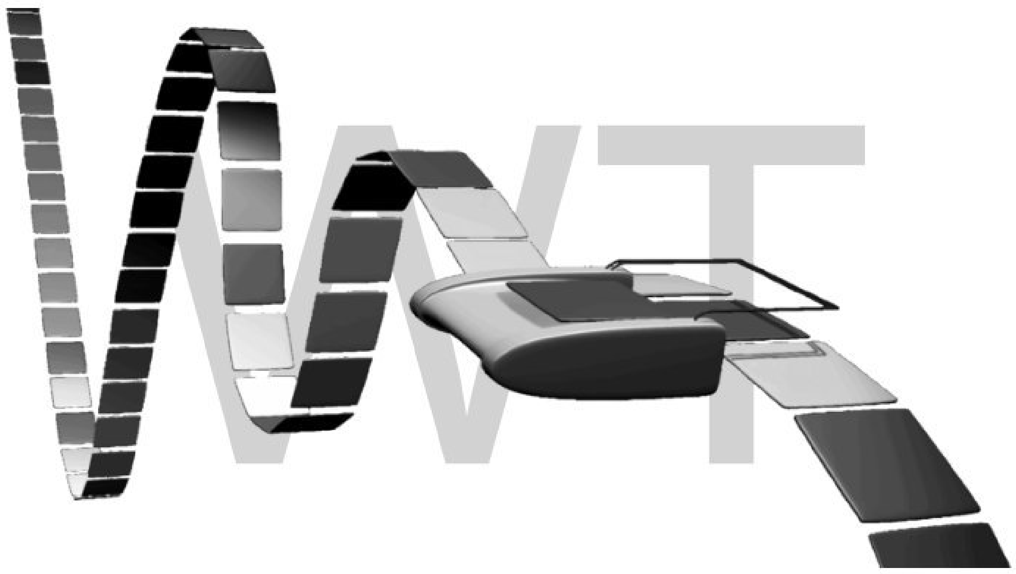
Chapter 1

Introduction to Turing Machine



An artistic representation of a Turing machine (Rules table not represented)

A **Turing machine** is a theoretical device that manipulates symbols on a strip of tape according to a table of rules. Despite its simplicity, a Turing machine can be adapted to simulate the logic of any computer algorithm, and is particularly useful in explaining the functions of a CPU inside a computer.

The "Turing" machine was described by Alan Turing in 1937, who called it an "*a*-(utomatic) machine". Turing machines are not intended as a practical computing technology, but rather as a thought experiment representing a computing machine. They help computer scientists understand the limits of mechanical computation.

Turing gave a succinct definition of the experiment in his 1948 essay, "Intelligent Machinery". Referring to his 1936 publication, Turing wrote that the Turing machine, here called a Logical Computing Machine, consisted of:

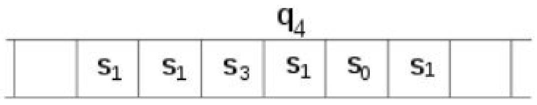
...an infinite memory capacity obtained in the form of an infinite tape marked out into squares, on each of which a symbol could be printed. At any moment there is one symbol in the machine; it is called the scanned symbol. The machine can alter the scanned symbol and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behavior of the machine. However, the tape can be moved back and forth through the machine, this being one of the elementary operations of the machine. Any symbol on the tape may therefore eventually have an innings. (Turing 1948, p. 61)

A Turing machine that is able to simulate any other Turing machine is called a universal Turing machine (**UTM**, or simply a **universal machine**). A more mathematically-oriented definition with a similar "universal" nature was introduced by Alonzo Church, whose work on lambda calculus intertwined with Turing's in a formal theory of computation known as the Church–Turing thesis. The thesis states that Turing machines indeed capture the informal notion of effective method in logic and mathematics, and provide a precise definition of an algorithm or 'mechanical procedure'.

