what it means "to execute an instruction" will be cleared up below). The instruction is executed in one step; after that the machine starts executing the instruction whose number (call it α) equals the jump (one of the jumps if there are two) of the first instruction. This instruction is executed in one step too; after that the instruction is executed whose number equals the jump of the instruction with number α. Generally speaking, each instruction is executed in one step while

in going from the execution of one instruction to the execution of another, the following rule is observed: let at the kth step the instruction with number i be executed; then if this instruction has a single jump j, at the $(k + 1)$th step the instruction with number j is executed; if this instruction has two jumps, j_1 and j_2, at the $(k + 1)$th step one of the two instructions is executed, that is with number j_1 or with number j_2 (it will be pointed out below which); finally, if the instruction executed at the kth step has no jump at all, then at the $(k + 1)$th and all subsequent steps no instruction is executed: the machine stops. It remains to make it clear what it means to execute an instruction and which of the two jumps, if there are two, is taken as the number of the next instruction.

The execution of the move-to-the-right instruction consists in the carriage being moved one cell to the right. The execution of the move-to-the-left instruction consists in that the carriage is moved one cell to the left. The execution of the instruction of printing a label consists in that the carriage prints a label on the cell being examined. This instruction is executable only provided the cell being examined before executing the instruction is blank. If, however, the cell being examined has already been labelled, the instruction is regarded nonexecutable. The execution of the instruction of erazing the label consists in the carriage cancelling the label in the cell being examined. The execution of this instruction is possible only in the case when the cell being examined is labelled; but if there is no label in the cell being examined the instruction is considered nonexecutable. The execution of the transfer-of-control instruction with the upper jump j_1 and the lower jump j_2 in no way changes the internal state of the machine: none of the labels is cancelled or printed and the carriage is at rest (the machine does, so to say, a do-nothing step). It, however, the cell being examined before the execution of the instruction was blank, the next should be the instruction with number j_1; if this section was labelled, the instruction with number j_2 should be executed next (consequently, the role of the transfer-of-control instruction consists in that the carriage, in executing this instruction, sort of "recognizes" whether it examines the label; that precisely was meant in the last but one statement of Section 1.1). The execution of the

halt instruction does not in any way change the internal state of the machine either and leads to its halt.

Now if having preset a program and an internal initial state we set the machine in motion, one of the three operations will be accomplished:

(1) In the course of performing the program the machine comes to a nonexecutable instruction (e.g. printing a label in the nonblank cell or erasing a label in the blank cell). Then the execution of the program is stopped, the machine halts; the so-called *no-result halt* takes place.

(2) In the course of performing the program the machine comes to a halt instruction. In this case the program is regarded to be accomplished, the machine stops. The so-called *result halt* takes place.

(3) In the course of performing the program the machine does not come to the execution of either instruction dealt with in (1) and (2). The execution of the program never stops, the machine keeps going. The machine operation goes on forever.

Sec. 1.4. Examples of Performing Programs

Let us set an initial internal state shown, for example, in Fig. 5 and the following program:

1. $\vee 4$ 4. $\Rightarrow 5$
2. $\xi 3$
3. $\Leftarrow 2$ 5. $?\begin{matrix}\nearrow 4\\ \searrow 3\end{matrix}$.

Let us see how the machine will operate given such an initial state and a program.

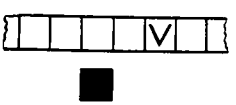

Fig. 5.

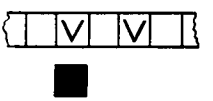

Fig. 6.

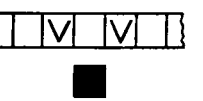

Fig. 7.

At the first step instruction No. 1 will be executed after which the internal state of the machine will turn as shown in Fig. 6. After instruction No. 1 is executed we

should pass to the execution of the instruction whose number coincides with the jump of instruction No. 1, i.e. instruction No. 4. It will be executed at the second step and the machine internal state will become as depicted in Fig. 7. Now instruction No. 5 is to be executed (for the jump of instruction No. 4 is equal to 5). This instruction will be executed at the third step; as a result of this step the internal state of the machine will not change and

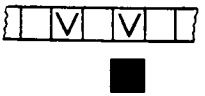

Fig. 8.

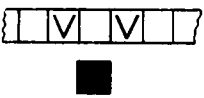

Fig. 9.

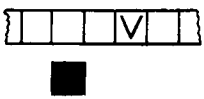

Fig. 10.

remains as it was in Fig. 7. Since the cell being examined is blank in this case, the next instruction to be executed is the one whose number is equal to the upper jump, i.e. No. 4. After instruction No. 4 is executed at the fourth step, the machine will reach the internal state shown in Fig. 8. Now, at the fifth step instruction No. 5 will be executed. This time the cell being examined is labelled, therefore the next instruction to be executed is that with the number equal to the lower jump, i.e. No 3. After execution, at the sixth step, of instruction No. 3 the machine comes to the internal state shown in Fig. 9 and starts, at the seventh step, executing instruction No. 2. But the latter will appear to be nonexecutable as it orders erazing the label in the blank cell. Consequently, a no-result halt will take place at the seventh step.

Various programs applied to the same initial internal state can lead to various outcomes, i.e. to a no-result halt, to a result halt, to a haltless operation of the machine. Let us set, for instance, the initial internal state shown in Fig. 10 and apply to that initial state the program:

1. $\Rightarrow 2$
2. $\Rightarrow 3$
3. $\bigvee 1$.

The machine will perform two steps and a no-result halt will occur at the third. Another program is applied to

the same initial internal state:

1. $\Rightarrow 2$
2. $\Rightarrow 3$
3. stop.

The machine will perform two steps and a result halt will occur at the third. Finally, yet another program is applied to the same initial internal state:

1. $\Rightarrow 1$.

The machine will operate perpetually. Let us apply the program:

1. $?\swarrow\begin{matrix}1\\1\end{matrix}$.

Again the machine will operate perpetually (in spite of the fact that neither the recording on the tape nor the carriage position will change).

The same program when applied to various initial internal states can in exactly the same manner produce various results. Let us regard, for instance, the following program:

1. $?\swarrow\begin{matrix}4\\1\end{matrix}$ 3. stop
2. ξ 3 4. $\Rightarrow 2$

and apply it to initial internal states A, B, C depicted in Fig. 11. For initial internal state A we have a result halt at the fourth step, for B, an endless operation of the machine, for C, a no-result halt at the third step. Applied to the same initial internal states, the program

1. $?\swarrow\begin{matrix}4\\3\end{matrix}$ 3. stop
2. $\vee 4$ 4. $\Rightarrow 2$

produces a no-result halt for A, result halt for B, and an endless operation for C.

Exercise 1. Can there be a program producing, at any initial internal state, a result halt? A no-result halt? An endless operation of the machine? What is the least number of instructions in these programs?

Exercise 2. A certain program is known to produce: (a) a result halt when applied to a definite initial internal

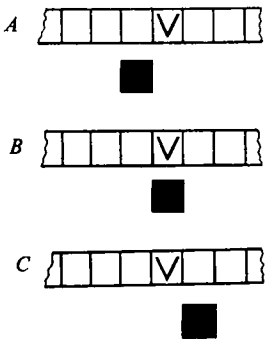

Fig. 11.

state; (b) a no-result halt when applied to a definite initial internal state; (c) an endless work of the machine when applied to a definite initial internal state. Prove that the number of instructions in this program is not less than 4. Write all such programs of length 4.

Sec. 1.5. Methodological Notes

This section is addressed to the teachers of primary school, to the supervisors of school groups for younger children and in general for all those who are going to instruct schoolchildren of earlier grades on the Post machine.

As already stated in the Preface, it is possible to train to work on the Post machine (drawn in chalk on the blackboard or with the aid of paper tape and buttons or clips used as labels) even seven-year-olds. In presentation to the schoolchildren it is, of course, necessary not only to miss some details but to introduce new stipulations. So it is appropriate

(1) to say nothing of the coordinate system on the tape (that is needed only to specify the term "state") and not to introduce the concept of the internal state of the machine as a whole and of the condition of the tape;

(2) to employ as the numbers of programs some symbolic representations (for instance, representation of geometric figures) rather than numerals;

(3) to assume the tape finite rather than infinite and to agree that the machine stops when the carriage comes to the end of the tape;

(4) not to speak at all of the no-result halt; if it nevertheless occurs in performing some program, then say that "the machine gets out of order";

(5) to introduce the instructions not all at a time but step by step, each new instruction being followed by visual exercises.

The exercises should, of course, be chosen to match the school age. It is helpful to offer the following

Exercise. The machine starts with the blank initial internal state (that is the state in which all the cells are blank) and operates by the program:

1. $\vee 2$
2. $\Rightarrow 3$
3. $\Rightarrow 1$.

The question is: what will happen to the tape? The answer is graphic enough; it is shown in Fig. 12.

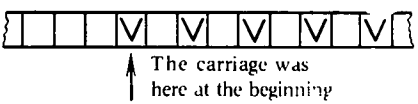

Fig. 12.

We believe, though, that a schoolboy or a schoolgirl must (and can) apply any program to any initial internal state, so that any exercises in performing programs taking not very much time (and even exercises in writing some programs, for example, a program leading to the condition of the tape depicted in Fig. 12) quite do for the purpose.

Certainly, in presenting the material to junior schoolchildren it is no good mentioning the words "algorithm", "Post machine" (you had better exclude the word "machine" altogether and say, for example, "we pass to the next square on the right" instead of "the carriage moves one cell to the right" and so on). And what is more, any comment on what the given device is used for should be delayed until it is clearly understood and the execution of programs becomes free and easy to grasp. Here a quotation from *Prelude to Mathematics* by W. W. Sawyer* is relevant: "I will tell you first what I do; I will tell you the reason afterward".

The capability of perceiving any system of concepts or any reasoning, in general, before (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 makes, perhaps, for its memorizing but 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 schoolchildren.

Concluding the presentation of a certain device, the presentation with a sufficient number of examples and exercises, means the completion of the important stage; following it, at the next stage, we can pass to the application of the device described. For the Post machine such an application consists in computations that can be done with it. To demarcate the two stages we confine ourselves to a brief outline of the Post machine in this chapter.

---

* W. W. Sawyer, *Prelude to Mathematics*, Penguin Books, Bristol, 1955, p. 125.

** Multiple digit numbers are not attached a conventional quantitative meaning but are taken as chains of figures; the sum being determined by the algorithm of addition in columns (see V. A. Uspensky "On Teaching Mathematics in Primary School" *Mathematics in School*, No. 2, 1966 (in Russian); what we mean is that only later is the formal summation described used to get the sum of two quantities of apples or notebooks.

After the outline it is appropriate to tell the schoolchildren that the Post machine can be employed to obtain the results of arithmetic operations and to offer examples of programs (and later problems on programming too) leading to the results of operations with the numbers recorded on the tape. Addition of unity to a number is one of the simplest operations. To this end we must, of course, agree in advance how to record numbers on the tape of the Post machine and make some more specifications. But this is the subject of the next chapter specially devoted to the addition of unity. We can suggest, as a helpful example, that the reader should put an exact sense into the following formulation: "Write a program for the Post machine performing addition of a unity" and perhaps it will turn out that such a sense is not the only one?

2. ADDITION OF UNITY ON THE POST MACHINE

Various calculations can be performed on the Post machine. This chapter presents to the reader one of simplest calculations, the addition of unity to an arbitrary number; this can be accomplished in a few ways depending on one formulation of initial conditions or another.

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 when operating on real high-speed computers too.

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 the given goal*; and this is called *programming*. This chapter deals with the problem (a series of problems, to be more exact) on preparing programs for the Post machine that result in an increase of an initial number by a unity.

Apart from the common principal problem, there are many other common features typical of programming for the Post machine and for real machines. Here are a few points noticeable even in analysing the problem on addition of unity:

1. We can prepare *different* programs that lead to the (*same*) specified goal, i.e. that carry on the specific processing of data; the reader will see (in Sec. 2.2, Exercise) that the Post machine can even operate by an infinitely great number of programs executing the addition of unity (even by the simplest mode of the addition).

2. If a class of initial data applied to which a program leads to a needed result extends, the programming in this case becomes more difficult and the program itself, which is a solution of the problem, more complex; it will be seen later how the program of adding unity will become more complex with extending a class of admissible initial conditions.

3. When preparing programs for the solution of more general or more complex problems it is often expedient to use programs, built earlier for the solution of more particular and simpler problems, as "prefabricated" building units; the reader will see later in Sec. 2.4 how programs for the solution of more particular problems can be used when building a program for a more general problem on addition of unity.

4. Preparing a program, we have to bear in mind not only what numbers it should be applied to but how these numbers are arranged in the "store" or "memory" of the machine; it will be seen later that the variants under study here of the problem on addition of unity will only differ in the position of the initial number in respect to the carriage.

5. To strive for making programs as short as possible is natural and this can in a number of cases be a decisive matter for practical computers and problems; the reader will see later that special attention will be paid to minimizing the program length.

Sec. 2.1. Recording Numbers on the Post Machine and the Statement of the Problem on Addition of Unity

Various operations with numbers can be performed on the Post machine. But we must first of all agree how to record numbers in the Post machine. We will always speak of integral nonnegative numbers 0, 1, 2, 3, 4, ...

Let us consider the finite sequence of the labelled cells following in succession one another, which sequence is confined between two blank cells. Such sequence of labelled cells will be referred to as an *array*, and the number of cells in it, an *array length*. For example, Fig. 13 shows an array of length 3, and Fig. 14, three arrays of length 5, length 1 and length 2 each.

Now let us agree to record number n on the tape by means of an array of length $n + 1$ and the array itself will be referred to as a *machine record* of number n. Fig-

ures 13 and 14, consequently, show machine records of numbers 2, 4, 0, and 1.

Let us set a goal to perform the addition of unity on the Post machine. We will understand our task as follows: a program is to be built that being applied to the tape, containing the record of number n, would lead to a result halt; after the halt number $n + 1$ should be recorded (we mean *one* program applicable for *any* n).

The problem was not formulated quite specifically in the preceding statement, as we know nothing of: (a) an

Fig. 13.

Fig. 14.

initial internal state (i.e. the state in the beginning) of the machine; (b) an end internal state (i.e. the state after the result halt) of the machine; (c) which cell the number is recorded in; (d) what else is recorded on the tape; (e) which cell the carriage should examine. We will assume that in the beginning and in the end of the program operation the tape only contains the records of the respective numbers (n in the beginning and $n + 1$ in the end) arranged arbitrarily; the rest of the tape is blank. To facilitate the task we will impose no restrictions on the end state; so any program leading to $n + 1$ record on the tape will satisfy us wherever the record was and wherever the carriage was. At the same time we will make broader assumptions on the relative positions of the carriage and the machine record of the number in the initial internal state of the machine. Consequently, we will deal with a series of problems rather than one problem. We strongly recommend that the reader should try to solve the problem himself before reading its solution.

Sec. 2.2. Addition of Unity in the Simplest Case

We will begin with the strongest restriction imposed on the relative positions of the machine record of the number and the carriage at the beginning, and thereby, with the simplest problem. We will call it problem 1.

Problem 1 (lengthy statement). Write the Post machine program possessing the following property. Whatever number n is, if the initial internal state of the Post machine is such that the tape contains the machine record of number n (and the rest of the tape is blank) and the

Fig. 15. The states of class A_n differ from one another only in the "shift" relative to the coordinate system.

carriage faces the leftmost cell of the record, executing the program must yield a result halt; after that number $n + 1$ must be recorded on the tape (in an arbitrary space, the rest of the tape must be blank), the carriage being positioned against any cell.

Denote all such states of the Post machine by A_n; in each of these states precisely $n + 1$ cells are labelled and the carriage faces the leftmost of the labelled cells. Figure 15 shows a few states of class A_2; they only differ from one another in the position of an array and the position of the carriage, "rigidly linked to it", relative to the coordinate origin.

Denote by E_n the totality of all the Post machine states in each of which precisely $n + 1$ cells are labelled and

the carriage is positioned against any cell. E_n includes all the states denoted by A_n and many others. Figure 16 shows a few states of class E_2.

Now we can state Problem 1 shorter.

