5 Turing Machines and Computability

The English mathematician Alan M. Turing proposed in 1936 that a certain class of ideal machines, now called Turing machines, be considered as computing devices. Although these machines are of the utmost simplicity, it appears that any effectively computable function, no matter how complex it may be, can be computed by a machine in this class if that machine can use as much time and auxiliary storage as may be needed; this statement is known as Turing's thesis. In particular, we can define a certain machine in the class that can simulate any other machine in the class and thus, we believe, compute any effectively computable function. This machine, called the universal Turing machine, is the theoretical prototype of real general purpose computers.

This model of the computing process predated by several years the first large general purpose programmable computing machines, and it has had a profound impact on many areas of computer science. In fact, the fundamental idea of using a programming language that differs from the machine language of the computer which interprets that programming language appears, in essence, in Turing's work. The importance of his ideas may be judged in part by the fact that the annual award of the Association for Computing Machinery for outstanding contributions to computer science is named after Turing.

We see in this chapter that the class of functions that can be defined by Turing machines is the same as the class of partial recursive functions defined in the last chapter. Therefore, the two approaches are equally satisfactory in making precise the notion of an effectively computable function.

However, Turing's approach, with its machine orientation, provides a bette model for real computers than the more abstract mathematical framework o recursive function theory.

5.1 MOTIVATIONS UNDERLYING TURING MACHINES

In our discussion of the theory of recursive functions we took a very forma approach in giving a precise meaning to the vague notion of an effectivel computable function. There does not seem to be much of heuristic value i the schema we used to define the general recursive functions, and ther appears little that might have been motivated by our intuitive impressions o effective procedures. We shall now take another quite different approac toward defining sharply the class of effectively computable functions. Thi approach will involve the employment of certain highly simplified and idealized machines whose capabilities will be used to give meaning to the notio of effective computability. That is, we shall deal with processes that ar machine-like and that therefore bear some resemblance to our use of rea computers. However, the device that we shall employ, the Turing machine will be, in a sense, stripped down to its essentials. It will, it is true, be a computer of sorts but one vastly less efficient and, in compensation, les complex than a modern computer. Any manufacturer who would marke such a machine would fail in short order. In order to appreciate the motivation underlying the prescription of this device it is worth reviewing briefl the notions brought out by Turing in his fundamental paper of 1936, "On Computable Numbers With an Application to the Entscheidungsproblem" [5], where he formulates the device that today we call a Turing machine. We shall see that he was guided in part by certain rudimentary impressions of human mental processes.

Turing considered the process of computation as performed by a human and by stripping away the superfluous and extraneous by, in a sense, distilling the situation to its core, he arrived at the fundamental characteristics of these machines. Consider, using the analogy of Rogers [4], a man seated in a room and performing some prescribed computation, defined for given input data, using pencil and paper. Let us suppose that the room has a window in it to permit input information to be given to him on sheets of paper and to allow him to transmit his results in similar form. Assume that he has available to him an unlimited supply of scratch paper to use in the course of his calculations or, better—for the room would be rather crowded with this supposition—that additional paper will be brought to him whenever he needs it. If he accumulates such prodigious intermediate results during the course of the computation that they cannot be accommodated in the room, we shall store these partial results for him outside the room and furnish them to him when he needs them. Suppose that our human computer has been instructed beforehand about the procedure he is to execute when we provide him with input data. He, however, knows no mathematics beyond the most elementary arithmetic operations and knows no shortcuts or deviations from the prescriptions given him.

Let us analyze this situation and try to break it down into atomic steps. We first have the problem of providing input data to our human computer in camera. Consider the sheets of paper we pass through the window to him and that he uses in the course of his calculations. It is helpful, and in no way restrictive, in making our conceptions precise if we assume that he uses paper divided into squares, like graph paper, and that he always writes any character in one of the squares. Does this paper have to be two-dimensional or can we replace it by a one-dimensional equivalent? That is, can we assume a strip or tape divided into squares instead of the two-dimensional arrays of squares on the sheets? The feasibility of doing this is, perhaps, suggested by the fact that we can form an image on a TV screen by scanning the picture over horizontal, or other, lines. If our human computer has sufficient memory, he can then dispense with the two-dimensional format. To avoid requiring him to remember too much, let us permit him to jot down on scratch paper anything he needs to remember. He can then always refer back to this written record when demands on his memory become excessive.

