

Chapter 1

The Anatomy of a Turing Machine

WE FOLLOW CHAITIN in adopting a slight variation on Turing's original machine.

Turing's model of an ideal computer was remarkably concise, largely because he used a single tape for input, output and 'scratch' calculations.

However, Turing's work showed—paradoxically perhaps—that almost any model with unlimited memory would serve as well as his own; at least, so long as *computability* was the only question at issue. Once one steps beyond that—which we shall not do—into the realms of complexity or efficiency, the equivalence of different models is by no means so clear.

For our purpose, Turing's model is *too* concise, since we need to distinguish more clearly between input and output.

The Chaitin model that we shall follow shares the following essential features of Turing's machine:

1. The machine 'moves' in discrete steps $0, 1, 2, \dots$, which we may call *moments*.
2. The 'working part' of the machine is a doubly infinite tape, divided into squares numbered by the integers: Each square contains one of the bits 0, 1. At any moment all but a finite number of the bits are 0.

Mathematically, the tape contents are defined by a map

$$t : \mathbb{Z} \rightarrow \{0, 1\},$$

where

$$t(i) = 0 \text{ if } |i| > N$$

for some N .

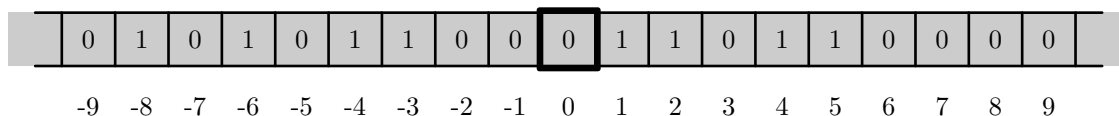


Figure 1.1: Tape with numbered squares

3. At each moment the machine is in one of a finite number of *states*

$$q_0, q_1, \dots, q_n.$$

State q_0 is both the initial and the final, or halting, state. The computation is complete if and when the machine re-enters this state.

We denote the set of states by

$$Q = \{q_0, q_1, \dots, q_n\}.$$

4. Both the *action* that the machine takes at each moment, and the *new state* that it moves into, depend entirely on 2 factors:

- (a) the state q of the machine;
- (b) the bit $t(0)$ on square 0 of the tape.

It follows that a machine T can be described by a single map

$$T : Q \times \{0, 1\} \rightarrow A \times Q,$$

where A denotes the set of possible actions.

To emphasize that at any moment the machine can only take account of the content of square 0, we imagine a ‘scanner’ sitting on the tape, through which we see this square.

Turning to the *differences* between the Chaitin and Turing models:

1. Turing assumes that the input x to the machine is written directly onto the tape; and whatever is written on the tape if and when the machine halts is the output $T(x)$.

In the Chaitin model, by contrast, the machine reads its input from an *input string*, and writes its output to an *output string*.

2. Correspondingly, the possible actions differ in the 2 models. In the Turing model there are just 4 actions:

$$A = \{\text{noop}, \text{swap}, \leftarrow, \rightarrow\}.$$

Each of these corresponds to a map $\mathcal{T} \rightarrow \mathcal{T}$ from the set \mathcal{T} of all tapes (ie all maps $\mathbb{Z} \rightarrow \{0, 1\}$) to itself. Thus **noop** leaves the content of square 0 unchanged (though of course the state may change at the same moment), **swap** changes the value of $t(0)$ from 0 to 1, or 1 to 0, while \leftarrow and \rightarrow correspond to the mutually inverse shifts

$$(\leftarrow t)(i) = t(i + 1), \quad (\rightarrow t)(i) = t(i - 1),$$

Informally, we can think of \leftarrow and \rightarrow as causing the scanner to move to the left or right along the tape.

In the Chaitin model we have to add 2 more possible actions:

$$A = \{\text{noop}, \text{swap}, \leftarrow, \rightarrow, \text{read}, \text{write}\}.$$

The action **read** takes the next bit b of the input string and sets $t(0) = b$, while **write** appends $t(0)$ to the output string. (Note that once the bit b has been read, it cannot be re-read; the bit is removed from the input string. Similarly, once **write** has appended a bit to the output string, this cannot be deleted or modified. We might say that the input string is a ‘read-only stack’, while the output string is a ‘write-only stack’.)

To summarise, while the Turing model has in a sense no input/output, the Chaitin model has input and output ‘ports’ through which it communicates with the outside world.

Why do we need this added complication? We are interested in the Turing machine as a tool for converting an input string into an output string. Unfortunately it is not clear how we translate the content of the tape into a string, or vice versa. We could specify that the string begins in square 0, but where does it end? How do we distinguish between 11 and 110?