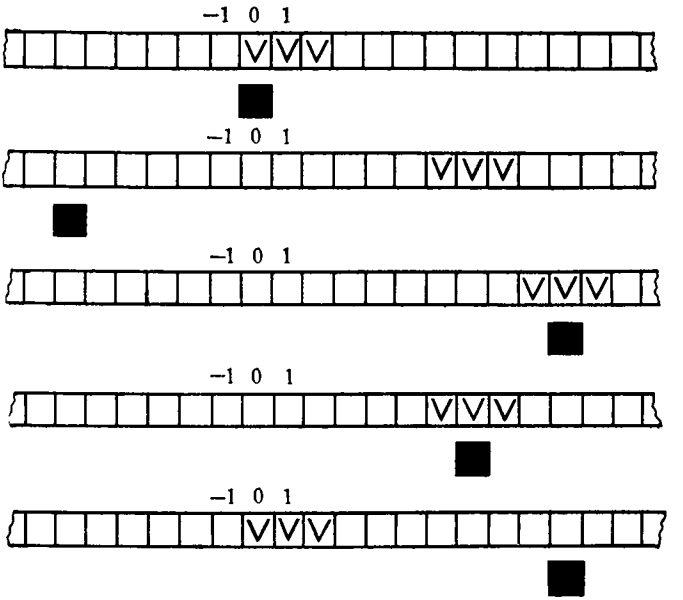

Fig. 16.

Problem 1 (shorter statement). Write a program for the Post machine that, for any n, being applied to an arbitrary state from class A_n produces a result halt in a state from class E_{n+1}.

Before approaching problem 1, we note that it does not specify what the application of the sought program to the states belonging to no one of classes A_n ($n = 0, 1, 2, \ldots$) should lead to, consequently, what will take place in such states chosen as initial does not matter to us: any program converting states from A_n into states of E_{n+1} will come right whatever it did to the rest of the states.

Such a program, for example, will be a solution of problem 1:

Program I_1

1. $\Leftarrow 2$
2. $\vee 3$
3. stop.

We said "for example" because this solution is not unique; other programs satisfying the conditions of the problem are possible to be built. Such a program, for example, will also be a solution to problem 1:

1. $\Rightarrow 2$
2. $?\begin{cases}3\\3\end{cases}$
3. $\Leftarrow 4$
4. $\Leftarrow 5$
5. $\vee 6$
6. stop.

The foregoing program I_1 is, however, the shortest (one of the shortest, to be more precise) of the programs satisfying the conditions of problem 1. Indeed, we can prove (and recommend the reader doing so) that no program of length 1 or 2 can be a solution of problem 1; at the same time there is exactly one more program of length 3 that is also a solution of our problem; here is the program:

Program I_2

1. $\Leftarrow 3$
2. stop
3. $\vee 2$.

Exercise. Prove that there is an infinite multitude of programs that are solutions of problem 1.

Note. With fixed n the states from A_n only differ from one another in the shift relative to the origin of coordinates (Fig. 15). Whatever program we apply to these initial states, the results produced will, obviously, differ from one another in the same shift. It is therefore sufficient to choose one state a_n from each class A_n and build a program converting each a_n into a certain state from E_{n+1}. Every program like that will in itself be a solution of problem 1. Note that state a_n can be chosen from A_n quite at will.

Problem 1′, differing from problem 1 only in that the carriage first examines the rightmost cell of the array, can be solved in a like manner. Just denote by A'_n the class

of all the states from E_n in which the carriage examines the rightmost labelled cell. Then the shorter statement of problem 1′ will be as follows:

Problem 1′ (shorter statement). Write the program for the Post machine that, for any n, being applied to an arbitrary state from class A'_n produces a result halt in a certain state from class E_{n+1}.

The following two programs will be the only shortest programs satisfying the conditions of problem 1′:

Program I'_1

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

Program I'_2

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

Sec. 2.3. Addition of Unity in More Complicated Cases

Now we will not stipulate that the carriage should by all means examine one of the extreme cells of the array as we did in problems 1 and 1′. We will only require that at the initial state the carriage should examine one of the cells of the array.

Problem 2 (lengthy statement). The program for the Post machine is to be built that possesses the following property. Whatever number n is, if the initial internal state of the Post machine is such that number n is recorded onto the tape (and the rest of the tape is blank) and the carriage faces one of the number record cells, the program performed should lead to a result halt; after that number $n + 1$ should be recorded onto the tape (in any space of it; the rest of the tape should be blank); the carriage can be positioned anywhere in this case.

Denote by B_n the totality of the Post machine states; precisely $n + 1$ cells are labelled in every state and the carriage faces one of the labelled cells. Class B_n is, ev-

identically, a part of class E_n and contains in turn class A_n as a part. Figure 17 depicts a few states from class B_n and Figure 18, the general view of the state from class B_n. Then we get the following shorter statement of problem 2:

Problem 2 (shorter statement). Build the program for the Post machine that, for any n, being applied to an arbitrary state from class B_n leads to a result halt in a certain state from class E_{n+1}.

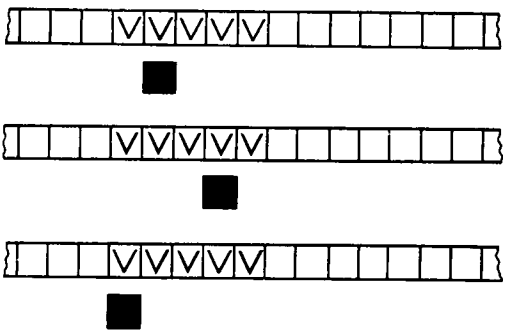

Fig. 17.

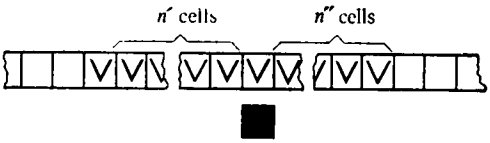

Fig. 18. The general view of the state from class B_n. Here $n' \geqslant 0$, $n'' > 0$, $n' + n'' = n$.

Each solution of problem 2 will, obviously, be at the same time a solution of problem 1 and problem 1′ too. There are, however, solutions of problem 1 that are not solutions of problem 2. Such are programs I_1 and I_2. For problem 2, just as for problem 1, there is an infinite multitude of programs that are its solution. We will, as before, take interest in the shortest programs. We can prove (and recommend the reader doing so) that no program of lengths 1, 2 or 3 can be a solution to problem 2. At the same time, solutions of length 4 of problem 2 are possible. Here is one of the solutions:

Program II_1

1. $\Leftarrow 2$ 3. $\vee 4$
2. $?\begin{matrix}3\\1\end{matrix}$ 4. stop.

Exercise 1. Arrive at another two solutions of problem 2 of length 4 that contain a move-to-the-left instruction. See to it that each of the solutions found contains an instruction of label printing, a transfer-of-control instruction and a halt instruction.

Exercise 2. Prove that, apart from program II_1, there are precisely another eleven (II_2, II_3, ..., II_{12}) solutions of length 4 of problem 2 that contain a move-to-the-left instruction. Write down these solutions.

Exercise 3. Verify that programs II_1', II_2', ..., II_{12}' resulting from programs II_1, II_2, ..., II_{12} by substituting $\Rightarrow$ for $\Leftarrow$ are also solutions of problem 2. Prove that programs II_1, II_2, ..., II_{12}, II_1', II_2', ..., II_{12}' exhaust all the solutions of length 4 of problem 2.

Consider now the initial states in which the carriage surveys a certain blank cell rather than a certain cell of the initial array. We will assume here that whether the carriage positioned to the left or to the right of the initial array is known in advance. The words "is known in advance" imply that two programs one of which is for the case when the carriage is at first to the left of the array and the other is for the case when the carriage is at first to the right of the array and not a single program operating in all the cases are to be built. So we have two problems rather than one here. In order to obtain shorter statements of these problems at once we will introduce the following notation:

C_n is a totality of all the states from class E_n in which the carriage is to the left of the array;

C_n' is a totality of all the states from class E_n in which the carriage is to the right of the array.

Figure 19 presents the general view of a state from class C_n, and Figure 20, the general view of a state from class C_n'.

Problem 3 (shorter statement). Build the program for the Post machine that, for any n, being applied to an

arbitrary state from class C_n leads to a result halt in a certain state from class E_{n+1}.

Problem 3′ (shorter statement). Build the program for the Post machine that, for any n, being applied to an arbitrary state from class C'_n leads to a result halt in a certain state from class E_{n+1}.

Each program that is a solution of problem 3 can evidently be turned into a solution of problem 3′ if the

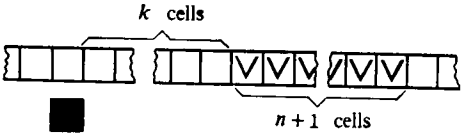

Fig. 19. The general view of the state from class C_n. Here $k \geqslant 0$.

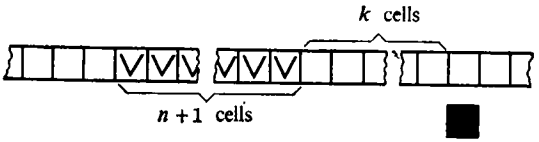

Fig. 20. The general view of the state from class C'_n. Here $k \geqslant 0$.

symbol $\Leftarrow$ is throughout substituted for the symbol $\Rightarrow$ and the symbol $\Rightarrow$, for the symbol $\Leftarrow$. The solution of problem 3′ is turned into the solution of problem 3 likewise. It is therefore sufficient to solve only one of these problems. Here is one of the solutions of problem 3:

Program III

1. $\Rightarrow 2$
2. $?\begin{matrix}/\,1\\ \backslash\,3\end{matrix}$
3. $\Leftarrow 4$
4. $\vee 5$
5. stop.

Try to prove that problem 3 does not permit shorter solutions. How many solutions of length 5 of this problem are there?

Sec. 2.4. Addition of Unity in Yet More Complicated Case

We shall consider in this section a union of all the classes B_0, C_0, B_1, C_1, B_2, C_2, B_3, C_3, ... from the preceding section as a class of initial states. This class will, consequently, consist of all such, and only such, states of the machine in each of which the carriage either faces any cell of the initial array or is positioned to the left of the array.

Denote by D_n a totality of all the states of class E_n in which the cell examined by the carriage is either labelled or positioned to the left of all the labelled cells. For every n, class D_n is, obviously, a union of classes B_n and C_n.

Problem 4. Build the program for the Post machine that, for any n, being applied to an arbitrary state from class D_n produces a result halt in a certain state of class E_{n+1}.

This problem can be reduced to problems 2 and 3, i.e. it can be demonstrated how to obtain a solution of problem 4 from arbitrary solutions of problems 2 and 3. Let us show how to do it. Let Ξ be an arbitrary list of instructions for the Post machine and k, an arbitrary integer. We will use $\Xi\ [+k]$ for the list, derived from Ξ by adding number k to all the numbers and to all the jumps of the instructions from Ξ. $I_1\ [+7]$ is, for example, such a list:

8. $\Leftarrow 9$
9. $\vee 10$
10. stop.

Now let II be an arbitrary program that is a solution of problem 2, and III, an arbitrary program that is a solution of problem 3. Let l be the length of program II. Now let us make up the following list of instructions:

1. $?\begin{cases} l+2 & \text{II} \quad [+1] \\ 2 & \text{III} \quad [+l+1]. \end{cases}$

It can be easily verified that this list of instructions is a program for the Post machine, that satisfies the conditions of problem 4. Indeed, every state from class D_n either

belongs to B_n or to C_n. List II [+1] works in the first case, and list III [| l | 1], in the second case.

The foregoing procedure is, however, not bound to lead (and, as we will see, does not, in fact, lead) to the shortest solutions of problem 4 even if to proceed from the shortest solutions of problems 2 and 3. Let us see what will come of it if the procedure is applied to programs II_1 and III. We will obtain such a solution of problem 4:

Program IV[10]

1.	?/6 \2	6.	⇒7
2.	⇐3	7.	?/6 \8
3.	?/4 \2	8.	⇐9
4.	∨5	9.	∨10
5.	stop	10.	stop.

It is clear that this program can be shortened, without affecting its operation, by merging of two half instructions into one or, as we will say, by absorbing one of those instructions by the other. To absorb instruction No. 10 by instruction No. 5 is easier. To this end jump 10 should be replaced by jump 5 in all the instructions where it is found (in our case in instruction No. 9 only) and then instruction No. 10 should be deleted from the list*.

We will obtain program IV[9] in which instruction No. 9 can be absorbed by instruction No. 4. To do this it is sufficient to replace jump 9 by jump 4 in instruction No. 8, and then to delete instruction No. 9. After the two absorptions being accomplished we will obtain a solution of problem 4 in the form of program IV[8].

---

* If we were going to absorb instruction No. 5 by instruction No. 10, we would have jump 5 replaced by jump 10 in all the instructions where it is found (in the given case in instruction No. 4), then have jump 5 removed from the list; after that we would have made all the numbers of the instructions, following the removed one, and all the jumps, coinciding with these numbers, less by one.

Program IV[8]

1. $?\begin{smallmatrix}/\,6\\\backslash\,2\end{smallmatrix}$ 5. stop

2. $\Leftarrow 3$ 6. $\Rightarrow 7$

3. $?\begin{smallmatrix}/\,4\\\backslash\,2\end{smallmatrix}$ 7. $?\begin{smallmatrix}/\,6\\\backslash\,8\end{smallmatrix}$

4. $\vee 5$ 8. $\Leftarrow 4$.

Note. In a general case the absorption (in the given program) of instruction No. α by instruction No. β consists in a sequential performance of three operations: (1) jump α is throughout replaced by jump β; (2) instruction No. α is deleted; (3) numbers $\alpha + 1$, $\alpha + 2$, $\alpha + 3$, ... (that can be both numbers of instructions and jumps), in all instructions of the list obtained, are replaced respectively by α, $\alpha + 1$, $\alpha + 2$, ... If instructions No. α and No. β coincided in everything but their numbers, the program obtained after the absorption of instruction α by instruction β and the original program will "operate in quite a similar manner". The words taken in quotes are cleared up as follows. Let us take two samples of the Post machine, each set in a certain—the same for both machines—initial state and make the first machine work by a certain program A and the second machine, by a certain program B. Assuming the machines working synchronously, we will regard in parallel the states arising in the first and the second machine at the same time. At the initial moment the states are identical by the condition of the problem. They may happen to be identical at all the subsequent moments too, each machine coming to halt, if at all, at the same time and this halt being of the same quality (i.e. it is either a result one or a no-result one in both machines). If the process described is characteristic of any initial state, common for both machines, we will say that programs A and B work quite in a similar manner. Speaking more roughly, A and B work quite in a similar way if, for any m, on performing m steps of program A the same state arises as on performing m steps of program B on condition it was true for $m = 0$, i.e. in the very beginning of operation. Here are examples

of programs operating in a like manner: (a) I_1 and I_2; (b) I_1' and I_2'; (c) IV^{10} and IV^8.

Now let us see whether we can make program IV^8 shorter too. There are no two instructions that are distinguished by only their numbers; that is why the manner of absorption here can, generally speaking, lead to a program that will not operate in the same way as the initial one. Yet program IV^8 can be shortened by absorption.

Let us compare instructions No. 8 and No. 2 with this purpose. They cannot merge into a single instruction as they have different jumps. For all that, instruction No. 2 appears to be able to take upon itself, in a certain sense, the functions of instruction No. 8 (and, therefore, to absorb it).

Suppose we are to execute, at a certain stage of the program operation, instruction No. 8. Assume that the cell immediately to the left of the one examined by the carriage at the given moment is blank. Then instruction No. 8 will move the carriage to position it against the blank cell whereupon instruction No. 4 will print a label onto it. If instead of instruction No. 8 we execute instruction No. 2, we will have the following: instruction No. 2 will move the carriage and position it against the same blank cell as before, instruction No. 3 will make the machine do a "do-nothing" step whereupon instruction No. 4 will operate once more. Let us see what will happen if the cell immediately to the left of the examined one is not blank but labelled. If so, a no-result halt will take place in the former case (i.e. after instruction No. 8 has been executed) and there will be no no-result halt in the latter case (i.e. after instruction No. 2 has been executed).

The above reasoning demonstrates that if executing consecutively instructions No. 8 and No. 4 does not cause a no-result halt, it can be replaced by executing instructions Nos. 2, 3, 4 (while a reverse replacement can, in general, essentially change the program operation: executing instructions Nos. 2, 3, 4 never causes a no-result halt but replacing these instructions by executing instructions Nos. 8, 4 may lead to a no-result halt). If, therefore, we confine ourselves to the initial states for which, on performing program IV^8, a no-result halt is impossible (and such states are known to be all the states of D_n

where $n = 0, 1, 2, \ldots$)*, program IV^7 resulting from absorbing instruction No. 8 by instruction No. 2 will also be a solution of problem 4. Here is the program:

Program IV^7

1. ?/6 \2
2. ⇐3
3. ?/4 \2
4. ∨5
5. stop
6. ⇒7
7. ?/6 \2.