Next, consider the kinds of operations he might perform in the course of the calculations, keeping in mind that he has memorized the procedure that he has been taught to apply. We shall, of course, allow him to write down a statement of this procedure if it is excessively long. Suppose, for example, that he must multiply two multi-digit numbers. In the course of doing this he must perform, say, the partial product 8×7. He reads the 8, he reads the 7 and knows that he must write down a 6, shift one digit position, add the carry of 5, and so forth. To use language that generalizes and at the same time somewhat simplifies this further, we might say that when he is in a certain mental *state*, namely, knowing that he is at a particular point in the multiplication process, he reads an 8 and a 7 and enters another mental *state*, that which dictates his writing a 6 and remembering a carry of 5. On occasion he may erase a character he has written and replace it by another. He may go back and forth on the paper and make reference to, or change, earlier obtained results. If he needs more paper, additional blank paper will be provided him.

These are the things he does—replaces symbols by symbols (including erasing since the blank is considered to be one of the possible symbols), moves to other parts of the paper and at different times invokes some one o a fixed number of small "subroutines." We consider this last operation to b that of entering some mental state. Some fixed number of symbols wi always suffice for all purposes and, for any particular algorithmic process some finite number of states of mind will be adequate. Turing gave a interesting, but perhaps not completely convincing, argument against a infinitude of such mental states. He wrote that "if we admitted an infinity o states of mind, some of them will be arbitrarily close and will be confused. He offered a similar reason for not allowing an infinite alphabet of sym bols–that some of them would be blurred if there were too many of them

Turing then constructs a machine that will do the work of our human computer by enabling it to perform the atomic steps implied by the above considerations. It is interesting to note that at about the same time (1936), and independently of Turing, the American mathematician Emil Post, also seeking to give a formal explication of effectiveness based on an understanding of what takes place in the human brain, arrived at a similar formulation of a machine ("Finite Combinatory Processes, Formulation I" [3]). Post's very brief description, however, is more suggestive than substantive and omits most of the details necessary to a complete description of the

230 A. M. Turing [Nov. 12,

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

[illegible]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. [illegible]

[illegible]

Fig. 5.1 Alan Turing and the first page of his fundamental paper of 1936. (Photo courtesy of Bassano & Vandyk Studios, London, England. Article reproduced by permission of the Council of The London Mathematical Society, from *Proceedings of the London Mathematical Society*, Vol. 42, 1937, [illegible]

behavior of the machine. Nevertheless, there are those who believe that the machine we discuss should be called a *Post-Turing* machine in recognition of Post's fundamental contributions. This is yet another of the innumerable examples in the sciences of several individuals arriving at about the same time and independently of each other at the same key idea leading to the solution of a significant theoretical problem.

5.2 ORGANIZATION OF A TURING MACHINE

We now describe the salient characteristics of a Turing machine. These will differ slightly from Turing's original specifications, conforming more closely to modern expositions of the theory.

1. The machine includes a potentially infinite, in both directions, linear tape divided into boxes and read by a read head that scans one box at a time. At any given time the tape is finite, but there is no automatic recognition of the physical ends of the tape. It is potentially infinite in the sense that we assume that a blank square will be appended to the tape whenever needed, when the tape would, otherwise, be pulled through the read head. This corresponds to our assumption that an unlimited supply of paper was available to our human computer.

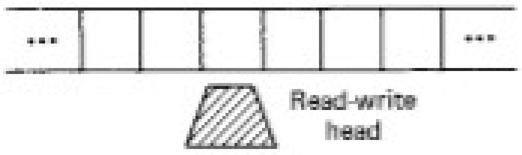

Fig. 5.2 Turing machine tape.

