Lecture notes on Turing machines

Ivano Ciardelli

1 Introduction

Turing machines, introduced by Alan Turing in 1936, are one of the earliest and perhaps the best known model of computation. The importance of this model is due to at least two factors.

1. Turing's analysis of computation is extremely fine-grained

In Turing's analysis, computation is reduced to a number of truly atomic steps, each of which involves no internal complexity, and can be mechanized in an obvious way. Compare this with unlimited register machines. In a URM computation, incrementing the content of a register n is a single computation step. At a given point in a computation, register n might contain a number, say 19. After the execution of instruction $S(n)$, the register contains the number 20. But how does this transition happen exactly?

Turing machines provide a more fine-grained analysis of the same process. First of all, in order to actually operate with numbers, numbers will have to be represented in some specific way in the memory—as strings of a specific alphabet. For instance, numbers could be represented in the usual decimal notation, where each cell in the memory holds one digit, and the beginning and end of the number's representation is marked by a symbol #. Thus, the representation of 19 in the memory is as follows:

...	#	1	9	#	...

In order to turn the representation of this number into the representation of its successor, namely

...	#	2	0	#	...

a number of manipulations on the memory cells are needed. We first need to update the rightmost digit, 9, to the "next" digit; then, since the result is 0, we need to move one cell to the left and increment the digit 1 as well; since the result this time is 2, and not 0, we can stop here (but if number we were incrementing had been, say, 99, we would have had to move the initial # symbol and make room for one more digit).

This illustrates how complex even the simplest operations are, when we are really concerned with the manipulation of actual representations of numbers.

Since Turing machines are concerned with such manipulations, writing Turing programs is usually very complicated. However, having an analysis at this fundamental level is important for two reasons:

1. by reducing computation to truly basic and mechanical steps, Turing showed most clearly how computations of all sorts could be implemented on artificial devices, inspiring the development of physical computers;

2. keeping track of how numbers are represented on memory is important to analyze the complexity of computations; for instance, comparing two large numbers, or transferring a large number to another part of the memory, has a higher computational cost than comparing or transferring small numbers; this is captured by Turing's view of computation, since numbers can only be operated on one digit at a time; but it is not captured, e.g., by the URM view of computation, since the operation of comparing or transferring a number is always regarded as a single computation step.

2. Turing machines arise from a general analysis of computation procedures

In his 1936 paper *On computable numbers*, Turing motivated the definition of the Turing machine by looking at what humans do when they perform a computation following an algorithm. Abstracting away from the inessential features of this process, he distilled the mathematical notion of a Turing machine. The strength of this approach is that it gives some *a priori* plausibility to the conjecture that anything which is computable in the intuitive sense is also computable by a Turing machine; this is because, if Turing is right, for a human to compute a given function would essentially amount to performing some (perhaps very complicated) procedure that could be formalized as a Turing program.

Notice that this important feature is not shared by the URM approach or by the recursive function approach. A priori, we have no reason to believe that anything that is computable can be computed using only the four kinds of instructions allowed in URM programs. And, similarly, we have no a priori reason to believe that any function which is computable can be obtained from basic recursive functions by means of composition, recursion, and minimalization.

2 From human computing to Turing machines

Turing's analysis of human computing

The starting point of Turing's analysis is the consideration of what a human computer (an accountant) does when she performs a computation with pencil and paper. The computer manipulates numbers written on a sheet of paper, which we can assume to be as large as required by the computation at hand. The numbers are represented on paper as sequences of symbols in some alphabet. We can assumed that the sheet of paper is squared, i.e., divided into cells, and

that throughout the computation each cell either contains a single symbol, or it is empty. At any given point in the computation, the computer can directly attend to only a part of the page: for simplicity, we may assume that it is only a single cell which is under her attention at any given time. However, the computer also has some memory of what she has done so far, as well as some idea of what she's about to do next. We can capture this by saying that at any point in the computation, the computer is in a certain *internal state*. Since the computer is a finite entity, only finitely many states are possible.

