

Turing Machines and Effective Computability

We introduce the most powerful of the automata we will study: Turing machines (TMs). TMs can compute any function normally considered computable; indeed we can define *computable* to mean computable by a TM.

Informal description of a Turing Machine

A (one-tape deterministic) TM consists of:

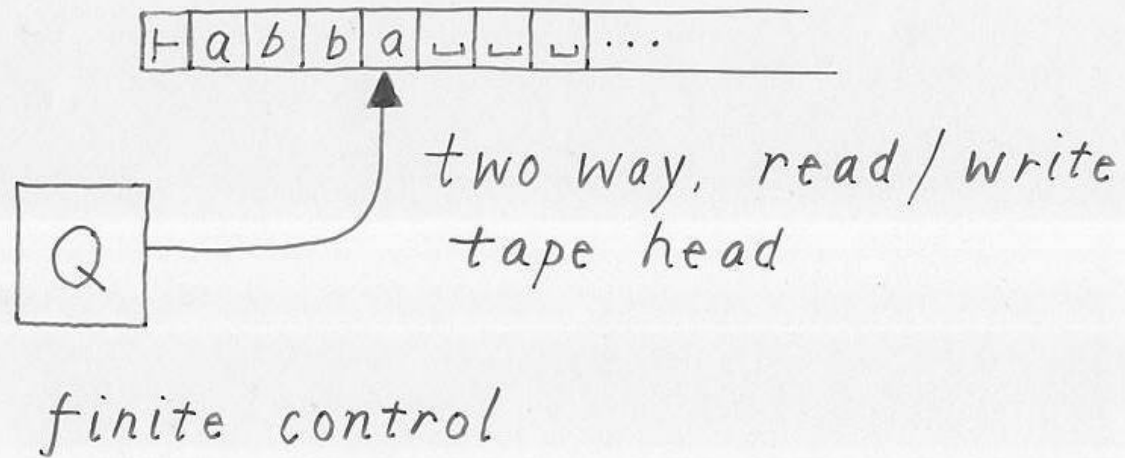
- a finite *input alphabet* Σ , a finite *tape alphabet* Γ such that $\Sigma \subseteq \Gamma$
- a finite set of *states* Q
- a *semi-infinite tape* of cells (infinite to the right)
- a *tape head* that can move left and right over the tape, reading and writing symbols onto tape cells

At the start of computation, contents of the tape are

$$\vdash w_1 \cdots w_n \sqcup \sqcup \sqcup \cdots$$

where $w = w_1 \cdots w_n$ is the input string, and $\vdash \in \Gamma$ is the left endmarker. The tape head is over \vdash , and the infinitely many cells to the right of the input all contain a special blank symbol $\sqcup \in \Gamma$.

Turing Machine



Workings of a Turing Machine

The machine starts in its start state with its head scanning the leftmost cell.

At each step, it reads the symbol under its head, and depending on that symbol and the current state, it writes a new symbol on that tape cell, then moves its head either left or right one cell, and enters a new state. The action is determined by a *transition function* δ .

It accepts its input by entering the accept state, and rejects by entering the reject state.

On some input, it may run infinitely without ever accepting or rejecting i.e. it *loops* on that input.

Note. Definitions of TMs in the literature differ slightly in nonessential details.

Here we use Kozen's definition.

Definition of a Turing Machine

A (one-tape deterministic) *Turing machine* (TM) is a 9-tuple

$$(Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$$

where

- Q is a finite set (the *states*)
- Σ is a finite set (the *input alphabet*)
- Γ is a finite set (the *tape alphabet*) containing Σ
- $\vdash \in \Gamma - \Sigma$, the *left endmarker*
- $\sqcup \in \Gamma - \Sigma$, the *blank symbol*
- $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is a finite function (the *transition function*)
- $q_0, q_{\text{acc}}, q_{\text{rej}} \in Q$ are respectively the *start*, *accept* and *reject* states, with $q_{\text{acc}} \neq q_{\text{rej}}$.

Some conventions

Intuitively $\delta(q, a) = (q', b, L)$ means “when in state q scanning symbol a , write b over the tape cell, move the head left by one cell, and enter state q' ”.