Exercise. Will programs IV^8 and IV^7 operate in quite a similar manner?

Absorbing, instruction No. 7 by instruction No. 1 in program IV^7, we will obtain the following program, IV^6:

Program IV^6

1. ?/6 \2
2. ⇐3
3. ?/4 \2
4. ∨5
5. stop
6. ⇒1.

Since programs IV^7 and IV^6 operate in quite a like manner, program IV^6, as well as IV^7, will be a solution of problem 4. Try to prove that problem 4 has no shorter solutions.

The problem on adding unity for the initial states with the examined cell being either labelled or being to the right of the labelled array is solved quite similarly. The solution of the respective problem—call it problem 4′—can be obtained replacing all the symbols ⇒ by ⇐ and the symbols ⇐ by ⇒ in an arbitrary solution of problem 4.

---

* For program IV^8 when applied to these states causes a result halt.

If we let D_n' be the totality of all the states from E_n in which the carriage either faces the array or is to the right of it, we can notice that, for any n, class E_n is a union of all classes D_n and D_n' and class B_n, an intersection of these classes. We can represent it in symbols:

$$D_n \cup D_n' \quad E_n, \quad D_n \cap D_n' = B_n.$$

Sec. 2.5. Addition of Unity in the Most General Case

We will now impose no restriction on the relative positions of the carriage and machine record of the number in the initial state. A program is, therefore, to be built that would perform addition of unity provided the recording of the initial number is found in an arbitrary cell of the tape and the carriage is also positioned arbitrarily. The corresponding requirement is stated in problem 5.

Problem 5. Write the program for the Post machine that, for any n, being applied to an arbitrary state of class E_n leads to a result halt in a certain state of class E_{n+1}.

Each program that is a solution of problem 5 will, obviously at the same time, be a solution of each foregoing problem. Neither of the foregoing solutions of problems 1, 1′, 2, 3, 3′, 4, 4′ is, however, a solution of problem 5 (verify this).

The initial states for problem 5 are given in Figs. 18, 19 and 20: for any state of E_n either belongs to B_n, or C_n, or C_n'. Now we must build the program leading, for every type of initial states depicted in Figs. 18, 19, 20, to the goal. Try to arrive at such a program by yourself and you will see it is not so easy. And, perhaps, there is no such program at all? Then try to prove it. The answer to the question whether problem 5 has a solution will be given in the next chapter.

3. ANALYSIS AND SYNTHESIS OF PROGRAMS FOR THE POST MACHINE

In the previous chapters we have already dealt both with the program synthesis problem (consisting in building programs performing the operations specified) and with the analysis problem (consisting in describing the conversions performed by programs). This chapter will handle problems solved for more complicated and more interesting situations: the analysis of programs will be discussed in connection with the problem left unsolved in the preceding chapter and the synthesis, in the problem on addition of numbers.

Sec. 3.1. Diagrams and Block Diagrams

The programs we have dealt with in the foregoing two chapters devoted to the Post machine were simple.

We could, therefore, without much effort *analyse* the specified program, i.e. to see how it works, and *synthesize* the needed program, i.e. build a program with the specific properties.

But let us suppose that a more complicated program is given, such as:

Program V

1.	$?\begin{smallmatrix}/\,2\\ \backslash\,3\end{smallmatrix}$	13.	$\Leftarrow 14$
2.	$\Leftarrow 4$	14.	$?\begin{smallmatrix}/\,5\\ \backslash\,13\end{smallmatrix}$
3.	$\Rightarrow 4$	15.	$\Rightarrow 16$
4.	$?\begin{smallmatrix}/\,5\\ \backslash\,3\end{smallmatrix}$	16.	$\Rightarrow 17$
5.	$\vee 6$	17.	$?\begin{smallmatrix}/\,23\\ \backslash\,18\end{smallmatrix}$

6. $\Leftarrow 7$

7. $?\swarrow^{8}_{15}$

8. $\Rightarrow 9$

9. $?\swarrow^{10}_{8}$

10. $\vee 11$

11. $\Rightarrow 12$

12. $?\swarrow^{13}_{19}$

18. $\xi\ 16$

19. $\Leftarrow 20$

20. $\Leftarrow 21$

21. $?\swarrow^{23}_{22}$

22. $\xi\ 20$

23. stop.

How shall we analyse this program?

We will do like this: we will partition our program (the portions will be called *blocks*) and try to grasp how each separate block operates and how separate blocks interact.

In order to come upon a handy partition of program V into blocks, let us draw a so-called *diagram* of this program.

In this diagram instructions are represented by circles and passages from one instruction to other, by arrows. In order to draw a diagram of program V we shall draw 23 circles and mark them by numbers from 1 to 23. Let us draw an arrow from circle No. i to circle No. j if and only if the ith instruction (i.e. the instruction with number i) has a jump equal to j. If the ith instruction is, in particular, a halt instruction, then no arrow will branch out from the ith circle, but if the ith instruction is a transfer-of-control instruction, then two arrows will branch out from the ith circle: one to the circle corresponding to the upper jump of the instruction and the other, to the circle corresponding to the lower jump. The entire program will then be represented by a drawing shown in Fig. 21. We will call this drawing a *diagram* of program V, and the circles composing it, *units* of the diagram.

The above method can be used to draw a diagram for any program. The succession of the instructions executed by the given program is graphically illustrated following the direction of arrows in the diagram of this program.

Now partition program V into groups of instructions, i.e. blocks Thereby the diagram will be partitioned into blocks (groups of units). Partitioning into blocks is more vividly shown as partitioning of a diagram.

For our purpose of analysing program V, it is expedient to partition the diagram in Fig. 21, and thereby the program itself, into the following eight blocks:

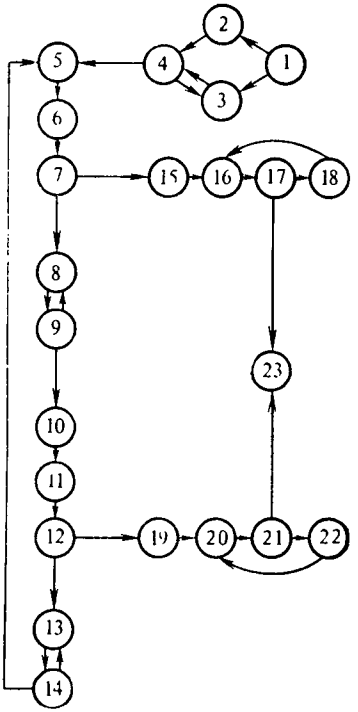

Fig. 21.

1st block—call it "start block"—consists of units (instructions) Nos. 1, 2, 3, 4;

2nd block—call it "check-to-the-left block"—consists of units (instructions) Nos. 5, 6, 7;

3rd block—call it "move-to-the-right block"—consists of units (instructions) Nos. 8, 9;

4th block—call it "check-to-the-right block"—consists of units (instructions) Nos. 10, 11, 12;

5th block—call it "move-to-the-left block"—consists of units (instructions) Nos. 13, 14;

6th block—call it "block of erasing to the right"—consists of units (instructions) Nos. 15, 16, 17, 18;

7th block—call it "block of erasing to the left"—consists of units (instructions) Nos. 19, 20, 21, 22;

8th block—call it "halt block"—consists of unit (instruction) No. 23.

For each partitioning of the given program into blocks we can draw a so-called *block diagram* of this program. With this purpose each block must be represented as a rectangle and we must draw, from a rectangle representing block α to a rectangle representing block β, as many arrows as there branch out from the units of block α to the units of block β. The block diagram of program V for the chosen partitioning into blocks is demonstrated in Fig. 22 (you will find the number and the designation of the respective block inside each rectangle).

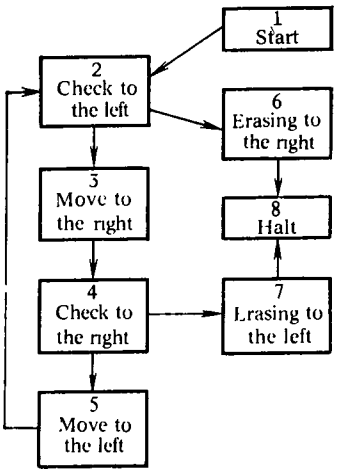

Fig. 22.

The block diagram of a certain program being before our eyes, we can visually imagine its operation as successive execution of instructions now of one block, now of another. The block containing instruction No. 1 will be referred to as a *start* one, and the block containing the halt instruction, a *halt* one; we will, for simplicity, assume the halt block to contain nothing more. Instead of saying "executing instruction of the given block" we shall agree, for brevity, to say merely "executing the given block". The start block is sure to be executed first.

While executing a nontermination block, one of the three cases can take place: (1) a no-result halt will occur when executing the block; (2) executing the block will never terminate, i.e. every time an instruction is executed another instruction of the same block will have to be executed; (3) executing the block will terminate, i.e. after executing a certain instruction we will go over to executing a certain instruction of another block. In short, getting into the nontermination block, we will either "get stuck" forever in it or, by one of the arrows branching out from it, we will escape into another block.

Sometimes we at once, by the diagram form, can single

out certain blocks in which it is known that there are certain ways of getting out (if the case of a no-result halt that is possible to occur at any moment is excluded). For the block diagram in Fig. 22 such blocks are, as follows from the diagram in Fig. 21, the 2nd and the 4th. Executing program V will, therefore, proceed as follows: block 1 is first executed; its execution either never comes to an end or if it does block 2 is executed. The execution of block 2 is sure to come to an end; after that block 3 or block 6 is executed; after block 3 (in case of its termination), block 4; after block 4, either block 5 or 7; after the 5th (in case of its termination), the 2nd; after the 2nd again either the 3rd or the 6th; and so on. Executing the 6th block (as well as the 7th one) may or may not come to an end; if it does, the 8th block is executed, and the machine comes to the halt.

Now we can, at last, solve an analysis problem for our program. We will limit ourselves to the case in which the initial state belongs (for certain n) to class E_n. In the next section it will be demonstrated, with the aid of the block diagram, that the program when applied to the initial state leads to a result halt in a state of class E_{n+1}.

A general problem on addition of unity (posed in Sec. 2.5) is thereby solved:

Write the program for the Post machine that, for any n, if applied to an arbitrary state from E_n (taken as an initial one) causes a result halt in a certain state of class E_{n+1}.

Program V of length 23 appears to be precisely a solution of this problem. The author has not succeeded in buiding a shorter program that could be a solution of this problem. At the same time the author does not know how to prove that the program developed is the shortest possible. Perhaps, one of the readers will succeed in doing one or the other.

Sec. 3.2. Analysis of the Program of Adding Unity

So, we will pass to analysing program V from Sec. 3.1, i.e. to looking into its operation. As we have agreed, we choose, as an initial state, the state belonging to one of the classes E_n ($n = 0, 1, 2, \ldots$). Denote by H_n the class of the states from E_n in which

the cell being examined and the cell next to it on the right are both blank.

We get down to performing program V. The 1st block is executed first. Now we show that its execution will come to an end (i.e. we get out of it and to the 2nd block); at the same time we will see what state the machine will be in after its execution. Let us consider three cases.

Case 1. In the initial state the cell being examined and the cell next to it on the left are both blank. In this case instructions Nos. 1, 2, 4 will work in succession, whereupon the execution of the block is accomplished, the machine coming to a state of class H_n.

Case 2. In the initial state the cell being examined is blank but the cell next to it on the left is labelled. This signifies that the array of length $n + 1$ we are concerned with is to the left of the cell being examined and, consequently, all the cells to the right of it are blank. In this case instructions Nos. 1, 2, 3, 4 will consecutively be executed, whereupon the execution of the block will be accomplished, the machine arriving at a state of class H_n.

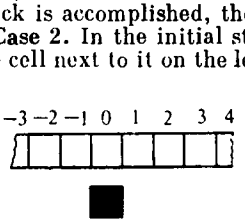

Fig. 23.

Case 3. In the initial state the cell being examined is labelled. This implies that the carriage is positioned just against one of the cells of the array of length $n + 1$; let it face the kth cell to the right of this array. In this case instructions Nos. 1, 3, 4, 3, 4, 3, 4, . . . will operate, the pair of instructions Nos. 3 and 4 being executed precisely k times until the carriage is against the first blank cell to the right; after this the execution of this block terminates and the machine will arrive at a state of class H_n.

So, whatever state of class E_n program V is applied to, the execution of the 1st block will come to an end, whereupon the machine will be in one of the states of class H_n.

To achieve our goal (that is to ascertain that program V is a solution of a general problem on adding unity) it, therefore, suffices to prove that applying this program, *beginning with the 2nd block*, to an arbitrary state of class H_n we will, sooner or later, get a result halt in the state of class E_{n+1}. It is this fact that we are going to prove now.

So, let the machine be in a certain state of class H_n. Let us fix this state and denote it by h. We know that a whole-numbered coordinate system is introduced onto the tape; according to it the cells of the tape are numbered by integers. Let us forget this "old" coordinate system and introduce the following, new (also whole-numbered), system: the cell being examined in state h will be numbered 0; the cells to the right and to the left of it, ± 1, ± 2, ± 3, . . . (Fig. 23). Let us emphasize that this coordinate system depends on state h, of H_n, being considered. Then the array that is a record of number n is found in the cells numbered m, $m + 1$, . . ., $m + n$, where m is any positive or negative number. The cells numbered 0 and 1 are known to be blank. Therefore, either $m + n < 0$ or $m < 1$.

Let us denote by H_n^m the state of the machine in which the cells numbered m, $m + 1$, . . ., $m + n$ are only labelled (and they only)

and the carriage examines cell No. 0. Then class of states H_n consists of states H_n^m, where either $m + n < 0$ or $m > 1$, i.e. of states

(a) H_n^{-n-1}, H_n^{-n-2}, H_n^{-n-3}, ...;

(b) H_n^2, H_n^3, H_n^4, ...

The general view of these states is shown in Fig. 24. The initial state h we are interested in is one of these states H_n^m.

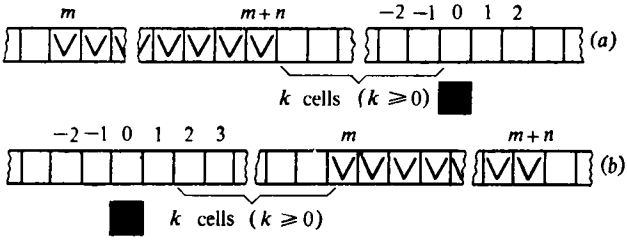

Fig. 24. The state H_n^m: (a) for $m + n < 0$, (b) for $m > 1$.

We are, therefore, to prove that under the program operation, beginning with the 2nd block, each of the states H_n^m where either $m + n < 0$ or $m > 1$ converts, with a result halt, into a state of

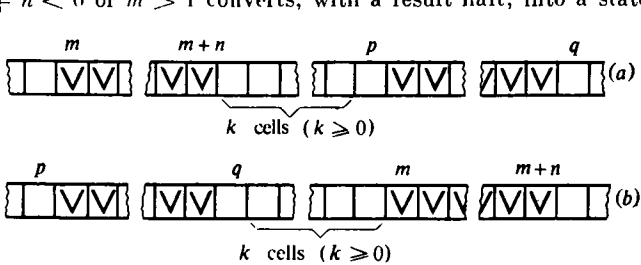

Fig. 25. The state of class $W_n^m(p, q)$ is either (a) for $m + n < p$ or (b) for $m > q$; the carriage can be positioned anywhere.

class E_{n+1}. That is what we are now going to find by following the conversions state H_n^m undergoes when executing in succession the blocks of the program.

With this purpose in view we will introduce certain notations. Let whole numbers m, n, p, q meet the following three stipulations: (1) $p \leqslant q$; (2) $n \geqslant 0$; (3) either $m + n < p$ or $m > q$. We will denote by $W_n^m(p, q)$ the class of all the states of the Post machine that meet the following two stipulations: (1) the cells numbered p and q are blank and all the cells between them, labelled; (2) the cells numbered $(m - 1)$ and $(m + n + 1)$ are blank and all the cells between them, labelled.