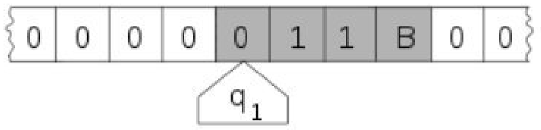
Studying their abstract properties yields many insights into computer science and complexity theory.

Informal description

The Turing machine mathematically models a machine that mechanically operates on a tape on which symbols are written which it can read and write one at a time using a tape head. Operation is fully determined by a finite set of elementary instructions such as "in state 42, if the symbol seen is 0, write a 1; if the symbol seen is 1, shift to the right, and change into state 17; in state 17, if the symbol seen is 0, write a 1 and change to state 6;" etc. In the original article ("On computable numbers, with an application to the Entscheidungsproblem"), Turing imagines not a mechanism, but a person whom he calls the "computer", who executes these deterministic mechanical rules slavishly (or as Turing puts it, "in a desultory manner").



The head is always over a particular square of the tape; only a finite stretch of squares is given. The instruction to be performed (q_4) is shown over the scanned square. (Drawing after Kleene (1952) p.375.)



Here, the internal state (q_1) is shown inside the head, and the illustration describes the tape as being infinite and pre-filled with "0", the symbol serving as blank. The system's full state (its *configuration*) consists of the internal state, the contents of the shaded squares including the blank scanned by the head ("11B"), and the position of the head. (Drawing after Minsky (1967) p. 121).

More precisely, a Turing machine consists of:

1. **TAPE** which is divided into cells, one next to the other. Each cell contains a symbol from some finite alphabet. The alphabet contains a special *blank* symbol (here written as 'B') and one or more other symbols. The tape is assumed to be arbitrarily extendable to the left and to the right, i.e., the Turing machine is always supplied with as much tape as it needs for its computation. Cells that have not been written to before are assumed to be filled with the blank symbol. In some models the tape has a left end marked with a special symbol; the tape extends or is indefinitely extensible to the right.
2. A **HEAD** that can read and write symbols on the tape and move the tape left and right one (and only one) cell at a time. In some models the head moves and the tape is stationary.
3. A finite **TABLE** ("action table", or *transition function*) of instructions (usually quintuples [5-tuples] : $q_i a_j \rightarrow q_{i1} a_{j1} d_k$, but sometimes 4-tuples) that, given the *state*(q_i) the machine is currently in *and* the *symbol*(a_j) it is reading on the tape (symbol currently under HEAD) tells the machine to do the following in sequence (for the 5-tuple models):
   - Either erase or write a symbol (instead of a_j written a_{j1}), *and then*
   - Move the head (which is described by d_k and can have values: 'L' for one step left *or* 'R' for one step right *or* 'N' for staying in the same place), *and then*
   - Assume the same or a *new state* as prescribed (go to state q_{i1}).

   In the 4-tuple models, erase or write a symbol (a_{j1}) and move the head left or right (d_k) are specified as separate instructions. Specifically, the TABLE tells the machine to (ia) erase or write a symbol *or* (ib) move the head left or right, *and then* (ii) assume the same or a new state as prescribed, but not both actions (ia) and (ib) in the same instruction. In some models, if there is no entry in the table for the current combination of symbol and state then the machine will halt; other models require all entries to be filled.

4. A **STATE REGISTER** that stores the state of the Turing table, one of finitely many. There is one special *start state* with which the state register is initialized. These states, writes Turing, replace the "state of mind" a person performing computations would ordinarily be in.

Note that every part of the machine—its state and symbol-collections—and its actions—printing, erasing and tape motion—is *finite*, *discrete* and *distinguishable*; it is the potentially unlimited amount of tape that gives it an unbounded amount of storage space.