2. Symbols written on the tape are drawn from a *finite* alphabet: $S_0, S_1, S_2, \ldots, S_p$. Usually, S_0 is taken to be the blank B, and S_1 to be the unit 1. One of the basic theorems of the theory states that an alphabet with only *two* characters, B and 1, will suffice in designing a machine to compute any function. This theorem attests to the universality of binary machines; they are just as powerful as decimal machines. (N.B. In a binary machine the 0 bit is the same as the blank B.)

3. The control or processing unit of the machine (thinking of the tape as a peripheral attachment) can assume one of a *finite* number of distinct internal

states or configurations: $q_1, q_2, q_3, \ldots, q_m$. As in the case of the alphabet, i can be shown that we can always design a machine to do what we want tha employs only two states, but at the expense of an expanded alphabet. Th internal state of a machine completely determines how the machine re sponds to a symbol being read on the tape.

The "program" for a given machine is assumed to be built in an consists of a set of instructions in the form of *quintuples* which are define below. A clock is an integral part of the machine, and at each time step th machine executes that one of its quintuples which may apply to the state o the machine and the symbol being read. A particular Turing machine is then, to be thought of as being comparable to a real computer with a store program in its memory. Each machine defines one particular mathematica function on input data provided on its tape, rather than being able to compute an arbitrary function. We shall, however, later introduce a *universa* Turing machine which can receive as part of its input data a description o any other Turing machine and then, by simulating the behavior of tha machine, compute the same function as the machine it is simulating would do.

Each quintuple ($q_i S_j S_k R q_l$, $q_i S_j S_k L q_l$, or $q_i S_j S_k N q_l$) defines the action of the machine when it is in a particular state reading a particular character. Its format is interpreted as follows:

q_i	S_j	S_k	R, L, or N	q_l
The machine in this state,	reading this symbol on the tape,	will replace it by this symbol,	move to the square on the right (R), or left (L), or not at all (N),	then enter this state.

Here q_l may be the same as q_i and S_k the same as S_j. The behavior of the machine, beginning from some starting position on a given tape, is completely determined by its initial state and by the set of quintuples which are built in. Normally, we shall assume that the initial state of a Turing machine is q_1. With this understanding the machine *is*, in a sense, its set of quintuples, for these specify unequivocally its behavior. It is generally recognized in computing practice that software can be converted into hardware, and conversely. The interchangeability of the two is made striking in Turing's formulation, but there are those who believe that it would have been better if Turing had looked at a set of quintuples as a *program* rather than as a definition of a *machine*.

5.3 THE USE OF TURING MACHINES TO COMPUTE FUNCTIONS

The result of a machine's computation is obtained from the information on its tape if and when the machine stops. That the machine may not stop at all is easily seen. Suppose the two quintuples

$$q_1\, S_2\, S_2\, R\, q_3 \text{ and } q_3\, B\, B\, L\, q_1$$

both appear in the machine, and the read head is positioned as shown on the tape in Fig. 5.3 while the machine is in state q_1.

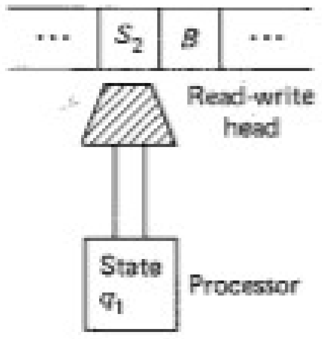

Figure 5.3

It is easily recognized that in this situation the machine will seesaw interminably between the two squares shown.

We have two choices in determining how the machine will stop. We can either introduce a special STOP, or quiescent, state q_ω which has no successor states under any conditions (i.e., no quintuple begins with q_ω), or we can say that the machine will stop when it is in state q_i scanning symbol S_j if it contains no quintuple beginning $q_i S_j$. We shall adopt the former convention. A little thought will show that the difference between the two conventions is insubstantial. We can always, with slight modifications, go from a machine that does not have a special stop state to a machine that employs one, as implied by one of the exercises.

