

Turing Machines

A Turing machine is a formal model for implementing an algorithm.

It can be also thought of as a very primitive programming language, but is capable of implementing any algorithm we can think of!

Informally, a Turing machine consists of:

1. A set of states (finite control).
2. An infinite tape (memory) where it can read and write. The tape initially consists of only the input string and blank everywhere else.
3. The machine can read or write one symbol in the tape in one time step; the symbol's location depends on where the cursor (arrow) of the machine points to in the tape.
4. After reading or writing a symbol, the machine can move the cursor left or right or stay at the same place. It can also change to another state.

Formal Definition

Definition 1. A Turing machine is a quadruple $M = (K, \Sigma, \delta, s)$, where:

- K is a **finite** set of states and $s \in K$ is the initial state.
- Σ is a **finite** set of symbols (**alphabet**) disjoint from K . Σ contains the special symbols \sqcup (blank) and \triangleright (start symbol).
- δ is a **transition function** from $K \times \Sigma$ to $K \cup \{h, \text{yes}, \text{no}\} \times \Sigma \times \{\rightarrow, \leftarrow, -\}$.
- The 3 special states h (halting state), “yes” (accepting state), and “no” (rejecting state) and the cursor directions, \leftarrow (left), \rightarrow (right), and $-$ (stay) are not in $K \cup \Sigma$.

δ is the “program” of the machine.

We assume, $\delta(q, \triangleright) = (p, \rho, D)$, then $\rho = \triangleright$ and $D = \rightarrow$, i.e., the cursor always does not erase the start symbol and moves right.

If, on input x , if the machine halts on x then the **output** M is $M(x)$.

If M accepts (enters “yes” state) then $M(x) = \text{yes}$ and if M rejects (enters “no”) then $M(x) = \text{no}$.

If h is reached then $M(x)$ is the string y (omitting \triangleright) at the time of halting.

If M never halts, we write $M(x) = \nearrow$.

Configuration of a TM

A configuration contains a complete description of the current state of the computation.

Formally, a configuration of a Turing machine M is a triple (q, w, u) , where $q \in K$, and $w, u \in \Sigma^*$.

q is the current state of M .

w is the string to the left of the cursor, including the symbol scanned by the cursor.

u is the string to the right of the cursor (may be empty).

“Yields” Relation

Configuration (q, w, u) **yields** configuration (q', w', u') in **one step** i.e.,

$$(q, w, u) \rightarrow^M (q', w', u')$$

if the following holds.

Let σ be the last symbol of w and let $\delta(q, \sigma) = (p, \rho, D)$. Then

$$q' = p \text{ and}$$

1. if $D = -$ then $w' = w$ with the ending σ replaced by ρ , and $u' = u$.

2. if $D = \rightarrow$ then w' is w with the last symbol replaced by ρ and the first symbol of u appended to it (\square if u is empty). u' is u with the first symbol removed (it is empty if u is empty).

3. if $D = \leftarrow \dots$

Yields is the **transitive closure** of **yields in one step**.

$(q, w, u) \rightarrow^{M^*} (q', w', u')$ if the latter configuration can be reached in some number of steps.

Example 1

Consider the TM $M = (K, \Sigma, \delta, s)$ where $K = \{s, q\}$, $\Sigma = (0, 1, \sqcup, \triangleright)$, and the following transition function δ :

$$\delta(s, 0) = (s, 0, \rightarrow)$$

$$\delta(s, 1) = (s, 1, \rightarrow)$$

$$\delta(s, \sqcup) = (q, \sqcup, \leftarrow)$$

$$\delta(s, \triangleright) = (s, \triangleright, \rightarrow)$$

$$\delta(q, 0) = (h, 1, -)$$

$$\delta(q, 1) = (q, 0, \leftarrow)$$

$$\delta(q, \triangleright) = (h, \triangleright, \rightarrow)$$

On input 11011, the computation proceeds as:

$$\begin{aligned} &(s, \triangleright, 11011) \xrightarrow{M} (s, \triangleright 1, 1011) \xrightarrow{M} (s, \triangleright 11, 011) \xrightarrow{M^*} \\ &(s, \triangleright 11011, \epsilon) \xrightarrow{M} (s, \triangleright 11011 \sqcup, \epsilon) \xrightarrow{M} (q, \triangleright 11011, \sqcup) \xrightarrow{M} \\ &(q, \triangleright 1101, 0 \sqcup) \xrightarrow{M} (q, 110, 00 \sqcup) \xrightarrow{M} (h, \triangleright 111, 00 \sqcup). \end{aligned}$$

Example 2

A TM which decides whether its input string is a palindrome.