Restrictions

1. The left endmarker is never overwritten, and the machine never moves left of the endmarker. I.e. for all $p \in Q$ there exists $q \in Q$ such that

$$\delta(p, \vdash) = (q, \vdash, R).$$

2. Once the machine enters q_{acc} it never leaves it, and similarly for q_{rej} . I.e. for all $b \in \Gamma$, there exist $c, c' \in \Gamma$ and $D, D' \in \{L, R\}$ such that

$$\delta(q_{\text{acc}}, b) = (q_{\text{acc}}, c, D)$$

$$\delta(q_{\text{rej}}, b) = (q_{\text{rej}}, c', D')$$

Example: A TM that accepts $\{ a^n b^n c^n : n \geq 0 \}$ (not context-free)

Informal high-level description.

At start state, it scans right over the input string, checking that it matches $a^*b^*c^*$, and *writing nothing* (i.e. overwriting with same letter) on the way across. When it sees the first \sqcup , it overwrites it with a right endmarker \vdash .

Now it scans left, *erasing* the first c (i.e. overwriting it with \sqcup) it sees, then the first b it sees, then the first a it sees, until it reaches \vdash .

It then scans right, erasing one a , one b and one c . It continues to scan left and right, erasing one occurrence of each letter in one pass.

If in some pass, it sees at least one occurrence of one letter and no occurrence of another, it rejects. Otherwise it eventually erases all letters and makes one pass between \vdash and \vdash seeing only blanks, at which point it accepts.

Implementation-level description. Formally the TM is

$$(\underbrace{\{1, 2, \dots, 10, q_0, q_{\text{acc}}, q_{\text{rej}}\}}_Q, \underbrace{\{a, b, c\}}_\Sigma, \underbrace{\Sigma \cup \{\vdash, \sqcup, \dashv\}}_\Gamma, \vdash, \sqcup, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$$

where δ is

	\vdash	a	b	c	\sqcup	\dashv
q_0	(q_0, \vdash, R)	(q_0, a, R)	$(1, b, R)$	$(2, c, R)$	$(3, \dashv, L)$	—
1	—	$(q_{\text{rej}}, -, -)$	$(1, b, R)$	$(2, c, R)$	$(3, \dashv, L)$	—
2	—	$(q_{\text{rej}}, -, -)$	$(q_{\text{rej}}, -, -)$	$(2, c, R)$	$(3, \dashv, L)$	—
3	$(q_{\text{acc}}, -, -)$	$(q_{\text{rej}}, -, -)$	$(q_{\text{rej}}, -, -)$	$(4, \sqcup, L)$	$(3, \sqcup, L)$	—
4	$(q_{\text{rej}}, -, -)$	$(q_{\text{rej}}, -, -)$	$(5, \sqcup, L)$	$(4, c, L)$	$(4, \sqcup, L)$	—
5	$(q_{\text{rej}}, -, -)$	$(6, \sqcup, L)$	$(5, b, L)$	—	$(5, \sqcup, L)$	—
6	$(7, \vdash, R)$	$(6, a, L)$	—	—	$(6, \sqcup, L)$	—
7	—	$(8, \sqcup, R)$	$(r, -, -)$	$(r, -, -)$	$(7, \sqcup, R)$	$(q_{\text{acc}}, -, -)$
8	—	$(8, a, R)$	$(9, \sqcup, R)$	$(r, -, -)$	$(8, \sqcup, R)$	$(q_{\text{rej}}, -, -)$
9	—	—	$(9, b, R)$	$(10, \sqcup, R)$	$(9, \sqcup, R)$	$(q_{\text{rej}}, -, -)$
10	—	—	—	$(10, c, R)$	$(10, \sqcup, R)$	$(3, \dashv, L)$

E.g. Input tape at start is $\vdash a b c \sqcup \sqcup \dots$.