The general view of the states of class $W_n^m(p, q)$ is depicted in Fig. 25. We will denote by $R_n^m(p, q)$ the state of class $W_n^m(p, q)$ in which the carriage examines the cell numbered p, and by $S_n^m(p, q)$ the state of class $W_n^m(p, q)$ in which the carriage faces the cell numbered q.

The following assertions can be directly verified:

1°. $H_n^m = R_n^m(0, 1)$.

2°. If before the 2nd block began operating the Post machine state was $R_n^m(x, y)$, with $x - 1 \neq m + n$, then after the 2nd block was executed the machine state will be $R_n^m(x - 1, y)$, the next block to be executed will be the 3rd block.

3°. If before the 3rd block is executed the Post machine state was $R_n^m(x, y)$, the execution of this block will come to an end whereupon the state of the Post machine will be $S_n^m(x, y)$.

4°. If before the 4th block is executed the Post machine state was $S_n^m(x, y)$, with $y + 1 \neq m$, then after executing this block, the state of the machine will be $S_n^m(x, y + 1)$ and the next to be executed is the 5th block.

5°. If before the 5th block is executed the Post machine state was $S_n^m(x, y)$, then the execution of this block will terminate; after that the state of the machine will be $R_n^m(x, y)$.

We will represent the execution of a block by a rectangle with the block number inside it; the states preceding the execution of the block and resulting from the execution will be written immediately to the left and immediately to the right of the respective rectangle. Then, basing ourselves on assertions 1°—5°, the Post machine operation proceeding from state H_n^m and performing our program beginning with the 2nd block, can be represented by the following sequence:

$$H_n^m = R_n^m(0, 1) \quad \boxed{2}\ R_n^m(-1, 1) \quad \boxed{3}\ S_n^m(-1, 1)$$

$$\boxed{4}\ S_n^m(-1, 2), \quad \boxed{5}\ R_n^m(-1, 2), \quad \boxed{2}\ R_n^m(-2, 2)$$

$$\boxed{3}\ S_n^m(-2, 2) \quad \boxed{4}\ S_n^m(-2, 3), \quad \boxed{5}\ R_n^m(-2, 3)$$

$$\boxed{2}\ R_n^m(-3, 3) \ldots$$

The sequence goes on until one of the following two events occurs.

The first event. At a certain moment of time, before the execution of the 2nd block began (perhaps, in particular, at the very beginning), such $R_n^m(x, y)$ proves to be the machine state that $x - 1 = m + n$.

The second event. At a certain moment of time, before the execution of the 4th block began, such $S_n^m(x, y)$ proves to be the machine state that $y + 1 = m$.

Note that one of these events will occur without fail (the question to the reader: why?)

Let us consider each of these events independently.

The first event. Let, by the time the 2nd block began operating, the state of the machine be $R_n^m(x, y)$, where $x - 1 = m + n$. This

state is shown in Fig. 26. After the 2nd block is executed, a state represented in Fig. 27 arises. The last instruction of the 2nd block,

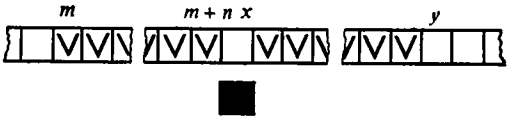

Fig. 26. The state R_n^m (for $m + n + 1$, y) is shown here.

instruction No. 7, "transfers control" to the 6th block, i.e. it is the 6th block that will have to be executed. That the execution of the

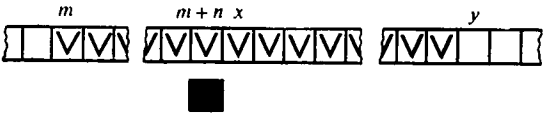

Fig. 27. Here is the state arising if the 2nd block is applied to the state depicted in Fig. 26.

6th block will terminate is easily verified and the state shown in Fig. 28 will arise. The 8th block will be applied to this state which block causes a result halt in this state.

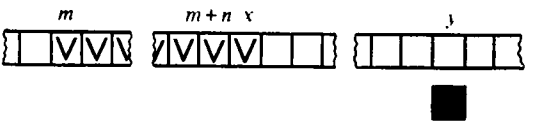

Fig. 28. Here is the state arising after execution of the 6th block applied to the state shown in Fig. 27. In this state a result halt will occur subsequently.

The second event. Let, by the time the 4th block began operating, the state of the machine be $S_n^m(x, y)$, where $y + 1 = m$. This state is demonstrated in Fig. 29. It is easily verified that the 7th

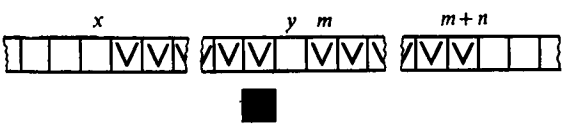

Fig. 29. The state $S_n^m(x, m - 1)$ is depicted here.

block will be executed after executing the 4th block, then the 8th one that will lead to a result halt in the state shown in Fig. 30.

It remains to note that both the state in Fig. 28 and the state in Fig. 30 belong to class E_{n+1}. So, in either of the two possible cases

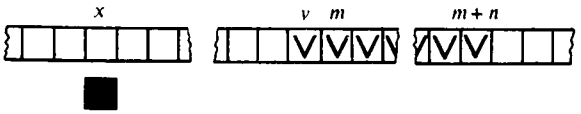

Fig. 30. In this state a result halt will occur after execution of the 4th, 7th and 8th blocks (applied to the state depicted in Fig. 29).

the machine operation will end in a result halt with a state of class E_{n+1} arising, which was to be proved.

Sec. 3.3. Again on the Problem on Addition of Unity

So, the general problem on addition of unity has been solved. Let us analyse its solution. The reader will, of course, admit that the main difficulty in solving this problem is to find the array of length $n + 1$ recorded onto the tape rather than to link one more cell to this array. When solving easier problems on addition of unity that were discussed in Chap. 2, we knew the direction in which to look for the array and, correspondingly, built the program that began operating with shifting the carriage in the respective direction (unless at the very beginning the carriage turned out to face the array sought for). Solving the general problem we do not know the direction where to move. That was precisely the reason why this problem must at first seem unsolvable to some readers.

What is the way by which one can hit upon program V or other similar program that the reader has already built or, perhaps, will build? Here it is. Imagine we are standing on the road that is infinite both ways, and somewhere on this road there is a magic stone we are looking for. We are not aware of the place where it is, we are only aware of that it is sure to lie somewhere. How shall we act? Whatever direction we have chosen the magic stone may happen to lie just in the opposite direction. We should, obviously, walk in turn now in one direction, now in the other, all the time making greater the swing of the alternate movement: first take a step in one direction, then two steps in the other, then again three steps in the one direction and then four steps in the other direction, and so on until we stumble across the magic stone.

Here is what we will now think over. How shall we judge the moment when to turn back, that is to change the direction of movement? One would think it is very simple: we must only count the steps and know, at each moment, how many steps we must make in the given direction and how many steps we have already made. In fact, this is a matter of some difficulty, that is how to remember the numbers of steps. If we retain them in our memory, we must be prepared to remember numbers as large as necessary which is impossible; actually, if the magic stone is quite far away from our start position, we will have to remember numbers whose remembering

surpasses our brain capabilities. We can write down the number of steps, say, in a notebook carried on ourselves but then we must be prepared to carry on ourselves as bulky notes as necessary (reading these notes will become a separate task in that case). If carrying any notes on ourselves is not allowed, it may seem, at first sight, impossible to overcome this obstacle. Yet there is a way out of this difficulty. It is as follows: we can employ the road itself to make notes. And we will take advantage of it. Following Tom Thumb's example, we will mark out steps on the road by crumbs or by little stones: having made a step, we will put, say, a little stone before us (unless a little stone is already there). Thus, following the little stones and reaching the place where there is no stone, we will put one more little stone and change the direction for the opposite one, and so until we find the magic stone. This is just the method of searching for the array that was realized in program V; the role of little stones was played by labels printed by the carriage to mark the cells it moved through. Note one more circumstance: in our case a label also plays the part of the magic stone; that is why we should see to it that a little stone and a magic stone should not be mixed up (in program V this was done by check blocks). The Tom Thumb method has a shortcoming consisting in that we have to keep in our pocket an unlimited supply of little stones or crumbs (of course, this shortcoming is not essential for the Post machine). We can, though, do with two little stones only. For this purpose we must put one little stone to our left and the other, to our right and then walk between them and move them; namely, every time we reach the little stone we are to carry it one step forward, turn back and walk backwards until the other little stone is reached that should be dealt with in exactly the same way.

Exercise. Build the program for the Post machine that realizes the method just described of searching for the magic stone with the aid of two little stones. Compare the length of the program built with that of program V.

Let us turn back to considering program V once again. The role of each block in this program is clear from its designation: the moving blocks move the carriage along the array of labels, printed by it before, to the end of the array; the check blocks control whether or not the carriage approached closely the record of the number sought; the erasing blocks erase the labels the carriage have printed throughout its moving. If a state of class H_n is first set and the program is performed beginning with the 2nd block, the required result will be produced. (So, if the program is required to produce a result when applied to a state from H_n only, we could have limited ourselves to the latter 19 instructions, first making, of course, all the numbers of addresses and jumps less by 4.) The purpose of the start block is to drive the machine out of an arbitrary state of class E_n into some state of class H_n. We will note that the start block operates in all the events: even if the carriage in the initial state is *already* positioned against the number record sought for, the start block moves the carriage off this array in order to begin, after that, searching for the same array. It, obviously, seems senseless. Is it not easier first to recognize whether or not the carriage is, at the beginning, against the array (i.e. whether Case 3 on p. 44 takes place)? If it is, one more label is to be added on (just as was required

in problem 2, Sec. 2.3); if it is not, move one cell to the left and see whether there is a label there (Case 2 on p. 44); if there is a label, add one more label on the array sought for (as was required in problem 1′, Sec. 2.2); if no—in this case only, with two no's—instructions. Nos. 5-23 of program V are to be included.

Let us build the program realising this scheme. It is expedient to build a program together with its block diagram. To this end, we shall refer to any of the programs of length 4 that are the solutions of problem 2 from Chap. 2 as program II; to any of the programs of length 3 that are the solution of problem 1′ from Chap. 2—as program I′; and the list of instructions from No. 5 to No. 23 from program V will be denoted by Γ. The block diagram in Fig. 31 represents the program being sought for. It will be recalled that, according to the designation in Sec. 2.4, for any list of instructions Ξ, Ξ [+m] denotes the list produced from Ξ by increasing the number of all the instructions and all the jumps by m.

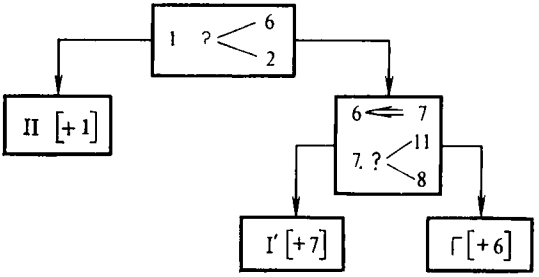

Fig. 31.

The program in Fig. 31, with program II chosen properly (namely, in case program II contains a move-to-the-right instructions, i.e. is one of programs II'_1, II'_2, . . ., II'_{12} mentioned in Exercise 3, Sec. 2.3), leads to a result halt by a procedure that is not longer than program V, i.e. in any initial state it does not need more steps than program V (we recommend that the reader should verify that it is not so in case when program II contains a move-to-the-left instruction, i.e. is one of programs II_1, II_2, . . ., II_{12}). In certain cases the program in Fig. 31 leads to a result halt even by a shorter procedure than program V, i.e. requires less number of steps of the machine operation. We invite the reader to verify this and make sure that the program in Fig. 31 requires, to lead to a result halt, either as many steps as program V does (for initial states as in Case 1 on p. 44), or 6 steps less (for initial states as in Case 2), or 5 steps less (for initial states as in Case 3).

But the program in Fig. 31 contains 29 instructions. Even if the three halt instructions of this program are merged into one, just the same it contains 27 instructions, which is more than 23. It is not strange as here there are actually three different programs for each of the three possible cases on p. 44 that may arise at the beginning. The role of the start block—somewhat paradoxical at first

sight—in program V just consists in reducing all the possible states to one—the state of class H_n. Owing to it the gain in the program length is achieved. Here we use, thus, one of the methods practised in mathematics and consisting in trying to reduce the solution of all arising problems to the solution of problems already solved. (In the example discussed, we reduce the solution of problem on addition of unity for an arbitrary initial state to the solution of the same problem for states of H_n.) This standard method generally results—if applied effectively—in various sorts of gain, for instance, in gain in the storage capacity needed to memorize the program (and in the general case, the method of solution) of one class of problems or other. It, therefore, plays an important role in mathematics. There is a joke popular, not without reason, with mathematicians who (at least some of them) take pride in it; the joke involves two problems.

Problem One. Given: matches, the stove unlit, the tap turned off, the empty kettle; required: to boil water. *Solution.* Strike a match, fire the stove, turn on the tap, fill the kettle with water, put it on the stove, boil the water.

Problem Two. Given: the stove fired, the tap turned on, the kettle filled with water; required: to boil the water. *Solution.* Reduce the problem to that already solved (i.e. to Problem One). With this purpose we quench the stove, pour the water out of the kettle, turn off the tap, whereupon it remains to act as in the solution of Problem One.

Note. The amount of actions for reducing Problem Two to Problem One can be cut down combining quenching of the stove with pouring water out of the kettle onto the stove.

Sec. 3.4. Addition of Numbers in Simple Cases

Now get down to addition of numbers on the Post machine. Only nonnegative whole numbers will be considered as addends. We will, as before, represent nonnegative whole number m by an array of length $m + 1$. To perform addition of numbers on the Post machine means to build the program that, being applied to the tape containing the record of numbers $m_1, m_2, \ldots, m_k$ $(k \geqslant 2)$, would lead to a result halt, the record of number $m_1 + m_2 + \ldots + m_k$ appearing on the tape after this halt. A number of specifications are to be made. First of all we will always assume that the tape contains, at the beginning, no record but that of numbers $m_1, \ldots, m_k$ and require that is should, at the end, contain no record except their sum; we will impose no restrictions on the position of the carriage at the end; as to its position at the beginning, we will, for simplicity, believe that in the initial state the carriage faces the leftmost of the labelled cells.

Next, various assumptions can be made on the distance between arrays serving for recording numbers. (The number of cells between the rightmost cell of the left-hand array and the leftmost cell of the right-hand array is, naturally, called the distance between two arrays.) In the simplest case (that we begin with) the distance between the neighbouring arrays equals 1. Finally, certain restrictions can be imposed on the number of addends believing their number to be earlier specified or arbitrary.

The simplest case is regarded in problem (a).

Problem (a). Build the program of addition of two numbers recorded at the distance of 1 from each other.

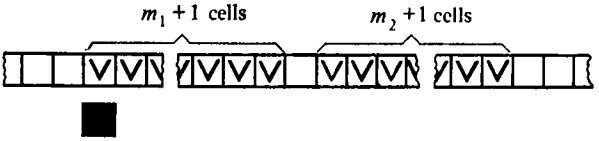

Fig. 32.

In virtue of the assumptions made the initial state for problem (a) will look as in Fig. 32.

The simplest method of solving this problem is filling in the empty cell between the arrays putting the label. Then, if m_1 and m_2 are the addends, $m_1 + m_2 + 3$ labelled cells will arise on the tape while there should be $m_1 + m_2 + 1$ labelled cells. Two unnecessary cells should be erased. Program $\tilde{A}$ to be found below is the one that realises the scheme of actions just stated.

Program $\tilde{A}$

1. $\Rightarrow 2$	4. $\Rightarrow 5$	7. ξ 8	
2. ? 3 / 1	5. ? 6 / 4	8. $\Leftarrow 9$	
3. $\vee$ 4	6. $\Leftarrow 7$	9. ξ 10	
		10. stop.	

We will gain in the number of instructions if we erase two labels not at the end but at the beginning immediately in the left-hand array. We should be careful here in order to allow for the case when there is only one label in the left-hand array. This scheme is realised in program A con-

sisting of 8 instructions only. Instruction No. 3 just takes into account the case when there is only one label in the left-hand array.