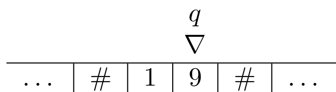
At any given point, the algorithm that the computer is executing determines what the computer should do next, on the basis of the content of the cell under consideration, and of the computer's mental state. The computer can perform three basic operations on the sheet: erase the content of the current cell; write down a symbol in the current cell, if this is empty; and turn her attention to a different cell. In addition, the computer will change her internal state through the computation depending on the symbols she sees in the paper, and on what the algorithm prescribes in response to them.

Turing machines: the idea

The analysis of human computation we just described is closely mirrored in the mathematical notion of a Turing machine—where such a machine represents a computer who is given the instructions to carry out a particular procedure.

A Turing machine operates on a space of cells, which for simplicity are arranged in a unidimensional tape, rather than in a bi-dimensional sheet. We assume that the tape is as long as needed, so that we never run out of tape in the course of a computation. We make this assumption because we want to analyze what can be computed in principle, abstracting away from concrete time and space limitations. At any point, a cell on the tape is either blank, or it contains a symbol from the given alphabet.[1]

The machine operates on the tape by means of a device called the machine's *head*. At any point, this head is placed on a specific cell, and it can be used to read and to modify the content of the cell. Moreover, at any given point the machine is in a certain internal state, which allows the machine to retain some trace of what has happened so far in the computation and which partly determines what is to be done next. Thus, the configuration of the machine at a given instant might be represented as follows, where ∇ denotes the machine head, and q stand for the current internal state of the machine.



The machine operates on the tape in accordance to its *program*, which consists of instructions of one of the following forms:

[1] Formally, it will be convenient to construe "blank" as a special symbol B of our alphabet.

- $\langle q, s, s', \leftarrow, q' \rangle$: if you are in state q and scanning a cell that contains s, replace s by s', move the head one cell left, and transition to state q';
- $\langle q, s, s', -, q' \rangle$: if you are in state q and scanning a cell that contains s, replace s by s', leave the head still, and transition to state q';
- $\langle q, s, s', \rightarrow, q' \rangle$: if you are in state q and scanning a cell that contains s, replace s by s', move the head one cell right, and transition to state q'.

We say that an instruction I is *applicable* at a certain point in the computation if the first two components of the instruction match the current state of the machine and the symbol contained in the cell which is currently being scanned. Thus, for instance, the instruction $\langle q, 9, 0, \leftarrow, q' \rangle$ is applicable when the machine is in the configuration drawn above. The program of the machine should be such that, at any given configuration, at most one instruction is applicable. If an applicable instruction is found, the machine executes it, which results in a new configuration being reached. For instance, executing the instruction $\langle q, 9, 0, \leftarrow, q' \rangle$ in the above situation leads to the following configuration:

		q' ∇			
...	#	1	0	#	...

To operate, a Turing machine needs to be set in its designated initial state q_0, with its head placed on a designated initial cell of a given tape. Starting in this configuration, the machine will then execute at each step the unique applicable instruction of the program, if such an instruction is found. If at some point no applicable instruction is found, the computation comes to an end and we say that the machine *halts* on the given tape. On the other hand, if at each step in the computation an applicable instruction is found, then the computation never comes to an end, and we say that the machine *doesn't halt* on the given tape.

3 Formal definitions

Definition 1 (Turing machine)**.** Let Σ be a finite set of symbols—our *alphabet*. A Turing machine over Σ is a triple $T = \langle Q, q_0, P \rangle$ where:

- Q is a finite set of *states* for the machine.
- $q_0 \in Q$ is a designated *initial state*.
- P, the *program* of the machine, is a set of instructions; each instruction is of the form $\langle q, s, s', \circ, q' \rangle$ with $q, q' \in Q$, $s, s' \in \Sigma$, and $\circ \in \{\leftarrow, -, \rightarrow\}$. This set of instructions is required to obey the following condition:

  Compatibility condition: for any state $q \in Q$ and symbol $s \in \Sigma$, there is at most one instruction $I \in P$ which has the form $I = \langle q, s, s', \circ, q' \rangle$.

Definition 2. Let σ be a tape, with a designated initial cell σ_0.[2] Let T be a Turing machine over the alphabet of the tape. If the computation of T on σ terminates, we write $T(\sigma)\downarrow$, and we denote by $T(\sigma)$ the tape as it is in the final configuration of the computation. If the computation of T on σ does not terminate, we write $T(\sigma)\uparrow$.