An accepting run:

$(\epsilon, q_0, \vdash a b c)$
 $(\vdash, q_0, a b c)$
 $(\vdash a, q_0, b c)$
 $(\vdash a b, 1, c)$
 $(\vdash a b c, 2, \sqcup)$
 $(\vdash a b, 3, c \dashv)$
 $(\vdash a, 4, b \sqcup \dashv)$
 $(\vdash, 5, a \sqcup \sqcup \dashv)$
 $(\epsilon, 6, \vdash \sqcup \sqcup \sqcup \dashv)$
 $(\vdash, 7, \sqcup \sqcup \sqcup \dashv)$
 $(\vdash \sqcup, 7, \sqcup \sqcup \dashv)$
 $(\vdash \sqcup \sqcup, 7, \sqcup \dashv)$
 $(\vdash \sqcup \sqcup \sqcup, 7, \dashv)$
 $(\vdash \sqcup \sqcup \sqcup, q_{acc}, \dashv)$

Configurations

As a TM computes, changes occur in the current state, the current tape contents, and the current head position. A description of these items is called a *configuration* of the TM, often represented as a triple

$$(u, q, v)$$

for the configuration where the current state is q , current tape content is $u v$ and the current head location is over the first symbol of v .

Let C and C' be configurations. We define the *next-configuration relation* \rightarrow (we read $C \rightarrow C'$ as “ C *yields* C' ”, meaning that the TM can legally go from C to C' in one step) formally as:

- $(ua, q, bv) \rightarrow (u, q', acv)$ if $\delta(q, b) = (q', c, L)$
- $(ua, q, bv) \rightarrow (uac, q', v)$ if $\delta(q, b) = (q', c, R)$

Special case: If the head is at the right end of the configuration, (ua, q, ϵ) is equivalent to (ua, q, \sqcup) .

We define $\xrightarrow{*}$, the reflexive, transitive closure of \rightarrow , inductively:

- $C \xrightarrow{0} C$, for all configurations C
- $C \xrightarrow{n+1} C''$ if $C \xrightarrow{n} C'$ and $C' \rightarrow C''$

We define $C \xrightarrow{*} C'$ (read “ C can yield C' in finitely many steps”) to be $C \xrightarrow{n} C'$ for some $n \geq 0$.

The language accepted by a TM

The *start configuration* of TM M on input w is $(\epsilon, q_0, \vdash w)$.

An *accepting configuration* is any configuration with state q_{acc} ; a *rejecting configuration* is any configuration with state q_{rej} .

Turing machine M is said to *accept* input w just in case $(\epsilon, q_0, \vdash w) \xrightarrow{*} C$ where C is some accepting configuration.

The *language* of M , written $L(M)$, is defined to be the collection of strings that M accepts.

Call a language *recursively enumerable* (or simply *r.e.*), if some TM accepts it.

When we start a TM M on an input, three outcomes are possible: M may accept, reject or loop (i.e. never terminate). Call M *total* (or a *decider*) if it halts on all inputs i.e. it never loops.

Call a language *decidable* (or *recursive*) if some total Turing machine accepts it.

Example: A TM that accepts $\{ ww : w \in \{ a, b \}^* \}$

This is a non-context-free but decidable language.

High-level description: $\Gamma = \{ a, b, \vdash, \dashv, \sqcup, \grave{a}, \grave{b}, \acute{a}, \acute{b} \}$

Two stages:

Stage I: Marking.

On input x , it scans out to the first blank symbol, making sure that x is of even length, and rejecting immediately if not. It then lays down a right endmarker \dashv , and repeatedly scans back and forth over the input.

In each pass from right to left, it marks the first unmarked a or b it sees with $\acute{}$ (i.e. overwriting a and b by \acute{a} and \acute{b} respectively). In each pass from left to right, it marks the first unmarked a or b it sees with $\grave{}$. It continues until all symbols are marked.

The reason: so that the centre of the string can be identified.

E.g. initially the tape contents are

$\vdash \quad a \quad a \quad b \quad b \quad a \quad a \quad a \quad b \quad b \quad a \quad \sqcup \quad \sqcup \quad \sqcup \quad \dots$

At the end of the marking stage:

$\vdash \quad \grave{a} \quad \grave{a} \quad \grave{b} \quad \grave{b} \quad \grave{a} \quad \acute{a} \quad \acute{a} \quad \acute{b} \quad \acute{b} \quad \acute{a} \quad \dashv \quad \sqcup \quad \sqcup \quad \dots$

Stage II: Erasing.