Program A

1. ξ 2
2. $\Rightarrow$3
3. ?$\begin{smallmatrix}/8\\\backslash 4\end{smallmatrix}$
4. ξ 5
5. $\Rightarrow$6
6. ?$\begin{smallmatrix}/7\\\backslash 5\end{smallmatrix}$
7. $\vee$ 8
8. stop.

Problem (b). Build the program of adding arbitrary amount of numbers recorded onto the tape at the distance of 1 from one another.

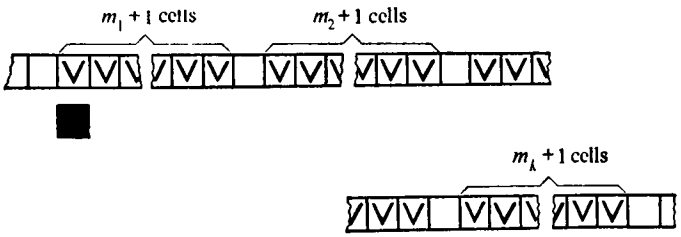

Fig 33.

By virtue of the assumptions made the initial state for problem (b) will be as shown in Fig. 33. It follows that the program should be made up that, for any number k and any numbers $m_1, m_2, \ldots, m_k$, would lead to a result halt in the state in which number $m_1 + m_2 + \ldots + m_k$ is recorded on the tape. Here is the shortest (more precise, one of the shortest) program known to the author:

Program B

1. ξ 2
2. $\Rightarrow$3
3. ?$\begin{smallmatrix}/4\\\backslash 2\end{smallmatrix}$
4. $\vee$ 5
5. $\Rightarrow$6
6. ?$\begin{smallmatrix}/7\\\backslash 8\end{smallmatrix}$
7. stop
8. $\Leftarrow$9
9. ?$\begin{smallmatrix}/10\\\backslash 8\end{smallmatrix}$
10. $\Rightarrow$11
11. ξ 12
12. $\Rightarrow$1.

We recommend that the reader should verify this program (as well as the rest of the programs in this section) employing a specific example and, thereby, comprehend the idea underlying its operation. The idea in the given case is that we split the labels off consecutively, two at a time, on the left (by instructions Nos. 1 and 11) putting instead one label into the nearest blank cell on the right (by instruction No. 4); if the cell next to the nearest blank cell on the right is also blank (it is verified by instructions Nos. 5 and 6) that signifies it is time to have done with it (transfer of control to instruction No. 7).

Now let us pass on to adding numbers recorded onto the tape at an *arbitrary* distance from one another.

Problem (c). Build the program of adding two numbers recorded at an arbitrary distance from one another.

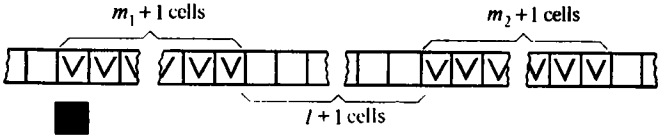

Fig. 34.

The initial state for problem (c) is shown in Fig. 34. The program leading to the goal should be built, no matter what nonnegative whole numbers m_1, m_2, l are. Here is one of such programs:

Program $\tilde{B}$

1. ξ 2	5. $\Rightarrow$ 6	9. $\Rightarrow$ 1	13. ξ 14
2. $\Rightarrow$ 3	6. ? 7 / 10	10. $\Leftarrow$ 11	14. stop.
3. ? 4 / 2	7. $\Leftarrow$ 8	11. ? 12 / 11	
4. $\vee$ 5	8. ? 9 / 7	12. $\Rightarrow$ 13	

The idea of this program is that the left-hand array "shifts" to the right until it merges with the right-hand

one. The shifting of the array is effected by removing the leftmost cell of the array to the nearest blank cell on the right (the label is split off by instruction No. 1 and printed by instruction No. 4). When the arrays become one array (it is made obvious by instructions Nos. 5 and 6), $m_1 + m_2 + 2$ cells prove to be labelled, i.e. one cell more than needed. Instructions Nos. 10, 11, 12 move the carriage to the left-hand end of the array, where the unnecessary label is erased (by instruction No. 13). So (it is important for further discussion), the carriage in the final position faces the cell immediately to the left of the resulting sum.

When solving problem (a) we have already seen that the program length can be shortened if the unnecessary label is erased at the start of the machine operation rather than at the end of it. Employing this method, we obtain the following program B_1 of length 12 practising the same method of shifting the left-hand array to the right and, therefore, greatly resembling program $\widetilde{B}$:

Program B_1

1. ξ 2	5. $\Rightarrow$ 6	9. ? 10 / 12
2. $\Rightarrow$ 3	6. ? 7 / 5	10. $\Leftarrow$ 11
3. ? 12 / 4	7. $\vee$ 8	11. ? 2 / 10
4. ξ 5	8. $\Rightarrow$ 9	12. stop.

The shorter programs than those of 12 instructions satisfying the conditions of problem (c) are not known to the author. At the same time many solutions of problem (c) of length 12 can be arrived at. First of all 11! programs can be easily built as a result of all kinds of rearrangements formed of all the instructions of program B_1 except the first one (each such rearrangement must, of course, be carried along with the appropriate change in the numbers and jumps). Besides, fundamentally different methods of addition can be sought for. For instance, the

method is possible to practise in which the following procedure is repeated many times: one label is added on the right to the record of the left-hand number, but instead one label on the left of the right-hand number is erased, and so on until the right-hand array is exhausted: an unnecessary label is erased at the very beginning by instruction No. 1. Program B_2 of length 12 found below realises this method.

Program B_2

1. ξ 2	5. $\Rightarrow$ 6	9. ? 12 / 10
2. $\Rightarrow$ 3	6. ? 5 / 7	10. $\Leftarrow$ 11
3. ? 4 / 2	7. ξ 8	11. ? 10 / 2
4. $\vee$ 5	8. $\Rightarrow$ 9	12. stop.

Sec. 3.5. Addition of Numbers in More Complicated Cases

Problem (d(k)). Build, for every k, the program of adding k numbers recorded with arbitrary distances between them.

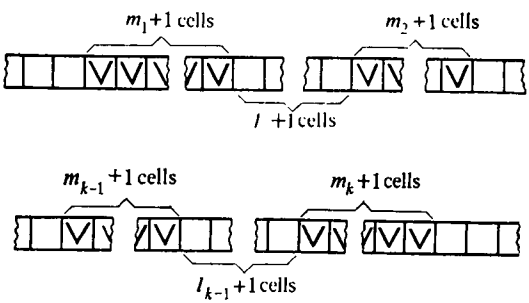

Fig. 35.

The initial state for problem (d(k)) is shown in Fig. 35. The program must, therefore, be built, for every k, that

leads to the goal with any nonnegative whole numbers m_1, m_2, . . ., m_k, l_1, l_2, . . ., l_{k-1}.

We will devise the program being sought for in the following way.

Let Σ be a program that is a solution of problem (c) and possess the following properties:

(1) when program Σ is applied to the initial state in problem (c) (i.e. the state in which there are records of two numbers on the tape and nothing else and the carriage examines the leftmost of the labelled cells) the carriage is never positioned to the right of the rightmost cell labelled in the initial state,

(2) after program Σ applied to the initial state for problem (c) has been performed, the carriage appears to face the leftmost of the labelled cells of the resulting array,

(3) the program contains only one halt instruction and it is the last instruction of the program.

Let program Σ contain $a + 1$ instructions. We denote a list of the earlier a instructions from Σ by Ξ. Let us build such a program:

Program Γ^* (k)

Ξ ⋮

$\Xi\,[+a]$ $\Xi\,[+(k-2)a]$

$\Xi\,[+2a]$ $(k-1)a+1$. stop.

⋮

Let the reader verify that this program is a solution of problem (d (k)) (actually, if $s_i = m_1 + \ldots + m_i$, it is easy to observe that the instructions from list Ξ perform the addition of numbers m_1 and m_2, the instructions from list $\Xi\,[+a]$, the addition of numbers s_2 and m_3, and so on; the instructions from list $\Xi\,[+ia]$, the addition of numbers s_{i+1} and m_{i+2}; finally, the instructions from list $\Xi\,[+(k-2)a]$ perform the addition of numbers s_{k-1} and m_k, i.e. provide the arrival at the sum sought for.

Exercise. In what way are properties (1)-(3) used if program $\Gamma^*(k)$ is found out to be a solution of problem $(d\,(k))$?

It remained to find program Σ with the properties required. Neither of programs $\widetilde{B}$, B_1 and B_2 will do. Program $\widetilde{B}$, however, possesses properties (1) and (3). It is easy to change it slightly so that property (2) was fulfilled too. It is sufficient to replace the last instruction of program $\widetilde{B}$ by the following two instructions:

14. $\Rightarrow$ 15 15. stop.

In view of the comment made (when accounting for program $\widetilde{B}$ operation) on the final position of the carriage after performing program $\widetilde{B}$, the resulting 15-instruction program $\widetilde{B}^*$ will possess not only properties (1) and (3) but property (2) as well.

Program B_2^* of length 15, resulting from program B_2, can also be taken as program Σ as the basis for building of program Γ^* (k) if, instead of the last instruction in this program, we will take the

following four:

12. $\Leftarrow 13$ 14. $\Rightarrow 15$

13. $?\begin{smallmatrix}\nearrow 14\\ \searrow 12\end{smallmatrix}$ 15. stop.

Program B_2^* possesses properties (2) and (3) and a certain property weaker than (1) (a question to the reader: what property?) but sufficient for program $\Gamma^*(k)$, based on B_2^*, to be a solution of problem (d (k)).

If there are 15 instructions in program Σ there will be $(k-1)\,14+1=14k-13$ instructions in program $\Gamma^*(k)$.

The program can, however, be built that meets the requirements of problem (d (k)) and contains less than $14k-13$ instructions.

In view of this we will denote the following list of instructions by Δ_i $(i=0, 1, 2, \ldots)$:

$3+9i.$ $\Rightarrow 4+9i$ $\quad$ $8+9i.$ $\Leftarrow 9+9i$

$4+9i.$ $?\begin{smallmatrix}\nearrow 3+9i\\ \searrow 5+9i\end{smallmatrix}$ $\quad$ $9+9i.$ $?\begin{smallmatrix}\nearrow 8+9i\\ \searrow 10+9i\end{smallmatrix}$

$5+9i.$ $\xi\, 6+9i$ $\quad$ $10+9i.$ $\Rightarrow 11+9i$

$6+9i.$ $\Rightarrow 7+9i$ $\quad$ $11+9i.$ $\vee\, 3+9i$

$7+9i.$ $?\begin{smallmatrix}\nearrow 3+9\,(i+1)\\ \searrow 8+9i\end{smallmatrix}$

We will denote the following program by $\Gamma(k)$:

Program $\Gamma(k)$

1. $\Rightarrow 2$ $\quad \Delta_0$

2. $?\begin{smallmatrix}\nearrow 3\\ \searrow 1\end{smallmatrix}$ $\quad \Delta_1$

$\vdots$

Δ_{k-2}

$9k-6$. stop.

Let the reader make sure that, first, $\Gamma(k)$ is actually a program and, second, it is a program meeting the requirements of problem (d(k)), there being $9k-6$ instructions in the program. Note that putting $k=2$, we get, in the form of $\Gamma(2)$, a new solution of problem (c), this solution containing 12 instructions, as many as there are in programs B_1 and B_2. Program $\Gamma(2)$ operates by the same method as program B_2 which consists in removing labels from the left-hand array to the right-hand one; unlike B_2, however, when performing program $\Gamma(2)$ the label is first erased in the right-hand array and is then added in the left-hand one, the last label in the right-hand array being erased without a new label arising in the left-hand one.

Problem (e). Build a program of addition of arbitrary quantity of numbers recorded at an arbitrary distance from one another.

Each program which is a solution of problem (e) will, obviously, at the same time, be a solution of each of the preceding problems. None of the solutions of problems (a), (b), (c), (d(k)) given above is not, however, a solution of problem (e) (check it).

The initial state for problem (e) is depicted in Fig. 35. Now a *unified* program is to be built that leads to the goal for *arbitrary* k. Try to build such a program on your own. You will see it is not easy. And, perhaps, there is no such a program at all? Then try to prove it. The question whether problem (e) has a solution will be answered in the next chapter.

4. THE POST MACHINE POTENTIALITIES

In the present chapter we will be engaged in the question: what calculations in general can be performed on the Post machine. In this connection we will have to touch upon the general notion of algorithm. An effort to compare the Post machine with electronic computers will be made in the concluding section. The account begins with analysing the problem left unsolved in Chap. 3.

Sec. 4.1. On the Problem of Addition of Numbers at Arbitrary Distances

Problem (e), the most general problem of addition of numbers recorded on the tape of the Post machine, was raised at the end of Chap. 3. This problem required finding such a common program for the Post machine that would perform addition of arbitrary quantity of numbers recorded onto the tape at arbitrary distances between them. Nothing but these numbers were assumed to be recorded onto the tape and the carriage was assumed to be positioned, at the beginning, against the leftmost of the labelled cells. At the termination of the program operation a sum of all the initial numbers and nothing else is to be recorded onto the tape (i.e. the rest of the tape is to be blank).

It will be recalled that the tape is rigidly linked to the "constant" coordinate system, in accordance with it the cells of the tape are numbered by integers from $-\infty$ to $+\infty$. We will regard, for definiteness, that all the initial numbers are recorded in the right-hand, "nonnegative", portion of the tape, the leftmost of the labelled cells being numbered 0. So, the carriage in the initial state examines cell No. 0.

We will try to analyse the solution of the problem posed without predicting, for the present, how its solution looks like.

So, let us assume that a certain program is a solution of problem (e) and denote it by E. Let us regard the state of the Post machine shown in Fig. 36. Two numbers each of

which equals 0 are recorded onto the tape in this state at the distance of a unity between them. We will take this state as initial. Then program E is to lead to a result halt, the number 0 being recorded in this case. The carriage, during the machine operation right up to the result halt, will only examine a finite number of cells. Let z be

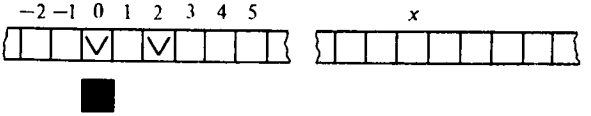

Fig. 36. Here $x \geqslant 3$.

the greatest number of cells examined by the carriage. Denote by x the greatest of numbers z and 3. Let us now regard the state of the Post machine in which the cells labelled are 0, 2, $x + 2$, while the rest are blank, the carriage will be positioned against cell No. 0 (Fig. 37).

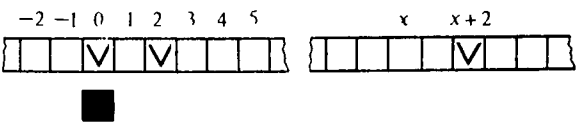

Fig. 37.

We will take this state as initial and apply program E to it. Let us see how it will work. It is easily seen to work "in the same way" as when applied to the initial state in Fig. 36.

"In the same way" taken in quotes is specified as follows. Let us consider two Post machines. We will say that a certain state of one machine is equivalent to a certain state of the other if in these states (1) the carriage of the first machine faces the same cell number as the carriage of the second; (2) each cell of the first machine, positioned to the left of cell No. $(x+1)$ is labelled or empty at the same time as the cell with the same number of the second machine; (3) all the cells of the first machine positioned to the right of cell No. x are empty; (4) in the second machine cell No. $(x + 1)$ is empty, cell No. $(x + 2)$ is labelled, while all the cells to the right of cell No. $(x + 2)$ are empty. We will assume that both machines operate by program E synchronously, the initial state of

the first machine is that shown in Fig. 36 and of the second machine is that shown in Fig. 37. These states are equivalent. It is easily proved by induction in terms of t that at any moment of time t the following assertion holds true: the states of both machines at this moment are equivalent and the instruction to be executed by the first machine coincides with the instruction to be executed by the second machine. It follows that, operating synchronously, both machines will pass from certain equivalent states to other equivalent states under the action of the same instructions. This very thing was meant when the program applied to the state in Fig. 37 was said to work in the same way as it does when applied to the state in Fig. 36. The machines will, synchronously, at a definite moment, come to a terminal state in which the halt instruction will work. Both machines will come to a result halt at the same time. Number 0 will be recorded onto the tape of the first machine, i.e. there will be only one labelled cell, this cell being to the left of cell No. $(x + 1)$. Owing to equivalent terminal states, there will, consequently, be one labelled cell on the tape of the second machine to the left of cell No. $(x + 1)$ and, besides, cell No. $(x + 1)$ itself will be also labelled; there will be two labelled cells in all.