In order to regard a Turing machine as computing a function of natural numbers, we need to make specific conventions as to how numbers are going to be represented on the tape. Many conventions are possible here: for instance, we could represent numbers in decimal notation, or in binary notation. Here, we follow Turing in adopting the simplest possibility, namely, a *tally* notation: a number $n \in \mathbb{N}$ will be represented simply as a sequence of n occurrences of the symbol '1'. We mark the beginning and the end of the representation of a number by means of a special symbol '#'. This allows us to encode easily multiple numbers on the tape, as sequences of '1' separated by the symbol '#'.

Definition 3 (Tally coding of natural numbers)**.** We will encode natural numbers on tapes by using the alphabet $\{1, \#, B\}$, where 1 is a tally symbol, # is a separator symbol, and B is the blank.

– A single natural number n is encoded by the tape below, where n is the number of symbols 1 which occur between the two # symbols, and all the rest of the tape is blank. We denote this tape by $\ulcorner n \urcorner$.

$\ulcorner n \urcorner$:=

...	#	1	...	1	#	...

The designated initial cell of the tape is the one corresponding to the leftmost # symbol.

– A sequence $n_1, \ldots, n_k$ is encoded by the tape below, where the first and the second dividers are separated by n_1 occurrences of the symbol '1', the second and third by n_2 occurrences of occurrences of '1', etc.

...	#	1	...	1	#	...	#	1	...	1	#	...

As before, the rest of the tape is blank, and the designated initial cell corresponds to the leftmost occurrence of '#'. We denote this tape by $\ulcorner n_1, \ldots, n_k \urcorner$.

Definition 4 (Turing computable functions)**.**
Let $f : \mathbb{N}^k \to \mathbb{N}$ be a partial function. We say that a Turing machine T over the alphabet $\Sigma = \{1, \#, B\}$ computes f if, for any numbers $n_1, \ldots, n_k \in \mathbb{N}$:

[2] Formally, a tape can be modeled as a map $\sigma : \mathbb{Z} \to \Sigma$, where $\mathbb{Z}$ is the set of integers. We think of each integer as numbering a tape cell, and of $\sigma(i)$ as the symbol contained in cell number i. The designated initial cell is cell number 0.

– if $f(n_1, \ldots, n_k) = m$, then $T(\ulcorner n_1,\ldots,n_k \urcorner)\downarrow$ and $T(\ulcorner n_1,\ldots,n_k \urcorner) = \ulcorner m \urcorner$
– if $f(n_1, \ldots, n_k)\uparrow$, then $T(\ulcorner n_1, \ldots, n_k \urcorner)\uparrow$

We say that a function f is *Turing computable* (abbreviated as T-computable) if there exists a Turing machine that computes f.

Definition 5 (Turing decidable predicates)**.**
Let M be a predicate of natural numbers. We say that a Turing machine T over alphabet $\Sigma = \{1, \#, B\}$ decides M if T computes the characteristic function c_M. We say that M is *Turing decidable* (T-decidable for short) if there exists a Turing machine that decides M.

Example 1 (Successor)**.** The function $s(x) = x + 1$ is T-computable. A Turing machine that computes this function has states $Q = \{q_0, q_1, q_2\}$, initial state q_0, and the following instructions as its program:

– $\langle q_0, \#, \#, \rightarrow, q_1 \rangle$
– $\langle q_1, 1, 1, \rightarrow, q_1 \rangle$
– $\langle q_1, \#, 1, \rightarrow, q_2 \rangle$
– $\langle q_2, B, \#, -, q_2 \rangle$

Example 2 (Sum)**.** The function $\mathrm{sum}(x, y) = x + y$ is T-computable. A Turing machine that computes this function has states $Q = \{q_0, q_1, q_2, q_3\}$, initial state q_0, and the following instructions as its program:

– $\langle q_0, \#, \#, \rightarrow, q_1 \rangle$
– $\langle q_1, 1, 1, \rightarrow, q_1 \rangle$
– $\langle q_1, \#, 1, \rightarrow, q_2 \rangle$
– $\langle q_2, 1, 1, \rightarrow, q_2 \rangle$
– $\langle q_2, \#, B, \leftarrow, q_3 \rangle$
– $\langle q_3, 1, \#, -, q_3 \rangle$