Formal definition

Hopcroft and Ullman (1979, p. 148) formally define a (one-tape) Turing machine as a 7-tuple $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ where

- Q is a finite, non-empty set of *states*
- Γ is a finite, non-empty set of the *tape alphabet/symbols*
- $b \in \Gamma$ is the *blank symbol* (the only symbol allowed to occur on the tape infinitely often at any step during the computation)
- $\Sigma \subseteq \Gamma \setminus \{b\}$ is the set of *input symbols*
- $q_0 \in Q$ is the *initial state*
- $F \subseteq Q$ is the set of *final* or *accepting states*.
- $\delta : Q \setminus F \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is a partial function called the *transition function*, where L is left shift, R is right shift. (A relatively uncommon variant allows "no shift", say N, as a third element of the latter set.)

Anything that operates according to these specifications is a Turing machine.

The 7-tuple for the 3-state busy beaver looks like this :

Q = { **A**, **B**, **C**, **HALT** }
Γ = { 0, 1 }
b = 0 = "blank"
Σ = { 1 }
δ = see state-table below
q_0 = **A** = initial state
F = the one element set of final states {**HALT**}

Initially all tape cells are marked with 0.

State table for 3 state, 2 symbol busy beaver

Tape symbol	Current state A			Current state B			Current state C		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	1	R	**B**	1	L	**A**	1	L	**B**
1	1	L	**C**	1	R	**B**	1	R	**HALT**

Additional details required to visualize or implement Turing machines

In the words of van Emde Boas (1990), p. 6: "The set-theoretical object [his formal seven-tuple description similar to the above] provides only partial information on how the machine will behave and what its computations will look like."

For instance,

- There will need to be some decision on what the symbols actually look like, and a failproof way of reading and writing symbols indefinitely.
- The shift left and shift right operations may shift the tape head across the tape, but when actually building a Turing machine it is more practical to make the tape slide back and forth under the head instead.
- The tape can be finite, and automatically extended with blanks as needed (which is closest to the mathematical definition), but it is more common to think of it as stretching infinitely at both ends and being pre-filled with blanks except on the explicitly given finite fragment the tape head is on. (This is, of course, not implementable in practice.) The tape *cannot* be fixed in length, since that would not correspond to the given definition and would seriously limit the range of computations the machine can perform to those of a linear bounded automaton.

Alternative definitions

Definitions in literature sometimes differ slightly, to make arguments or proofs easier or clearer, but this is always done in such a way that the resulting machine has the same computational power. For example, changing the set $\{L,R\}$ to $\{L,R,N\}$, where N ("None" or "No-operation") would allow the machine to stay on the same tape cell instead of moving left or right, does not increase the machine's computational power.

The most common convention represents each "Turing instruction" in a "Turing table" by one of nine 5-tuples, per the convention of Turing/Davis (Turing (1936) in *Undecidable*, p. 126-127 and Davis (2000) p. 152):

(definition 1): $(\mathbf{q_i}, \mathbf{S_j}, \mathbf{S_k/E/N}, \mathbf{L/R/N}, \mathbf{q_m})$
(current state $\mathbf{q_i}$, symbol scanned $\mathbf{S_j}$, print symbol $\mathbf{S_k}$/erase **E**/none **N** , move_tape_one_square left **L**/right **R**/none **N** , new state $\mathbf{q_m}$ **)**

Other authors (Minsky (1967) p. 119, Hopcroft and Ullman (1979) p. 158, Stone (1972) p. 9) adopt a different convention, with new state $\mathbf{q_m}$ listed immediately after the scanned symbol S_j:

(definition 2): $(\mathbf{q_i}, \mathbf{S_j}, \mathbf{q_m}, \mathbf{S_k/E/N}, \mathbf{L/R/N})$
(current state $\mathbf{q_i}$, symbol scanned $\mathbf{S_j}$, new state $\mathbf{q_m}$, print symbol $\mathbf{S_k}$/erase **E**/none **N** , move_tape_one_square left **L**/right **R**/none **N)**