Then it repeatedly scans left and right: In each pass from the left, it erases the first symbol it sees marked with $\grave{}$ but remembers that symbol. It then scans forward until it sees the first symbol marked with $\acute{}$, checks that it is the same, and erases it, otherwise it rejects.

When it has erased all symbols, it accepts.

Stage II of the example:

⊢	<i>à</i>	<i>à</i>	<i>ḃ</i>	<i>ḃ</i>	<i>à</i>	<i>á</i>	<i>á</i>	<i>ḃ</i>	<i>ḃ</i>	<i>á</i>	⊢	⊐	⊐	...
⊢	⊐	<i>à</i>	<i>ḃ</i>	<i>ḃ</i>	<i>à</i>	⊐	<i>á</i>	<i>ḃ</i>	<i>ḃ</i>	<i>á</i>	⊢	⊐	⊐	...
⊢	⊐	⊐	<i>ḃ</i>	<i>ḃ</i>	<i>à</i>	⊐	⊐	<i>ḃ</i>	<i>ḃ</i>	<i>á</i>	⊢	⊐	⊐	...
⊢	⊐	⊐	<i>ḃ</i>	<i>ḃ</i>	<i>à</i>	⊐	⊐	<i>ḃ</i>	<i>ḃ</i>	<i>á</i>	⊢	⊐	⊐	...
⊢	⊐	⊐	⊐	<i>ḃ</i>	<i>à</i>	⊐	⊐	⊐	<i>ḃ</i>	<i>á</i>	⊢	⊐	⊐	...
⊢	⊐	⊐	⊐	⊐	<i>à</i>	⊐	⊐	⊐	⊐	<i>á</i>	⊢	⊐	⊐	...
⊢	⊐	⊐	⊐	⊐	⊐	⊐	⊐	⊐	⊐	⊐	⊢	⊐	⊐	...

Non-deterministic Turing Machines (NDTM)

At any point of a computation, a NDTM may proceed in one of several possible ways, so that the transition function has type

$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

The computation of a NDTM is a tree, whose branches correspond to different possible runs of the machine.

If any branch leads to the accept state, the machine is said to accept its input.

Lemma. A language is acceptable by some TM iff it is acceptable by some NDTM.

Deterministic Multitape TMs

k -tape TMs have k semi-infinite tapes (numbered $1, 2, \dots, k$), each with its own independent read/write tape head. Initially the input occupies the first tape, and the other tapes are blank. In each step, the machine reads the k symbols under its heads, and depending on this information and the current state, it writes a symbol on each tape, moves the heads (they don't all have to move in the same direction), and enters a new state. Its transition function is of type

$$\delta : Q \times \Gamma^3 \rightarrow Q \times \Gamma^3 \times \{L, R\}^3$$

Simulating a 3-tape TM. Given a 3-tape TM M , we build a single-tape TM N with $\Sigma_N = \Sigma_M$, and an expanded tape alphabet

$$\Gamma_N = \Sigma_M \cup \{\vdash\} \cup (\Gamma_M \cup \widehat{\Gamma}_M)^3$$

where $\widehat{\Gamma}_M = \{\widehat{a} : a \in \Gamma_M\}$, allowing us to think of its tape as divided into three tracks. Each track will contain the contents of one of M 's tape.

Each track has only one marked symbol \hat{a} , indicating the position of the corresponding tape head. M configuration might be simulated by the following configuration of N :

On input $x = a_1 \cdots a_n$, N starts with tape contents $\vdash a_1 \cdots a_n \sqcup \sqcup \cdots$. It first copies the input to its top track, and fills in the bottom two tracks with blanks; it also shifts everything right one cell so that it can fill in the leftmost cell with the simulated left end of M .

Each step of M is simulated by several steps of N : N starts at the left of the tape, then scans out until it sees all three marks, remembering the marked symbols in its finite control. It then determines what to do according to the encoded δ_M : it goes back to all three marks, rewriting the symbols on each track and moving the marks appropriately. It then returns to the left end of the tape, ready to simulate the next step of M .