We came to the conclusion that program E applied to the state in Fig. 37 will lead to a result halt in the state in which precisely two cells will be labelled.

On the other hand, in the state shown in Fig. 37 there are records of three numbers on the tape each of which equals zero (the distances between them equal, respectively, 1 and $x - 1$), while the rest of the tape is empty and the carriage faces the leftmost of the labelled cells. Program E is, by assumption, a solution of problem (e). It is, therefore, to lead, when applied to this initial state, to a result halt in the state in which the sum of initially recorded numbers will prove to be recorded onto the tape while the rest of the tape is to be empty. In the given case the sum is equal to 0. So, only one cell will be labelled on the tape in the terminal state. But this is inconsistent with the above conclusion.

The inconsistence found out makes us certain that there can be no program which is a solution of problem (e).

The reason why problem (e) has no solution is obvious:

however long the carriage has travelled along the tape it, so to say, "will never know" whether it has already bypassed the records of all the addends or, perhaps, there is one more addend far to the right that the carriage is still to reach. We cannot, therefore, devise the program that would lead to a result halt, ensuring that all the addends are taken into account. Such a program can be built only for particular cases, for instance, when the number of addends is known beforehand (as in problem (d(k)) from Sec. 3.5) or when the addends are recorded at the specified (as in problem (b)) or, at least, at the limited distances between them.

Exercise. Devise the program, for every q, of adding an arbitrary quantity of numbers recorded at arbitrary, but not exceeding q, distances between them.

Problem (e) illustrates how important is the method of recording data on the tape.

Sec. 4.2. Post's Proposal

In the foregoing chapters the reader gained certain experience in programming for the Post machine. To confirm this experience we recommend doing exercises 1-5. The reader is supposed to choose on his own the initial position of the carriage and in exercises 2, 3, 5 the distance between the records of initial numbers as well. Nonnegative integers are implied throughout the given chapter.

Exercise 1. Build program of dividing numbers by 2, by 3, by 4, . . ., by the assigned number k. **Hint.** By division is understood finding the quotient or the partial quotient, so that the result of division of 7 by 3 will be 2.

Exercise 2. Devise the program of subtraction of one number from the other. **Hint.** If subtraction is impossible (the subtrahend is greater than the minuend), the program is not to lead to a result halt.

Exercise 3. Build the program of multiplication of two numbers.

Exercise 4. Build the program of squaring numbers.

Exercise 5. Build the program of division of one number by the other. **Hint.** If the divisor is 0, the program is not to lead to a result halt.

We will need for further discussion a fundamental, though simple, concept of an ordered number tuple. Ordered

pairs of numbers rather than individual numbers have often to be dealt with; both terms of the pair may be identical. We will put down the ordered pair like this: the first term, then the second, and enclose the pair in angle brackets. For instance ⟨33, 4⟩ and ⟨2, 2⟩ are ordered pairs. A notion of ordered three-tuple is introduced in a similar way. Such three ordered tuples are, for instance, possible: ⟨2, 5, 1⟩; ⟨6, 6, 6⟩; ⟨4, 5, 4⟩. And ⟨3, 8, 7, 8, 8, 2, 5⟩ is an ordered seven-tuple. An ordered set of numbers of any length is called a *number tuple*; the ordered pairs, the ordered three-tuples and the ordered seven-tuple written out above are tuples. The number of positions in a tuple is called its *length*. Thus ⟨6, 2, 2, 6⟩ is of length 4.

All the problems and the most of exercises in building programs dealt with so far were of the following type. A certain class of initial data was recorded: numbers (as in Exercises 1 and 4 of the present section or in a problem on addition of unity in Chap. 2), number tuples of the specified length (of length 2 as in Exercises 2, 3 and 5 in this section and in problem (a), (c) and (d(2)) in Chap. 3 or of length k as in problem (d(k)) in Chap. 3), number tuples of arbitrary length (as in problem (b) and (e) in Chap. 3).

Each initial datum from the chosen class—a number or a tuple—either was placed in correspondence to a certain resulting number (the square of the initial number, or the quotient of the first term of the initial tuple by the second, or the sum of all the terms of the initial tuple, and so on) or was not placed in correspondence to anything (for instance, the quotient or the partial quotient only corresponded to the pair whose second term is other than zero). The program was to be devised that would convert (with a result halt) any initial datum recorded onto the tape into a resulting number if any; if, however, being applied to the record of the initial datum with no resulting number determined the program is not to lead to a result halt. We will always imply, for simplicity, that in the initial state only an initial datum is recorded onto the tape and the carriage examines the leftmost of the labelled cells and that in the terminal state a resulting number is also recorded onto the tape and the carriage is positioned anywhere. We still have to specify how to record the number tuples. In Sec. 4.1 we saw that the very existence

of the program required depends on the way of recording the tuple, namely, on the distances between its terms. We will agree to record the tuple by placing its terms at the distance of unity between them. With an initial position of the carriage and the way of recording numbers and tuples so fixed we can speak, for brevity, of applying a program to a number or to a tuple and of converting them into a number or a tuple having actually in mind the records of numbers and tuples.

Now we will naturally ask the question: In what cases do the problems of the type just described on building programs have solutions? In other words, what calculations can be performed on the Post machine?

The answer to it is as follows. The problem on building a program leading from the initial datum to the resulting number has a solution if and only if there is some general method permitting to write down a resulting number from an arbitrary initial datum. This assertion which is an answer to our question will be called *Post's proposal* due to its author. It should be stressed that in Post's proposal a certain uniform method (as well as a uniform program) common for all the initial data is meant. In addition, it is assumed that if there is no resulting number the method under consideration (as well as the program) does not lead to a result which, if it had occurred, would have been known false.

Example 1. Problem "Build the cubing program for the Post machine" in solvable because there is a general method permitting to calculate n^3 by n.

Example 2. For the same reason the program for the Post machine that converts the record of the tuple $\langle x, y, z\rangle$ into the record of the number $\langle x^y + zy\rangle (z^2 - y)$ exists. This program (as well as the respective method of calculations) leads to no result for $z^2 < y$ (all the numbers we are dealing with are nonnegative!).

Example 3. Let it be required to build the program for the Post machine possessing the following property: if in the decimal expansion of number π there are $n + 2$ decimal digits occurring consecutively with only the first and the last digits being other than 9 and the rest of n's being 9's, the program converts this number n into 1; if, however, there is no such sequence of $n + 2$ digits in the expansion of π, the program converts n into 0. In the giv-

en case, though certain resulting number ξ_n equal to 0 or 1 corresponds to each n, the general method of calculating ξ_n for arbitrary n is not known to us. The analysis of the earlier 800 digits of expansion of π* demonstrates only that $\xi_n = 1$ for $n = 0, 1, 2, 6$. In general, however, many digits of expansion of π are written out we can, from the digit sequence obtained, extract only the information on equality of ξ_n to unity for definite values of n and no information on equality of ξ_n to zero for, at least, some n. Even if the equality $\xi_n = 0$, for any n, can be determined, it is only by an indirect method. The general method of calculating ξ_n for any n is unknown to us. If there is no such method, then, on the strength of Post's proposal, there can be no program needed. (Note that applying Post's proposal in this direction is obvious: if there were the program needed, the program alone would give a certain general method of calculating a resulting number from any n.) If, however, there is such a method, then owing to Post's proposal (its application in this direction is not obvious) the program needed exists.

Example 4. Let us determine number ζ_n for each n by the following method: $\zeta_n = 1$ if for any k in the decimal expansion of π a consecutive run of 9's occurs (just as there are 0, 1, 2, 3, 4, 5, 6 of such 9's); otherwise $\zeta_n = 0$, i.e. if there is the maximum array length of 9's following one after the other (this maximum length is sure to be more than 5, if any). The question is whether the program for the Post machine that converts each n into ζ_n exists. The answer can seem rather strange: "Yes, it does, although we cannot demonstrate it". In fact, either a consecutive run of any number of 9's occurs in the expansion of π or it does not. There is the general method of calculating ζ_n in each of these cases: we must put $\zeta_n = 1$ (for all n) in the former case and $\zeta_n = 0$ (for all n) in the latter case. It is another matter that we do not know which of the two general methods to choose as we do not know which of the two cases with regard to the expansion of π actually takes place. For each of these general methods of calculating ζ_n there is a corresponding program performing the calculation of ζ_n on the Post machine: for the general method ($\zeta_n \equiv 1$)

* They are given in: V. Litsman, *Giants and Dwarfs in the Realm of Figures*, Fizmatgiz, Moscow, 1959, p. 63 (in Russian).

Program

1. ξ 2
2. $\Rightarrow$ 3
3. ? $\swarrow^{4}_{1}\searrow$
4. $\Leftarrow$ 5
5. $\vee$ 6
6. $\Leftarrow$ 7
7. $\vee$ 8
8. stop

and for the other general method ($\zeta_n \equiv 0$)

Program

1. $\Rightarrow$ 2
2. ? $\swarrow^{4}_{3}\searrow$
3. ξ 1
4. stop.

We can, with certainty, maintain that one of the programs presented meets the demands made but the state of modern knowledge does not let us say which.

That the respective uniform method exists was known in Examples 1, 2 and 4 and remained unknown in Example 3. There are examples which are known to lack such method but they are too intricate to present them here.

Sec. 4.3. The Post Machine and Algorithms

The notion of a uniform method of calculation common for the entire class of possible initial data is one of the most important in mathematics, both "theoretical" and "everyday". It is this notion that the teaching of mathematics in the primary school is based on. Numerous exercises in four operations with multi-digit numbers do not have for an object learning to add, subtract, multiply and divide the numbers occurring in these exercises. They are aimed at practising the methods of addition, subtraction, multiplication and division in columns bearing in mind, of course, that these methods are applicable to *any* ordered pairs of numbers, not only to those occurring in the book of problems. When the pupil is said to be able to add it is not implied that sooner or later he will find the sum of any pair of numbers but it is implied that he knows the general method of addition.

So important a notion needs, of course, a special term to name it: the word "algorithm" is employed as the term. We can now say that an algorithm exists in Examples 1, 2 and 4 (although we do not know what it is like for Example 4), an algorithm is unknown in Example 3 (it is even unknown whether or not it exists). As it was noted, there are cases when an algorithm needed is not known to exist.

Although the idea of algorithm is often encountered in practice and in science, it is not always taken notice of: the word "algorithm" itself is not popular enough yet. Only "Euclidean algorithm" is a generally known word combination that serves to name one of the algorithms for finding the greatest common divisor. We, thereby, often find ourselves in a situation of a character from Moliere's *Le Bourgeois gentilhomme* who discovered, as late as at a mature age, for the first time that he had been speaking prose all his life. However that may be, we now know that what we were taught in school was algorithms.

Each algorithm is assumed to possess a certain set called the *range of its possible initial data* and consisting of all the objects (for instance, numbers or tuples) to which the algorithm under consideration is worth trying to apply. When applied to some of its possible initial data an algorithm either may produce a result (in this case an algorithm is said to be *applicable* to this initial datum and converts it into a result) or may not produce a result (in this case an algorithm is said to be *inapplicable* to this initial datum). For example, the initial data for the algorithm of subtraction in columns are the ordered pairs of numbers, the algorithm being applicable only to those whose second term is not more than the first. To think that for this algorithm the class of possible initial data consists only of pairs $\langle a, b \rangle$ where $a \geqslant b$ is hardly expedient. For we can try to subtract the second number from the first for the pair $\langle 1244444445444445, 124444454444445 \rangle$ as well; simply we will not obtain a result.

Let us see how the idea of algorithm is associated with the Post machine. We will consider the programs of the Post machine that being applied to any number either lead to no result halt at all or again produce an integer as a result (i.e. if a result halt occurs, an integer and only this integer appears on the tape). Each program like this defines the following algorithm whose range of possible

initial data is a set of all nonnegative integers and all the results are also integers: an initial integer is taken, and recorded onto the tape of the Post machine, the carriage is positioned against the leftmost of the labelled cells, the machine is started according to the program under consideration, a result halt is awaited and the number that appears recorded onto the tape is read.

Let us fix some number k and consider the programs that being applied to any tuple of length k either lead to no result halt at all or produce a number as a result. Each program like that defines the following algorithm whose range of possible initial data is a set of all the number tuples of length k and all the results are integers: an initial number tuple is taken, it is recorded onto the tape of the Post machine, the carriage is positioned against the leftmost of the labelled cells, the machine is started according to the program under consideration, a result halt is awaited and the number that will appear recorded onto the tape after this halt is read.

Each program of the Post machine that when applied to a number tuple of *any length* either leads to no result halt or produces a number as a result analogously defines a certain algorithm.

Let us denote numbers as N, the set of all the number tuples as N^∞, the set of all the number tuples of length k as N^k.

Now Post's proposal can, with the aid of the term "algorithm", be formulated as follows.

Let a certain class of initial data, N, N^k and N^∞ be given. Let each initial datum of this class be not placed in correspondence to anything or be placed to a certain resulting number (generally speaking, its own for each initial datum). We will regard two problems: (P) "Devise a program of the Post machine converting any initial datum for which the resulting number exists into this number and leading to no result for an initial datum for which the resulting number does not exist"; (A) derived from (P) by replacing the words "the program of the Post machine" by the word "algorithm". Then problems (P) and (A) are simultaneously either have or have no solutions.

So far we have not in any way substantiated Post's proposal but took it on trust. We will see now that it is,

to a certain extent, unavoidable. Post's proposal consists, in fact, of two assertions: (1) from solvability of problem (P) follows solvability of problem (A); (2) from solvability of problem (A) follows solvability of problem (P). The first assertion is obvious as the program being a solution of problem (P) alone forms an algorithm that is a solution of problem (A); we have mentioned it when analysing Example 3. As for the second assertion, the one that needs substantiation (and the one Post's proposal in essence boils down to), it was stated, in slightly other terms, by Post in his famous article "Finite Combinatory Processes—Formulation 1" as a "working hypothesis".

This "working hypothesis" that we will also call the Post hypothesis cannot be proved, at least with the idea underlying the word "proof" common in mathematics, and not at all because it is untrue, but because the notion of algorithm involved in it is not 'mathematically' defined. It should be recalled that on p. 67 we defined algorithm as "a uniform method of calculation common for the entire class of possible initial data". Other similar definitions are known:

an algorithm is "any system of calculations, performed by strictly determined rules, that, in a definite number of steps, is known to lead to the solution of the problem posed,"*

it is a "calculation process performed by a strict instruction and leading from variable initial data to the result sought for,"**

it is a "strict instruction determining the calculation process and leading from variable initial data to the result sought for,"***

"by an algorithm is understood every strict instruction, that specifies the computing process (called in this case algorithmic) beginning with an arbitrary initial datum (from a certain range of possible initial data for the given

---

* A. N. Kolmogorov, Algorithm. *Great Soviet Encyclopaedia*, 2nd ed., vol. 2, Moscow, 1950, p. 65 (in Russian).

** A. A. Markov, The Theory of Algorithms. *Proceedings of the Steklov Mathematics Institute of the USSR Acad. Sci.*, 38, Izd. Akad. Nauk SSSR, Moscow, 1951, p. 176 (in Russian).

*** A.A. Markov, The Theory of Algorithms. *Proceedings of the Steklov Mathematics Institute of the USSR Acad. Sci.*, 42, Izd. Akad. Nauk SSSR, Moscow, 1954, p. 3. (in Russian).

algorithm) and aimed at producing the result that is fully defined by this initial datum"*.

All these are not, of course, mathematical definitions but rather *descriptions* of the notion of algorithm. (In regard to the absence of an exhaustive "strict "definition the notion of algorithm resembles the notion of proof; the notion of proof, though, not only has no definition, neither has it a satisfactory description.**

The present-day advancement of mathematics does not permit to "prove rigorously" Post's hypothesis relying on such kind of algorithm descriptions.

The substantiation of Post's "working hypothesis" took some other path which is more traditional for a naturalist than for a mathematician, that is an experimental path supported by conceptial arguments. It is experimentally demonstrated that actually every time an algorithm is indicated it can be lent the form of the Post machine program leading to the same result. The conceptual argument—we will not present it here—corroborates that the views, held now by mathematicians, on the notion of the technique of calculation, that is an algorithm, do not enable one to synthesize an algorithm that could not be replaced by the Post machine program. Post himself saw the success of his ideas in changing the "working hypothesis" into a natural law. This can be thought to have come true.

