

### 15.6 Post's Machine

A few months later than Turing, and quite independently, Emil Post proposed a machine for defining a process that could solve a general problem. Just like Turing, Post had been galvanised into action by the recent results due to Gödel and Church. But Post had also, for a long time, been interested in a problem which appeared to be a suitable candidate for undecidability.

In 1921, and while still a student at Princeton, Post became interested in a class of problems of which the following is an example [13]. Being given a finite sequence made up of the symbols 0 and 1, examine the first term: if it is 0, remove the first three terms and add 00 to the end; if it is 1, remove the first three terms and add 1101 to the end. Then begin again. If we start with the sequence 10010, we obtain, successively:

```

10010
101101
1011101
11011101
111011101
0111011101
101110100
1101001101
10011011101 etc...

```

The sequences get longer, but from the 17th stage we find:

```

011011101110100
01110111010000
1011101000000
1101000001101
10000011011101
0000110111011101
011011101110100

```

and we have the same sequence as the 17th sequence, in other words, the process is periodic.

Is this the same for all sequences of 0's and 1's? Post mentions this problem in a paper in 1943 writing: "The little progress that has been made in the solution of such problems makes them candidates for undecidability". Post's problem has been generalised to what is called the *tag system*, which can be represented by a kind of small Turing machine [13].

In 1947, Post proved the undecidability of the problem, called today *Post's Correspondence Problem*, which can be stated as follows. Let U and V be two lists of k non-empty words from an alphabet A,  $U = \{u_1, u_2, \dots, u_k\}$  and  $V = \{v_1, v_2, \dots, v_k\}$ , does there exist a sequence of integers  $i_1, i_2, \dots, i_n$  such that the words  $u_{i_1} u_{i_2} \dots u_{i_n}$  and  $v_{i_1} v_{i_2} \dots v_{i_n}$  are the same? For example, consider:

```

A = {a, b}
U = {u1 = b, u2 = babb, u3 = ba}
V = {v1 = bbb, v2 = ba, v3 = a}

```

then there is a solution  $u_2 u_1 u_1 u_3 = v_2 v_1 v_1 v_3 = babbbbba$  and so the answer here is positive. However, the Post correspondence problem turns out to be undecidable, that is there is no algorithm, given any arbitrary U, V, for deciding the existence of such a solution sequence [9]. The undecidability of this problem is used to prove the undecidability of many other problems.

At the beginning of his 1936 paper, Post defines a 1-process, that is a sequence of instructions for directing a worker carrying out actions in a symbolic space. At the end of the paper, Post considers a number of fundamental questions. Is his idea equivalent to Church's effective calculability? Post does not prove this equality, but he remarks that Church's conception does itself need to be justified: does it correspond to the intuitive idea of calculability, that is an algorithm? Post's conclusion is that, for the moment, the equivalence of his conception and all other formulations should be accepted as a "working hypothesis".

**E. L. Post**  
 Finite Combinatory Processes, Formulation 1,  
*The Journal of Symbolic Logic*, vol. 1 (1936), 103-105.

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<sup>1</sup> and Church's results concerning absolutely unsolvable problems.<sup>2</sup>

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<sup>3</sup>, 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  $\dots, -3, -2, -1, 0, 1, 2, 3, \dots$ . 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 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:<sup>4</sup>

- Marking the box he is in (assumed empty),
- Erasing the mark in the box he is in (assumed marked),
- Moving to the box on his right,
- Moving to the box on his left,
- 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 *i*th 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.<sup>5</sup> 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 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.

[...]

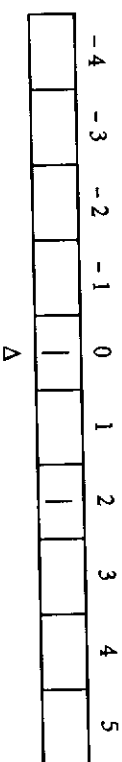
The writer expects the present formulation to turn out to be logically equivalent to recursiveness in the sense of the Gödel-Church development.<sup>6</sup> Its purpose, however, is not only to present a system of a certain logical potency but also, in its restricted field, of psychological fidelity. 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.<sup>7</sup> 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.

<sup>1</sup> Kurt Gödel, *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I*, *Monatshefte für Mathematik und Physik*, vol. 38 (1931), 173-198.  
<sup>2</sup> Alonzo Church, *An unsolvable problem of elementary number theory*, *American Journal of Mathematics*, vol. 58 (1936), 345-363.  
<sup>3</sup> Symbol space, and time.  
<sup>4</sup> As well as otherwise following the directions described below.

<sup>5</sup> 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. 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.  
<sup>7</sup> 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.

Post does not speak of a machine although his formulation, perhaps to a greater extent than Turing's resembles our concept of a computer. What Post calls a box can be compared to a memory which can be either empty or contain a value. Furthermore, his set of instructions can be compared to a sort of program with branches. But it is a minimal formulation of what we would call a program machine together with an operating program. In fact, just one symbol can be entered in the memory, and this can be moved only to a contiguous memory. It is certainly appropriate to describe this minimal formulation as "Formulation 1".

Using a vertical bar for the symbol, a configuration of boxes, where the box being visited is labelled 0, would look like this:



An instruction is preceded by a label, which is simply its number, and is terminated by a jump to another instruction indicated by its label. We can consider four types of instruction.

- First type: marking (a) or erasing (b). Marking would be written:  
 1: mark ; goto 3 ;  
 which means that the first instruction is to mark the square with | and then jump to instruction 3. Erasing would be an instruction like:  
 2: erase ; goto 1 ;  
 meaning that the second instruction is to erase the mark in the box being visited and then jump to instruction 1.