The definition of Algorithm

Informally an *algorithm* is a collection of simple instructions for performing some task. They are sometimes called *procedures* or *recipes*.

To have an algorithm for a problem is to know a way of *effectively computing* or solving the problem.

There are many examples: long division, “Sieve of Eratosthenes” (for finding prime numbers), Euclid’s greatest common divisor algorithm, etc.

What are algorithms? We know it when we see one.

But can we define precisely what they are?

Why is it important to have a definition of algorithm?

A story: Hilbert's 10th Problem

At the International Congress of Mathematicians, Paris, 1900, David Hilbert famously identified 23 problems in Mathematics and posed them as a challenge for the coming century.

Hilbert's 10th Problem. Devise an *algorithm*^a that tests whether a polynomial has an integral root.

^aIn Hilbert's words: "...a process according to which it can be determined by a finite number of operations."

E.g. $p(x, y, z) = 6x^3yz^2 + 3xy^2 - x^3 - 10$ has four terms over the variables x, y, z , and has a root at $(x, y, z) = (5, 3, 0)$.

Hilbert apparently assumed that an algorithm must exist - someone need only find it.

Effective computability

Various algorithms for computing certain problems effectively were known, but a general definition of “effectively computable” that could distinguish the computable and the noncomputable was sought.

Various formalisms have been proposed:

- Turing machines (Alan Turing 1936)
- Post systems (Emil Post)
- μ -recursive functions (Kurt Gödel, Jacques Herbrand)
- λ -calculus (Alonzo Church, Stephen C. Kleene)
- Combinatory logic (Moses Schönfinkel and Haskell B. Curry)

Church-Turing Thesis

Amazingly

Theorem. All the preceding formalisms are equivalent. I.e. a problem is *solvable* (or *decidable*) w.r.t. one formalism iff it is solvable by any of the other.

This remarkable coincidence says that there is only one notion of computability.

Church-Turing Thesis. Intuitive notions of algorithms = Turing machines.

The Church-Turing Thesis is not an assertion that can be proved - it is not a theorem. Rather, it may be regarded as a definition (of algorithm).

The Thesis is widely accepted by mathematicians and computer scientists.

Hilbert's 10th Problem: Matijasevich's breakthrough

We now know:

Theorem. (Matijasevich, 1970) Hilbert's 10th Problem is algorithmically unsolvable. I.e. no algorithm (= Turing machine) exists that solves the Problem.

Without a clear definition of algorithms, it would have been impossible for mathematicians of Hilbert's era to come to the same conclusion.

We need to know what algorithms are i.e. have an accepted definition of algorithm, before we can prove that none exists for solving a given problem.

Phrasing Hilbert's 10th Problem in our terminology

Let $D = \{ p : p \text{ is a polynomial with an integral root} \}$.

Hilbert's 10th Problem asks in essence whether the set D is *decidable*. [Formally we fix an alphabet Σ and code p as a string over it, to obtain a *language* corresponding to D .]

The answer (thanks to Matijasevich) is negative. However D is r.e.

We consider a simpler problem:

$$D_1 = \{ p : p \text{ is a polynomial over } x \text{ with an integral root} \}.$$

We define a TM M_a that accepts D_1 : On input p (a polynomial over x)

Evaluate p with x set successively to $0, 1, -1, 2, -2, \dots$. If at any point the polynomial evaluates to 0, *accept*.

If D_1 has an integral root, M_1 will eventually find it, and accept. If not, M_1 will run forever. There is a similar TM M that accepts D : here M runs through all

possible settings of its variables to integral values.

Both M_1 and M are not deciders.

But M_1 can be converted to a decider because we can calculate bounds within which roots of a single-variable polynomial must lie and restrict the search accordingly. They are

$$\left[-k \frac{c_{\max}}{c_1}, k \frac{c_{\max}}{c_1} \right]$$

where k is the number of terms, c_{\max} is the coefficient with the largest absolute value and c_1 is the coefficient of the highest order term.

E.g. the bounds for $4x^3 - 7$ are $[-2, 2]$.

Matijasevich's theorem shows that no such bounds for multivariate polynomials can be calculated.