Example: state table for the 3-state 2-symbol busy beaver reduced to 5-tuples

Current state	Scanned symbol	Print symbol	Move tape	Final (i.e. next) state	5-tuples
A	0	1	R	**B**	(**A**, 0, 1, R, **B**)
A	1	1	L	**C**	(**A**, 1, 1, L, **C**)
B	0	1	L	**A**	(**B**, 0, 1, L, **A**)
B	1	1	R	**B**	(**B**, 1, 1, R, **B**)
C	0	1	L	**B**	(**C**, 0, 1, L, **B**)
C	1	1	N	**H**	(**C**, 1, 1, N, **H**)

In the following table, Turing's original model allowed only the first three lines that he called N1, N2, N3 (cf Turing in *Undecidable*, p. 126). He allowed for erasure of the "scanned square" by naming a 0th symbol S_0 = "erase" or "blank", etc. However, he did not allow for non-printing, so every instruction-line includes "print symbol S_k" or "erase" (cf footnote 12 in Post (1947), *Undecidable* p. 300). The abbreviations are Turing's (*Undecidable* p. 119). Subsequent to Turing's original paper in 1936–1937, machine-models have allowed all nine possible types of five-tuples:

	Current m-configuration (Turing state)	Tape symbol	Print-operation	Tape-motion	Final m-configuration (Turing state)	5-tuple	5-tuple comments	4-tuple
N1	q_i	S_j	Print(S_k)	Left L	q_m	(q_i, S_j, S_k, L, q_m)	"blank" = S_0, 1=S_1, etc.	
N2	q_i	S_j	Print(S_k)	Right R	q_m	(q_i, S_j, S_k, R, q_m)	"blank" = S_0, 1=S_1, etc.	
N3	q_i	S_j	Print(S_k)	None N	q_m	(q_i, S_j, S_k, N, q_m)	"blank" = S_0, 1=S_1, etc.	(q_i, S_j, S_k, q_m)
4	q_i	S_j	None N	Left L	q_m	(q_i, S_j, N, L, q_m)		(q_i, S_j, L, q_m)
5	q_i	S_j	None N	Right R	q_m	(q_i, S_j,		(q_i, S_j,

						N, R, q_m)		R, q_m)
6	q_i	S_j	None N	None N	q_m	(q_i, S_j, N, N, q_m)	Direct "jump"	(q_i, S_j, N, q_m)
7	q_i	S_j	Erase	Left L	q_m	(q_i, S_j, E, L, q_m)		
8	q_i	S_j	Erase	Right R	q_m	(q_i, S_j, E, R, q_m)		
9	q_i	S_j	Erase	None N	q_m	(q_i, S_j, E, N, q_m)		(q_i, S_j, E, q_m)

Any Turing table (list of instructions) can be constructed from the above nine 5-tuples. For technical reasons, the three non-printing or "N" instructions (4, 5, 6) can usually be dispensed with.

Less frequently the use of 4-tuples are encountered: these represent a further atomization of the Turing instructions (cf Post (1947), Boolos & Jeffrey (1974, 1999), Davis-Sigal-Weyuker (1994)).

The "state"

The word "state" used in context of Turing machines can be a source of confusion, as it can mean two things. Most commentators after Turing have used "state" to mean the name/designator of the current instruction to be performed—i.e. the contents of the state register. But Turing (1936) made a strong distinction between a record of what he called the machine's "m-configuration", (its internal state) and the machine's (or person's) "state of progress" through the computation - the current state of the total system. What Turing called "the state formula" includes both the current instruction and *all* the symbols on the tape:

Thus the state of progress of the computation at any stage is completely determined by the note of instructions and the symbols on the tape. That is, the **state of the system** may be described by a single expression (sequence of symbols) consisting of the symbols on the tape followed by Δ (which we suppose not to appear elsewhere) and then by the note of instructions. This expression is called the 'state formula'.
—*Undecidable*, p.139–140, emphasis added

Earlier in his paper Turing carried this even further: he gives an example where he places a symbol of the current "m-configuration"—the instruction's label—beneath the scanned square, together with all the symbols on the tape (*Undecidable*, p. 121); this he calls "the *complete configuration*" (*Undecidable*, p. 118). To print the "complete configuration" on one line he places the state-label/m-configuration to the *left* of the scanned symbol.