In conclusion we will give another statement of the Post hypothesis. Let us consider any two algorithms whose ranges of possible initial data are identical. We will call

---

* V. A. Uspensky, Algorithm. *Great Soviet Encyclopaedia*, 3rd ed., vol. 1, 1978, p. 400 (in Russian).

** Now, in the latter half of the 20th century, as in the time of Euclid, proof still remains no more than a convincing argument that can convince us so that we are ready, using it, to convince others. Formal characterization of the notion of proof with the aid of mathematical logic, for all its importance, does not permit to avoid entirely using this notion in its intuitive sense just described. The great discovery made in the thirties by Kurt Gödel, an outstanding mathematician and logician of our time, consists in that the attempt to specify formally the notion of proof proves inevitably incomplete: the truths are found out that are proved by intuition but do not permit constructing the proof within the limits of specification made. (See V. A. Uspensky, "The Gödel incompleteness theorem in elementary presentation". *Advances in Mathematical Sciences*, 29, No. 1, 1974, pp. 3-47 (in Russian)).

such two algorithms *equivalent* as they are either both applicable or both inapplicable to any initial datum of the common set of these data; and if applicable, they give the same result. The notion of equivalence enables us to state Post's hypothesis as follows: each algorithm whose result are numbers and the set of possible initial data are N, N^k or N^∞ is equivalent to an algorithm with the same range of possible initial data defined by a certain Post machine program (defined as it was being accounted for on pp. 68 and 69).

Sec. 4.4. Additional Comments on Post's Hypothesis (Post's Thesis and Post's Principle)

The aim of this section is to formulate certain statements which are corollaries or variations of Post's hypothesis and to place the hypothesis among other similar hypotheses of the theory of algorithms. This section requires, however, a little better background than the previous and the next ones, at least, being acquainted with the general notions of a set, a subset, a function.

1. Our first comment will be about functions whose values can be computed on the Post machine. We limit ourselves, for simplicity, to the functions, whose arguments and values are nonnegative integers. Such functions will be referred to as numerical. Numerical function k of arguments will be treated as a function of a number tuple of length k. We will not here distinguish between the set of nonnegative integers N and set of N^1 number tuples of length 1. The function defined on arbitrary subset M of set N^k will be said to be defined in N^k. Function f defined in N^k is called computable if such an algorithm exists that (1) is applicable to any number tuple $\langle a_1, \ldots, a_k\rangle$ entering in the domain of function f and converts this tuple into $f(a_1, \ldots, a_k)$; (2) is inapplicable to any tuple not entering in the domain of function f. The above notions are of quite a general character. Let us now introduce more particular notions concerning computability on the Post machine. Numerical function f defined in N^k will be referred to as $\alpha\beta$-computable on the Post machine (or simply $\alpha\beta$-computable) as there is a Post machine program possessing the following property: if arbitrary number tuple $\langle a_1, \ldots, a_k\rangle$ (and nothing else is recorded onto the tape, the carriage is positioned against the leftmost labelled cell and the machine is started, then a result halt will occur if f is defined on a tuple $\langle a_1, \ldots, a_k\rangle$; in this case, after the halt, number $f(a_1, \ldots, a_k)$ (and this number only) will appear to be recorded onto the tape. On the strength of Post's hypothesis every computable numerical function is $\alpha\beta$-computable on the Post machine (the reverse is evident in view of that the Post machine program itself is an algorithm). It is no use trying to prove this statement (at the level of rigour common in mathematics) without making use of Post's hypothesis. For reasons that will be given below the statement that every computable function is $\alpha\beta$-computable is naturally called *Post's thesis.*

2. The restrictions on the computability of functions on the Post machine can be made stronger. Function f defined in N^k will be called ββ-computable on the Post machine (or, in brief, simply ββ-computable) as there is a program for the Post machine possessing the following property: if arbitrary number tuple $\langle a_1, \ldots, a_k \rangle$ (and nothing else) is recorded onto the tape, the carriage is positioned *anywhere* and the machine is started, then a result halt will occur if and only if f is defined on tuple $\langle a_1, \ldots, a_k \rangle$ and in this latter case, after the halt, the record of number $f(a_1, \ldots, a_k)$ (and only this number) will appear on the tape. Every ββ-computable function is, obviously αβ-computable. Since the notions of an αβ-computable function and of a ββ-computable function are strictly defined we can try to prove (at the level of rigour common in mathematics) tho identity of these notions. Such an attempt is successful: we *can prove* that every αβ-computable function is ββ-computable and, hence, due to Post's proposal every computable numerical function is ββ-computable.

The definitions of αβ-computability and ββ-computability differ in the restrictions imposed on the position of the carriage at the beginning of computations. The position of the carriage at the end of computations can likewise be of interest. Let us add on to the definitions of αβ- and ββ-computability after the words "(and only this number) will appear on the tape" the phrase: "the carriage facing the leftmost cell of this record." We will get the definitions of αα-computability and βα-computability accordingly. We can prove that both the class of βα-computable functions and the class of ββ-computable functions are identical with the class of αβ-computable functions.

To prove the identity of all the four classes is not difficult. It is even more simple to prove that the definitions differing only in restrictions imposed on the carriage position at the end are equivalent. To this end, we must turn to the program performing the search for an array on the tape (as it was pointed out in Sec. 3.3, such a program had, as a matter of fact, been built) and change it into the program searching not only for the array but its leftmost cell. A little more cumbersome is the proof of equivalence of definitions differing in the initial position of the carriage; here we have to devise the program enabling us to find the leftmost labelled cell of this record as the tuple is recorded onto the tape.

3. Thus, we have four definitions of the same family of numerical functions. The functions of this family will be, naturally, referred to as computable on the Post machine or, shorter, *computable according to Post*. Post's thesis can now be stated like this: every computable function is computable according to Post.

Here we face the following general situation: a certain accurately defined class of numerical functions is given which is specified by a definite mathematical definition (the class of functions computable according to Post in the given case); it is asserted that every computable (in the intuitive sense) function belongs to this class. This kind of assertion was first made by A. Church in 1936 and was called *Church's thesis*. True, Church did not examine all the computable functions but total functions alone (defined everywhere on the respective interval N^k). Church's thesis has stated that every function of this kind is general recursive. The notion of a general

recursive function had, however, "a rigorous mathematical" definition. Later S. Kleene, having generalized this thesis, declared that any computable function (not only a total function) is a partial recursive function*. This statement referred to by Kleene himself as a thesis is naturally called *Kleene's thesis* or *Kleene-Church's thesis***. The term "thesis" should be naturally applied to all the statements of the type considered. The assertion that every computable function is computable according to Post is, therefore, called Post's thesis by us.

None of these theses can be proved (in the usual mathematical meaning of the word) as they are an attempt to replace an intuitive notion by a formal one; but admitting one of them, we can derive any other.

4. Let an arbitrary set and two arbitrary algorithms be given. We say that these two algorithms are *equivalent* relative to this set as soon as they are both applicable or both inapplicable to any element of this set, and if applicable, lead to the same result. For instance, two algorithms that have a common range X of possible initial data and are equivalent in the sense of Sec. 3 can be said to be equivalent relative to X. The algorithms of conversion of numbers and number tuples specified by the Post machine programs will be called *Post's algorithms*. In the previous chapters we came more than once across a few programs existing for the solution of the same problem; each such program led to Post's algorithm of its own, nevertheless, all these algorithms were equivalent relative to the set of initial data contemplated in the problem under consideration.

We can suggest the following statement of Post's hypothesis which is equivalent to the previous one: let X be one of the sets N, N^k, N^∞ and let be given such an algorithm (with an arbitrary range of possible initial data) that the result of its application to any element of X, if this result does exist, is an integer; then this algorithm is equivalent, relative to X, to certain Post algorithm.

It is clear that having admitted the new statement of Post's hypothesis, we derive the earlier one as a corollary. That the new statement can be derived from the earlier one is substantiated by a more detailed analysis of the notion of algorithm which is beyond the scope of this book.

5. Now we will pass on to another slightly more general conception of Post's algorithm and, hence, more general statement of Post's proposal. Let us begin with agreeing on what objects we will consider possible initial data for Post's algorithms in a new,

---

* The numerical function is called partial recursive if it can be derived starting from $y = x + 1$ and $y = 0$ by performing, in any number, the following operations enabling one to derive new functions from those already derived: (1) to substitute functions into the function; (2) to define a new function by induction; (3) to define the function implicitly. For more details see "Recursive Functions" in *Great Soviet Encyclopaedia*, 2nd ed., vol. 36 or 3rd ed., vol. 21 (in Russian).

** For the statement of Church's and Kleene's theses see S. C. Kleene, *Introduction to Metamathematics*, D. Van Nostrand Company, Inc., Princeton, N.Y., 1952.

broader sense (before we have regarded only numbers and number tuples as initial data).

We will consider all possible chains made up of digits 0 and 1 : 01001101, 1, 001000, 1001, and so on.

All such chains—a so-called "empty" chain containing no digits at all is also included here—are also referred to as "words in {0, 1} alphabet". We will pick out the chains that begin with and end in 1; such chains will be called *Post's words*. Only the second and the fourth word in {0, 1} alphabet out of the four given above are Post's. The set of all Post's words will be denoted as P.

Let us agree to represent Post's word on the tape by labelled (for 1) and blank (for 0) cells. The word 10011 is, for instance, represented as shown in Fig. 38.

If, in turn, there are only a finite number of labelled cells on the tape, Post's word can be believed to be represented on it, between

Fig. 38. The Post word 10011 is represented here.

the leftmost and the rightmost labelled cell including these cells. It is natural to identify Post's words with their representations on the tape. The record of an arbitrary number or a number tuple is, in particular, Post's word: number 2 will be represented as 111, number tuple ⟨3, 2⟩ will be represented as 11110111, and 110101 is the representation of number tuple ⟨1, 0, 0⟩; it will be noted, too, that number 0 will be represented as 1 and number 1, as the word 11.

If the Post machine program is applied to the state in which only a finite number of cells is labelled (i.e. Post's word is represented on the tape) and leads to a result halt, then only a finite number of cells will again be labelled on the tape (i.e. a certain Post word will again be represented). Let us fix the initial position of the carriage relative to the representation of the initial Post word on the tape—for definiteness we will, at the beginning, position it against the leftmost labelled cell. Then the program, if it leads to a result halt at all, will lead from the tape with the initial Post word represented on it to the tape containing a certain new Post word. Hence, the program can be regarded to convert Post's words into Post's words, i.e. specify a certain *algorithm* of converting Post's words: it is every algorithm of this kind that we will call now Post's algorithm. Thus, possible initial data (as well as possible results) of arbitrary Post algorithm (in new broad sense) are Post's words; only numbers and number tuples or, in our present terms, Post's words in which no consecutive run of two zeroes occurs were previously admitted as possible initial data for Post's algorithms.

6. Many problems dealt with in the foregoing chapters can be interpreted in terms of Post's algorithms. (Many, but not all as, by virtue of our agreements, Post's algorithms presuppose the carriage positioned, at the beginning, against the leftmost labelled cell. Only the first of the problems in Chap. 2 can, therefore, be stated in the language of Post's algorithms.) The problem on addi-

tion of two numbers at the distance 1—problem (a) from Chap. 3—is, for instance, interpreted as a problem on synthesizing Post's algorithm converting any Post's word of the form

$$\underbrace{11\ldots1}_{(m_1+1)\text{ times}}\ 0\ \underbrace{11\ldots1}_{(m_2+1)\text{ times}}$$

into the word

$$\underbrace{1\ 1\ \ldots\ 1}_{(m_1+m_2+1)\text{ times}};$$

the problem on addition of two numbers at arbitrary distances—problem (c) from Chap. 3—as a problem on synthesizing Post's algorithm converting any Post word of the form

$$\underbrace{1\ 1\ \ldots\ 1}_{(m_1+1)\text{ times}}\ \underbrace{0\ 0\ \ldots\ 0}_{(l+1)\text{ times}}\ \underbrace{11\ldots1}_{(m_2+1)\text{ times}}$$

into word $\underbrace{1\ 1\ \ldots\ 1}_{(m_1+m_2+1)\text{ times}}$. For the given Post word, we will call any chain of consecutively running 1's of this word bordering on zeroes its array. Thus, there are three arrays (1, 111 and 111) in the word 101110111. The problem on addition of k numbers at arbitrary distances (with fixed k) — problem (d (k)) from Chap. 3—can be interpreted as a problem on synthesizing Post's algorithm converting any Post word, containing exactly k arrays, into the word made up of h unities where h is the number of unities in the initial word less by (k — 1). All these problem, as we saw, are solvable.

7. Is it possible to synthesize, for every algorithm performing conversions of Post's words, Post's algorithm equivalent to it (relative to P)? No, not for everyone. As we saw in Sec. 4.1, the problem on addition of arbitrary quantity of numbers recorded at arbitrary distances cannot, actually, be solved on the Post machine. It means, in our new terms, the following: there is no Post algoritbm converting any Post word into the word $\underbrace{1\ 1\ \ldots\ 1}_{[i-(k-1)]\text{ times}}$, where i is the number of unities, and k is the number of arrays in the initial word. At the same time it is clear that there is an algorithm (in the intuitive sense) performing such a conversion; thereby we arrive, as an example, at an algorithm which is equivalent to no Post algorithm. Other, simpler algorithms which are not equivalent to any Post algorithms can be cited as examples: such are, in particular, the algorithm calculating the overall number of unities in Post's word and the algorithm calculating the overall number of arrays.

8. In Sec. 4.1 there was pointed out the reason why for certain simple, one would think, problems there were no corresponding Post algorithms. Speaking in terms of Post's words, the reason is too long chains of zeroes in these words. This reason is exhaustive in the following precise sense. We will denote by $P^{(n)}$ the set of all Post's words in each of which no more than n consecutively running zeroes occur. We will be concerned with algorithms whose possible initial data and possible results are Post's words. For

every algorithm of this kind and for every n there is a Post algorithm equivalent to the initial algorithm relative to $P^{(n)}$. We will call this assertion the *Post principle*. The statement of Post's proposal cited at the end of Sec. 4.3 is an application of this principle to algorithms of the particular kind for which the range of possible initial data is $P^{(1)}$ and the set of possible results, $P^{(0)}$. We can make the same comments on the possibility of "proving mathematically" the Post principle as we have made on Post's proposal: the Post principle as well as Post's proposal and Post's thesis is a natural law rather than a mathematical theorem.

There are known other similar assertions. They are all natural laws supported both by experience and speculative conceptions on the structure of conceivable algorithms. In each of these assertions class $\boldsymbol{B}$ of algorithms of its own is present which claims to manage with algorithms of $\boldsymbol{B}$. This class $\boldsymbol{B}$ is accurately defined (just as accurately as the class of the Post algorithms), the notion of an algorithm of class $\boldsymbol{B}$ is, therefore, said to be a refinement of a general notion of algorithm; here, however, the word "refinement" should not be attached more importance than it is attached in the given case. A number of such refinements, apart from the Post algorithms, are known: these are the Turing algorithms, the Markov normal algorithms, the Kolmogorov algorithms*. The natural law of its own similar to the Post principle holds for each of them. It is of great significance that these laws can all be derived one from another. So, once one of these laws is admitted, all the others can be proved then. This fact justifies once again the validity of these laws.

For one of these laws A. A. Markov suggested the name of the *normalization principle*. The term "principle" can be, naturally, applied to all such laws. Hence the name *Post's principle*.

Sec. 4.5. The Post Machine and Electronic Computers

What are the similarity of and the difference between the Post machine and the modern electronic computer?

First, we will be concerned with the similarity that is revealed at least in the following three aspects:

1. Just like in the Post machine, elementary, (i.e. further indivisible) information media can be singled out in the electronic computer (in the Post machine such monatomic media are cells of the tape): just like in the Post machine each elementary medium of this kind can be in one of the two states (blank or labelled in the Post machine); all the information stored in the machine at the mo-

---

* A. N. Kolmogorov, V. A. Uspensky, "On the definition of algorithm", *Advances in Mathematical Sciences*, 13, No. 4, 1958, pp. 3-28 (in Russian).

ment is presented as the distribution of these two states over the elementary media.

2. Just like for the Post machine, a certain limited set of elementary actions is specified for the electronic computer that can, just like the Post machine, perform an action from this set in one step.