There is one important restriction in writing down the quintuples of a machine to perform some task. We cannot include two quintuples which are contradictory in that they instruct the machine to do two different things under the same conditions. This would occur if we were to allow two different quintuples with the same two starting symbols $q_i S_j$. If, for example, the two quintuples

$$q_2\, S_3\, S_2\, R\, q_3 \text{ and } q_2\, S_3\, B\, L\, q_1$$

both appear in the machine, the situation will be ambiguous when the machine is in state q_2 scanning S_3 on its tape. If we allowed the machine to follow, in random fashion, one or the other of the two quintuples, it would not have the deterministic behavior that we expect of an effective procedure.

With these assumptions, then, a given Turing machine, i.e., a given set of quintuples, will act to transform in some manner a particular starting tape made up of characters drawn from its alphabet. Starting in the specified initial state, it defines a particular function on tapes with designated starting squares, mapping initial tapes into final tapes and final read head positions. The map or function is defined for a given starting tape and starting position if the machine eventually stops after starting on the tape. If it never stops because it enters an endless loop in moving over some finite segment of the tape, or because it extends without limit the tape, the map or function for that machine is undefined for that starting tape and starting position.

Consider, for example, the machine consisting of the two quintuples,

$$\{q_1\ 1\ 1\ R\ q_1,\ q_1\ B\ 1\ R\ q_2\},$$

and employing as alphabet the two characters $\{B, 1\}$. Assume that the machine starts in state q_1 and that q_2 is the STOP state.

It is clear that this machine will, when it starts on some particular 1 in a tape made up of blanks and 1's, move to the right to find the first blank. It will then change this blank to a 1, move one square to the right and stop.

On the other hand, the machine that uses the same alphabet and consists of

$$\{q_1\ 1\ 1\ R\ q_2,\ q_2\ B\ B\ L\ q_1,\ q_2\ 1\ 1\ L\ q_1\}$$

will, when it starts on a 1, go immediately into an endless loop moving back and forth between its starting square and the one to the right.

5.4 THE INSTANTANEOUS DESCRIPTION OF A TURING MACHINE

Consider the problem of interrupting the operation of a Turing machine when it is about to begin the execution of some particular quintuple and recording that information needed to resume the computation at some later time—that is, all the information that completely determines the subsequent course of the machine's actions. The analogous problem for real computers has been frequently studied in setting up backup and recovery procedures. These are of special importance when there is concern about the possible failure of a computer system during a very lengthy data processing task, or in situations where it is necessary to anticipate system interrupts to allow execution of a high priority job. In the case of Turing machines, the problem has a simple solution. All we need record at the given instant are (1) the internal state of the machine, (2) the list of symbols that are written on the tape, and (3) the identity of the tape square currently being scanned by the read head.

We can encode all of this information in a string of symbols as follows. Suppose the machine is in state q_3, that the information on the tape is $S_2 S_1 S_2 S_0 S_2 S_3 S_1$, and that the read head is positioned over the third square from the left. We represent all of this information by writing

$$S_2 S_1 q_3 S_2 S_0 S_2 S_3 S_1$$

The position of q_3, just before the symbol that is about to be read, indicates the location of the read head. Such a string of tape symbols together with one state symbol constitutes an "instantaneous description" of a Turing machine. If we omit the state symbol, the resulting string is the *tape description* alone.

If a machine eventually stops after starting in some initial instantaneous description, ID_i, we can consider the terminal instantaneous description, ID_t, to be the "result" of the computation. The machine effects a transformation, or mapping, from instantaneous descriptions into instantaneous descriptions: $\mathrm{ID}_i \rightarrow \mathrm{ID}_t$. In fact, every single quintuple determines a partial function on instantaneous descriptions. For example, the quintuple $q_2 S_2 S_3 R q_1$ causes the instantaneous description $S_3 S_4 S_0 q_2 S_2 S_1 S_1$ on a particular machine to be succeeded by $S_3 S_4 S_0 S_3 q_1 S_1 S_1$. A computation on a Turing machine proceeds as an eventually terminating sequence of instantaneous descriptions, as determined by its quintuples. If the machine does not eventually stop, no computation is defined.

A machine "snapshot," as this term is used on real machines, is analogous to the instantaneous description of a Turing machine. A snapshot includes all information of interest to the programmer in determining, at a designated point during a computation, what transformations the program has accomplished at that point in its execution. It includes much of the information that would be needed to resume the computation at that point.