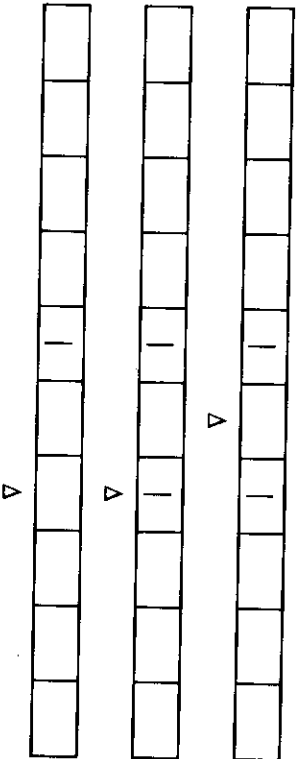
- Second type: movement to the right (c) or left (d) in the symbolic space. These instructions would be, for example:  
 3 : right ; goto 5 ;  
 4 : left ; goto 2 ;

- Third type: conditional jump. A conditional jump would be written:
  - 5 : if marked then 3 if not 2 ;
 meaning the fifth instruction is to examine the contents of the box. If the box is marked, jump to instruction 3, if it is not marked, jump to instruction 2.
- Fourth type: end instruction, which can be written:
  - 6 : stop.

Consider, now the following set of instructions:

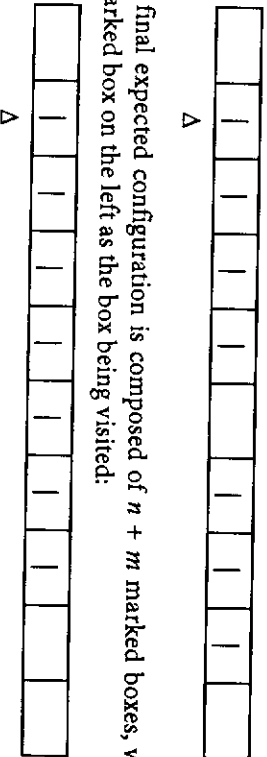
- 1 : right ; goto 2 ;
- 2 : if marked then 3 if not 1 ;
- 3 : erase ; goto 4 ;
- 4 : stop .

Starting with the configuration given above, we would have the sequence:

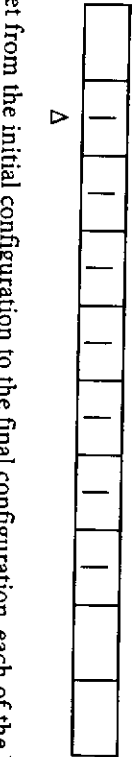


The sequence of the labels of the instructions that are carried out is: 1, 2, 1, 2, 3, 4. This program has the effect of erasing the contents of the first box to the right that contains an entry.

Let us now look at what Post calls a "general problem", that of adding strictly positive integers, and try to find a "1-solution". What is needed, then is to find a "1-process", that is a set of instructions which terminates for each "specific problem", that is to say for each two given integers  $n$  and  $m$ , it gives as solution the integer  $n + m$ . The initial state could be represented by a configuration of boxes of which there are  $n$  marked boxes, followed by an empty box, and then  $m$  marked boxes. We suppose that the box being visited is the first marked box at the left. For  $n = 4$  and  $m = 3$ , the configuration would look like this:



The final expected configuration is composed of  $n + m$  marked boxes, with the first marked box on the left as the box being visited:



To get from the initial configuration to the final configuration, each of the bars on the right has to slid one box to the left, and the marker has to return to the first of the marked boxes. This can be done by the following set of instructions:

- 1: if marked then 2 if not 3 ;
- 2: right ; goto 1 ;
- 3: mark ; goto 4 ;
- 4: if marked then 5 if not 6 ;
- 5: right ; goto 4 ;
- 6: left ; goto 7 ;
- 7: erase ; goto 9 ;
- 8: if marked then 9 if not 10 ;
- 9: left ; goto 8 ;
- 10: stop.

For the "specific problem" considered, the sequence of labels of the instructions is: 1, 2, 1, 2, 1, 2, 1, 3, 4, 5, 4, 5, 4, 6, 7, 9, 8, 9, 8, 9, 8, 9, 8, 9, 8, 9, 8, 10.

Post concludes his paper by touching on the question of the equivalence of his system with the definitions of recursive functions given by Gödel and Church. He considers more generally all the definitions which could be made to translate the idea of process or calculation. His "working hypothesis" is that they are all equivalent to his formulation, which has the merit of psychological fidelity.

## 15.7 Conclusion

With the introduction of the concept of an algorithm, the history of algorithms changes into the history of a new field of science: the field of algorithms. Here, it is not a question of seeking an algorithm for solving a particular problem but the solution of problems posed by the general study of algorithms. This area of research has been developed, in particular, alongside the construction of computers and the invention of programming languages.

As we have seen, the origin of the idea of an algorithm lies in the question of the existence of an algorithm for the solution of a problem. But the field of algorithmic study is also concerned with the complexity of algorithms, that is the intuitive idea of the cost of an algorithm in the time taken for it to carry out a task (temporal complexity) and in the amount of memory required (spatial complexity). The temporal and spatial complexities of an algorithm may be defined, respectively, by the number of movements and the number of cells used by the corresponding Turing machine, as a function of the length of the input data.

The concept of programming languages, and writing algorithms in these languages, poses a number of questions related to syntax: the description of the syntax of a language, the non ambiguous nature of the syntax, verification that a program conforms to the syntax of the language, the writing of a compiler for interpreting and executing a program from the syntax of the language. All these problems were responsible for the construction of a number of concepts and theories: grammars, Chomsky classification, automata theory and theories of languages, etc.