3. Just like the Post machine, the electronic computer operates on the basis of a special instruction, i.e. a program that prescribes what elementary actions should be performed and in what order.

Hence, in order to record information in the machine, it is necessary to identify it with a certain combination of states of elementary media. In turn, in order to realize an algorithm in the machine it must be presented as a sequence of definite elementary actions. This is what we call the skill of a programmer. And the skill of a designer is, in particular, in supplying the machine with such a set of elementary actions that the algorithms intended for realization in this machine were easily breakable into these actions.

At the same time we are going to draw the reader's attention to the following essential features of electronic computers that the Post machine either absolutely lacks or does not show duly.

1. The information stored in the Post machine is arranged in the storage (i.e. the device intended for storing data; in the Post machine such a device is a tape) linearly. This implies that each cell of the tape has only two "neighbours" which the machine can process after the given cell has been processed. In an electronic computer one area or other of the storage can, generally speaking, have much more neighbouring areas, "neighbouring" in the sense that the machine can pass on to processing any of them after processing of the initial area. The benefit of such organization of the storage is obvious if only to reduce the number of steps in the machine operation—for in the Post machine the carriage, in order to pass from examining one cell on to examining another, is bound to move through all the cells between them. The storage of the electronic computer usually consists, speaking in more detail, of two devices: an "external" storage of great capacity, storing information linearly similar to the Post machine tape, and an "internal" storage of relatively little capacity

but with more extensive "linkages between neighbours", i.e. individual areas of this device.

2. In the case of the Post machine we have not spoken of how exactly the program performance proceeds. We can imagine, for instance, a man near the Post machine who is reading a program written on the paper and, following it, moves the carriage, prints and erases the label. It is essential to emphasize that in the electronic computer the program is performed *automatically*. The program for the electronic computer is specially coded and recorded so that it is easy for "perception" by the computer. Such an easy way of recording is most often realised as a system of holes punched on the tape (a punched tape) or on the cards (punched cards). After the punched tape (or the punched cards) with the program has been made the electronic computer controlled by this punched tape (or punched cards) operates without human intervention. (Of course, it is not difficult to produce a device providing for automatic, without human intervention, operation of the Post machine.) In a first approximation the operation of an electronic computer can be compared with that of a mechanical piano in which keys are depressed in a sequence precisely determined by a program recorded onto the punched tape and it is performed without the participation of man.

Note. The electronic computer and a mechanical piano are, however, analogous only in a first approximation. As a matter of fact, the operation of a mechanical piano differs but little from that of a street-organ. In a street-organ projections and recesses on the roller open, on rotation, the passages of sounding pipes; each opening (i.e. the opening of the passage of the given pipe at the given moment) is controlled by its own projection or recess on the roller surface; the number of sounds in a melody is equal to the number of irregularities: the longer is the melody, the bigger should be the roller. In a mechanical piano each depression of a key is caused by a certain section of a punched tape and each such a section causes only one depression of only one key; hence, the longer is the melody, the longer should be the program recorded onto a punched tape. Roller record in a street-organ (by means of projections and recesses on its surface) and tape record in a mechanical piano (by means of holes) are, in some way, a note

record. The electronic computer is another matter. Here, as in the Post machine, the same instruction of a program can be executed *many times*. The number of steps of the machine operation is, therefore, independent of the program length: the same program can supply any number of steps which is necessary for achieving the result. (For instance, the program devised as a solution of problem 2 from Chap. 2 is of length 4, while the number of steps needed to produce the result required in this problem depends on the position of the carriage at the beginning.) This property is of great importance and makes it possible to realize algorithms, namely, to build *unified programs* leading to a result for the whole class of possible initial data. This property results from the possibility of going over from executing one instruction to executing any other, whereas in a mechanical piano or a street-organ an "instruction" to depress the key or to open the sounding passages can only be followed by an immediate "instruction" recorded on the roller or on the tape.

3. The most essential, fundamental difference of the electronic computer from the Post machine consists in the following. The initial datum (for instance, the tuple of numbers that are to be added) is recorded in the storage, onto the tape of the Post machine before the operation began. The program that must be applied to this initial datum and by which the machine operates is sort of being aside. In the electronic computer the program is put in the storage together with the initial datum before the operation began. (This is, by the way, another difference between the electronic computer and a mechanical piano: in a mechanical piano the program recorded onto the punched tape is "read" by a special device and the playing takes place as the reading goes on; so, this device plays the part of a pianist who plays from the notes laid out before him. In the electronic machine the punched tape with the program and the initial datum recorded onto it is processed, before the operation began, by a special device, i.e. an "input", that reads the information on the punched tape and puts it in the storage; in this respect the machine resembles the pianist who first learns the notes and then plays from memory without looking at the notes.) We will call what is contained in the storage of the machine at a certain moment (before the operation began) an initial

information. In the Post machine the initial information consists of an initial datum alone. In the electronic computer the initial information consists of an initial datum and a program. It is important to emphasize that the initial information is divided into an initial datum and a program only conventionally in the electronic computer. All the information, including the portion which is called "a program", can undergo various conversions in the process of machine operation. The possibility of changing the instructions themselves during operation, which the Post machine does not display, is a very significant feature of modern electronic computers. On the other hand, what is contained in any area of the storage can more or less often be interpreted as an instruction. That is why the division of the information contained in the storage into a program and an initial datum, in general, makes sense only at the initial moment; afterwards at every step the whole information is processed.

The question can naturally be asked: by what law is the information contained in the storage processed? This processing is, in turn, based on a certain rule that we will call the Universal Program. The Universal Program cannot, in any event, be confused with a particular program which is meant for the solution of one or other specific problem and is applicable to the initial data. The particular program depends on the problem being solved while the Universal Program is common for all the problems; in practical electronic computers it is provided for by the machine design (such machines are, therefore, often called *universal*). Let us sum up what has been said. In the Post machine an initial datum is put in the storage, whereupon the machine operates by the program which is not contained in the storage. In the storage of the electronic computers there is put an initial information, at the beginning, that is made up of a particular program of solving the given problem and an initial datum, after this the operation (namely, processing of the storage contents) proceeds by the Universal Program realized in the machine design.*

* The Universal Program can, though, be devised for the Post machine too, namely, each Post machine program can be coded as a Post word and then this program (i.e. the corresponding word) can be recorded onto the tape next to the initial datum to which it

Another, one would think, very essential feature distinguishing an electronic computer from the Post machine can be indicated: the tape, that serves as the storage in the Post machine, is of an infinite capacity which is impossible in practical computers. Nevertheless, even in the Post machine an infinite tape can be replaced by a finite tape that can be pasted on as the necessity arises (for only a finite length of the tape will just the same be used up by every moment of the Post machine operation): when the carriage reaches the end of the tape another few cells are pasted on to it. On the other hand, the capacity of an external storage of the electronic computer can also, in principle, grow infinitely by adding on new areas to it (say, new magnetic tapes). So, both the Post machine and the electronic computer can be treated as having a storage which is finite at each given moment but of infinitely growing capacity. It is this circumstance making these machines fit for realizing any algorithm on them that mainly makes related the Post machine and the electronic computer and makes it possible to regard the Post machine as a simplified model of the electronic computer.

SUPPLEMENT

As a supplement we present Emil Post's article "Finite combinatory processes—formulation 1". This article was published in the quarterly *The Journal of Symbolic Logic**, in September, 1936. It was furnished with the following editorial note: "Received October 7, 1936. The reader should compare an article by A. M. Turing "On computable numbers" shortly forthcoming in the *Proceedings of the London Mathematical Society*. The

---

should apply. Further, the Universal Program can be built that, being applied to the record which consists of the record of a certain program P and the record of a certain initial datum x, would lead to the same result as the direct application of P to the record x. Still, in this case the Universal Program will be merely one of the possible Post machine programs; in case of the electronic computer, however, the Universal Program is not at all one of the particular programs admissible for the electronic computer but is realized by the very design of a computer.

* *The Journal of Symbolic Logic.*, vol. 1, No. 3, 1936.

present article, however, although bearing a later date, was written, entirely independently of Turing's." Turing's article appeared in 1936 too.*

---

* A. M. Turing "On computable numbers, with an application to the Entscheidungsproblem", *Proc. London Math. Soc.*, ser. 2, **42**, Nos. 3, 4, 1936, pp. 230-265. A correction—*ibid.*, **43**, No. 7, 1937, pp. 544-546.

FINITE COMBINATORY PROCESSES— FORMULATION 1

Emil L. Post

The present formulation should prove significant in the development of symbolic logic along the lines of Gödel's theorem on the incompleteness of symbolic logics[1] and Church's results concerning absolutely unsolvable problems.[2]

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.

In the following formulation of such a solution two concepts are involved: that of a *symbol space* in which the work leading from problem to answer is to be carried out,[3] and a fixed unalterable *set of directions** which will both direct operations in the symbol space and determine the order in which those directions are to be applied.

In the present formulation the symbol space is to consist of a two way infinite sequence of spaces or boxes,** i.e., ordinally similar to the series of integers ..., —3, —2, —1, 0, 1, 2, 3, ... The problem solver or worker is to move and work in this symbol space, being capable of being in, and operating in but one box at a time. And apart from the presence of the worker, a box is to admit of but two possible conditions, i.e., being empty or unmarked, and having a single mark*** in it, say, a vertical stroke.

One box is to be singled out and called the starting point. We now further assume that a specific problem is

---

[1] Kurt Gödel, Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I, *Monatshefte für Mathematik und Physik*, vol. 38, 1931, pp. 173-198.

[2] Alonzo Church, "An unsolvable problem of elementary number theory", *American Journal of Mathematics*, vol. 58, 1936, pp. 345-363.

[3] Symbol space, and time.

* The term now in current use is a *set of instructions.—Ed.*

** These are referred to as *cells* in this work.—*Ed.*

*** The term commonly used now is a *label.—Ed.*

to be given in symbolic form by a finite number of boxes being marked with a stroke. Likewise the answer is to be given in symbolic form by such a configuration of marked boxes. To be specific, the answer is to be the configuration of marked boxes left at the conclusion of the solving process.

The worker is assumed to be capable of performing the following primitive acts: [4]

(a) *Marking the box he is in (assumed empty),*

(b) *Erasing the mark in the box he is in (assumed marked),*

(c) *Moving to the box on his right,*

(d) *Moving to the box on his left,*

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

The set of directions which, be it noted, is the same for all specific problems and thus corresponds to the general problem, is to be of the following form. It is to be headed:

Start at the starting point and follow direction 1. It is then to consist of a finite number of directions to be numbered 1, 2, 3, . . ., n. The ith direction is then to have one of the following forms:

(A) *Perform operation* O_i [O_i = (a), (b), (c), or (d)] *and then follow direction* j_i,

(B) *Perform operation* (e) *and according as the answer is yes or no correspondingly follow direction* j_i' or j_i'',

(C) *Stop.*

Clearly but one direction need be of type (C). Note also that the state of the symbol space directly affects the process only through directions of type (B).

A set of directions will be said to be *applicable* to a given general problem if in its application to each specific problem it never orders operation (a) when the box the worker is in is marked, or (b) when it is unmarked.[5] A set of directions applicable to a general problem sets up a deterministic process when applied to each specific problem. This process will terminate when and only when it comes to the direction of type (C). The set of directions will then be said to set up a *finite 1-process* in connection with the

---

[4] As well as otherwise following the directions described below.

[5] While our formulation of the set of directions could easily have been so framed that applicability would immediately be assured it seems undesirable to do so for a variety of reasons.

general problem if it is applicable to the problem and *if the process it determines terminates for each specific problem.* A finite 1-process associated with a general problem will be said to be a *1-solution* of the problem if the answer it thus yields for each specific problem is always correct.

We do not concern ourselves here with how the configuration of marked boxes corresponding to a specific problem and that corresponding to its answer, symbolize the meaningful problem and answer. In fact the above assumes the specific problem to be given in symbolized form by an outside agency and, presumably, the symbolic answer likewise to be received. A more self-contained development ensues as follows. The general problem clearly consists of at most an enumerable infinity of specific problems. We need not consider the finite case. Imagine then a one-to-one correspondence set up between the class of positive integers and the class of specific problems. We can, rather arbitrary, represent the positive integer n by marking the first n boxes to the right of the starting point. The general problem will then be said to be *1-given* if a finite 1-process is set up which, when applied to the class of positive integers as thus symbolized, yields in one-to-one fashion the class of specific problems constituting the general problem. It is convenient further to assume that when the general problem is thus 1-given each specific process at its termination leaves the worker at the starting point. If then a general problem is 1-given and 1-solved, with some obvious changes we can combine the two sets of directions to yield a finite 1-process which gives the answer to each specific problem when the latter is merely given by its number in symbolic form.

With some modification the above formulation is also applicable to symbolic logics. We do not now have a class of specific problems but a single initial finite marking of the symbol space to symbolize the primitive formal assertions of the logic. On the other hand, there will now be no direction of type (C). Consequently assuming applicability, a deterministic process will be set up which is *unending.* We further assume that in the course of this process certain recognizable symbol groups, i.e., finite sequences of marked and unmarked boxes, will appear which are not further altered in the course of the process. These will be the derived assertions of the logic. Of

course the set of directions corresponds to the deductive processes of the logic. The logic may then be said to be *1-generated.*

An alternative procedure, less in keeping, however, with the spirit of symbolic logic, would be to set up a finite 1-process which would yield the nth theorem or formal assertion of the logic given n, again symbolized as above.

Our initial concept of a given specific problem involves a difficulty which should be mentioned. To wit, if an outside agency gives the initial finite marking of the symbol space there is no way for us to determine, for example, which is the first and which the last marked box. This difficulty is completely avoided when the general problem is 1-given. It has also been successfully avoided whenever a finite 1-process has been set up. In practice the meaningful specific problems would be so symbolized that the bounds of such a symbolization would be recognizable by characteristic groups of marked and unmarked boxes.

The root of our difficulty however, probably lies in our assumption of an infinite symbol space. In the present formulation the boxes are conceptually at least physical entities, e.g., contiguous squares. Our outside agency could no more give us an infinite number of these boxes than he could mark an infinity of them assumed given. If then he presents us with the specific problem in a finite strip of such a symbol space the difficulty vanishes. Of course this would require an extension of the primitive operations to allow for the necessary extension of the given finite symbol space as the process proceeds. A final version of a formulation of the present type would therefore also set up directions for generating the symbol space.[6]

---

[6] The development of formulation 1 tends in its initial stages to be rather tricky. As this is not in keeping with the spirit of such a formulation the definitive form of this formulation may relinquish some of its present simplicity to achieve greater flexibility. Having more than one way of marking a box is one possibility. The desired naturalness of development may perhaps better be achieved by allowing a finite number, perhaps two, of physical objects to serve as pointers, which the worker can identify and move from box to box.

The writer expects the present formulation to turn out to be logically equivalent to recursiveness in the sense of the Gödel-Church development[7]. Its purpose, however, is not only to present a system of a certain logical potency but also, in its restricted field, of psychological fidelity. In the latter sense wider and wider formulations are contemplated. On the other hand, our aim will be to show that all such are logically reducible to formulation 1. We offer this conclusion at the present moment as a *working hypothesis.* And to our mind such is Church's identification of effective calculability with recursiveness[8]. Out of this hypothesis, and because of its apparent contradiction to all mathematical development starting with Cantor's proof of the non-enumerability of the points of a line, independently flows a Gödel-Church development. The success of the above program would, for us, change this hypothesis not so much to a definition or to an axiom but to a *natural law.* Only so, it seems to the writer, can Gödel's theorem concerning the incompleteness of symbolic logics of a certain general type and Church's results on the recursive unsolvability of certain problems be transformed into conclusions concerning all symbolic logics and all methods of solvability.

College of the City of New York

---

[7] The comparison can perhaps most easily be made by defining a 1-function and proving the definition equivalent to that of recursive function. (See Church, loc. cit., p. 350). A 1-function $f(n)$ in the field of positive integers would be one for which a finite 1-process can be set up which for each positive integer n as problem would yield $f(n)$ as answer, n and $f(n)$ symbolized as above.

[8] Cf. Church, loc. cit., pp. 346, 356-358. Actually the work already done by Church and others carries this identification considerably beyond the working hypothesis stage. But to mask this identification under a definition hides the fact that a fundamental discovery in the limitations of the mathematicizing power of *Homo Sapiens* has been made and blinds us to the need of its continual verification.

Printed in the Union of Soviet Socialist Republics