We want to be able to measure the *length* of the input program or string p , since we have defined the entropy of s to be the length of the shortest p for which $T(p) = s$. While we could indeed define the length of an input tape in the Turing model to be (say) the distance between the first and last 1, this seems somewhat artificial. It would also mean that the input and output strings would have to end in 1’s; outputs 111 and 1110 would presumably be indistinguishable.

Having said all that, one must regret to some extent departing from the standard; and it could be that the extra work involved in basing Algorithmic Information Theory on the Turing model would be a small price to pay.

However that may be, for the future we shall use the Chaitin model exclusively; and henceforth we shall always use the term “Turing machine” to mean *the Chaitin model of the Turing machine*.

1.1 Formality

Definition 1.1. *We set*

$$\mathbb{B} = \{0, 1\}.$$

Thus \mathbb{B} is the set of possible *bits*.

We may sometimes identify \mathbb{B} with the 2-element field \mathcal{F}_2 , but we leave that for the future.

Definition 1.2. *A Turing machine T is defined by giving a finite set Q , and a map*

$$T : Q \times \mathbb{B} \rightarrow A \times Q,$$

where

$$A = \{\text{noop}, \text{swap}, \leftarrow, \rightarrow, \text{read}, \text{write}\}.$$

1.2 The Turing machine as map

We denote the set of all finite strings by \mathbb{S} .

Definition 1.3. *Suppose T is a Turing machine. Then we write*

$$T(p) = s,$$

where $p, s \in \mathbb{S}$, if the machine, when presented with the input string p , reads it in completely and halts, having written out the string s . If for a given p there is no s such that $T(p) = s$ then we say that $T(p)$ is undefined, and write

$$T(p) = \perp,$$

Remark. Note that T must read to the *end* of the string p , and then halt. Thus $T(p) = \perp$ (ie $T(p)$ is undefined) if the machine halts before reaching the end of p , if it tries to read beyond the end of p , or if it never halts, eg because it gets into a loop.

Example. As a very simple example, consider the 1-state machine T (we will generally ignore the starting state q_0 when counting the number of states) defined by the following rules:

$$\begin{aligned}(q_0, 0) &\mapsto (\text{read}, q_1) \\ (q_1, 0) &\mapsto (\text{write}, q_0) \\ (q_1, 1) &\mapsto (\text{read}, q_1)\end{aligned}$$

This machine reads from the input string until it reads a 0, at which point it outputs a 0 and halts. (Recall that q_0 is both the initial and the final, or halting, state; the machine halts if and when it re-enters state q_0 .)

Thus $T(p)$ is defined, with value 0, if and only if p is of the form

$$p = 1^n 0.$$

Remark. By convention, if no rule is given for (q_i, b) then we take the rule to be $(q_i, b) \rightarrow (\text{noop}, 0)$ (so that the machine halts).

In the above example, no rule is given for $(q_0, 1)$. However, this case can never arise; for the tape is blank when the machine starts, and if it re-enters state q_0 then it halts.

It is convenient to extend the set \mathbb{S} of strings by setting

$$\mathbb{S}^\perp = S \cup \{\perp\}.$$

By convention we set

$$T(\perp) = \perp$$

for any Turing machine T . As one might say, “garbage in, garbage out”.

With this convention, every Turing machine defines a map

$$T : \mathbb{S}^\perp \rightarrow \mathbb{S}^\perp.$$

1.3 The Church-Turing Thesis

Turing asserted that *any effective calculation can be implemented by a Turing machine*. (Actually, Turing spoke of the Logical Computing Machine, or LCM, the term ‘Turing machine’ not yet having been introduced.)

About the same time — in 1936 — Church put forward a similar thesis, couched in the rather esoteric language of the *lambda calculus*. The two ideas were rapidly shown to be equivalent, and the term ‘Church-Turing thesis’ was

coined, although from our point of view it would be more accurate to call it the Turing thesis.

The thesis is a philosophical, rather than a mathematical, proposition, since the term ‘effective’ is not — perhaps cannot be — defined. Turing explained on many occasions what he meant, generally taking humans as his model, eg “A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal machine.” or “The idea behind digital computers may be explained by saying that these machines are intended to carry out any operations which could be done by a human computer.”

The relevance of the Church-Turing thesis to us is that it gives us confidence— if we believe it—that any algorithmic procedure, eg the Euclidean algorithm for computing $\gcd(m, n)$, can be implemented by a Turing machine.

It also implies that a Turing machine cannot be ‘improved’ in any simple way, eg by using 2 tapes or even 100 tapes. Any function $T : \mathbb{S}^\perp \rightarrow \mathbb{S}^\perp$ that can be implemented with such a machine could equally well be implemented by a standard 1-tape Turing machine. What we shall later call the class of computable or Turing functions appears to have a natural boundary that is not easily breached.