A variant of this is seen in Kleene (1952) where Kleene shows how to write the Gödel number of a machine's "situation": he places the "m-configuration" symbol q_4 over the scanned square in roughly the center of the 6 non-blank squares on the tape and puts it to the *right* of the scanned square. But Kleene refers to "q_4" itself as "the machine state" (Kleene, p. 374-375). Hopcroft and Ullman call this composite the "instantaneous description" and follow the Turing convention of putting the "current state" (instruction-label, m-configuration) to the *left* of the scanned symbol (p. 149).

Example: total state of 3-state 2-symbol busy beaver after 3 "moves" (taken from example "run" in the figure below):

1**A**1

This means: after three moves the tape has ... 000110000 ... on it, the head is scanning the right-most 1, and the state is **A**. Blanks (in this case represented by "0"s) can be part of the total state as shown here: **B**01 ; the tape has a single 1 on it, but the head is scanning the 0 ("blank") to its left and the state is **B**.

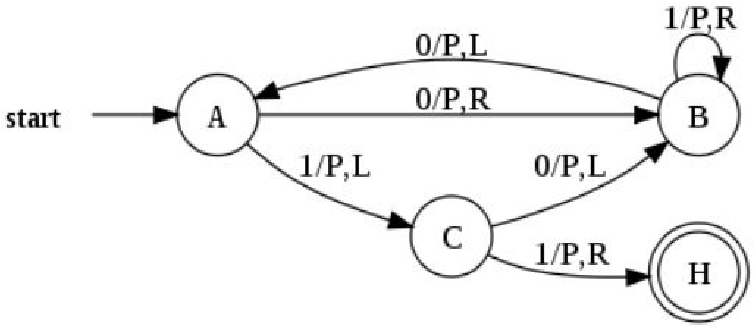
"State" in the context of Turing machines should be clarified as to which is being described: (*i*) the current instruction, or (*ii*) the list of symbols on the tape together with the current instruction, or (*iii*) the list of symbols on the tape together with the current instruction placed to the left of the scanned symbol or to the right of the scanned symbol.

Turing's biographer Andrew Hodges (1983: 107) has noted and discussed this confusion.

Turing machine "state" diagrams

The table for the 3-state busy beaver ("P" = print/write a "1")

Tape symbol	**Current state A**			**Current state B**			**Current state C**		
	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state	Write symbol	Move tape	Next state
0	P	R	**B**	P	L	**A**	P	L	**B**
1	P	L	**C**	P	R	**B**	P	R	**HALT**



The "3-state busy beaver" Turing Machine in a finite state representation. Each circle represents a "state" of the TABLE—an "m-configuration" or "instruction". "Direction" of a state *transition* is shown by an arrow. The label (e.g.. **0/P,R**) near the outgoing state (at the "tail" of the arrow) specifies the scanned symbol that causes a particular transition (e.g. **0**) followed by a slash /, followed by the subsequent "behaviors" of the machine, e.g. "**P P**rint" then move tape "**R R**ight". No general accepted format exists. The convention shown is after McClusky (1965), Booth (1965), Hill and Peterson (1974).

To the right: the above TABLE as expressed as a "state transition" diagram.

Usually large TABLES are better left as tables (Booth, p. 74). They are more readily simulated by computer in tabular form (Booth, p. 74). However, certain concepts—e.g. machines with "reset" states and machines with repeating patterns (cf Hill and Peterson p. 244ff)—can be more readily seen when viewed as a drawing.

Whether a drawing represents an improvement on its TABLE must be decided by the reader for the particular context.