On input w M works as follows:

1. In state s , searches for the first input symbol. It makes it into a \triangleright and “remembers” the symbol in its state.

2. It then moves to the right until it meets the first \sqcup and then one step to the left to scan the last input symbol. If this last symbol agrees with the input, it is replaced with \sqcup ; otherwise M halts with “no”.

3. Then the rightmost \triangleright is found and the process is repeated.

4. If we end up with ϵ or if there is no last symbol then M halts with “yes”.

Example 3: Reachability

The following is a **high-level** description of a TM M that decides the reachability problem.

On input $\langle G, u, v \rangle$, the encoding of a graph G and two of its vertices, u and v :

1. Select u and mark it.
2. Repeat the following stage until no new nodes are marked:
 - 2.1 For each node in G mark it if it is attached by a (in-)edge to a node that is already marked.
3. Scan all the nodes of G to determine if v is marked in which case say “yes”, otherwise say “no”.

“Turing machines as algorithms and problems as languages”.

Languages

Turing machines can **accept** and **decide** languages.

Definition 2. Let $L \subset (\Sigma - \{\sqcup\})^*$ be a language.

1. Let M be a TM such that, for any string $x \in (\Sigma - \{\sqcup\})^*$:

if $x \in L$, then $M(x) = \text{yes}$, else $M(x) = \text{no}$.

*Then M **decides** L . L is called a **recursive language** if it is decided by some TM. E.g., the TM in Example 2, decides palindromes.*

2. Let M be a TM that for any string $x \in (\Sigma - \{\sqcup\})^*$:

if $x \in L$, then $M(x) = \text{yes}$, else $M(x) = \nearrow$.

*Then M **accepts** L . L is called **recursively enumerable** if L is accepted by some TM.*

Recursive \subseteq Recursively Enumerable

Theorem 1. *If L is recursive, then it is recursively enumerable (r.e.).*

Proof. Let a TM M decide L . We construct a TM M' that behaves exactly like M , but when M is about to halt in state *no*, M' diverges (moves to the right forever and never halts). M' accepts L . \square

TMs as Enumerators

Let M be a TM. Define the language

$$E(M) = \{x : (s, \triangleright, \epsilon) \xrightarrow{M^*} (q, y \sqcup x \sqcup, \epsilon) \text{ for some } q, y\}.$$

We say that $E(M)$ is the language **enumerated** by M .

Theorem 2. *L is r.e. iff there is a machine M such that $L = E(M)$.*

Proof

Suppose $L = E(M)$.

Consider M' which, on input x , simulates M on an empty string and halts accepting only if during the simulation the string of M ever ends with $\sqcup x \sqcup$. If M halts without producing $\sqcup x \sqcup$, M' diverges.

Suppose that L is r.e., accepted by some M .

Consider the following TM M' on empty input. It simulates M on the (lexicographically) first i inputs, one after the other, for the first i steps.

If M halts with “yes” one of these inputs, say x , it writes $\sqcup x \sqcup$ at the end of its string before continuing. In no other time it writes \sqcup on its string.

M' is an enumerator of L .

TMs as Functions

TMs can compute string functions, e.g., as in Example 1.

Definition 3. *Let f be a function from $(\Sigma - \{\sqcup\})^*$ to Σ^* , and let M be a TM with alphabet Σ . We say that M **computes** f if for any $x \in (\Sigma - \{\sqcup\})^*$, $M(x) = f(x)$.*

*If such a M exists, f is called a **recursive function**.*

TMs as Algorithms

We assume that problem instances can be encoded as strings (in binary encoding).

An algorithm for a decision problem is simply a TM that decides the corresponding language, i.e., it accepts if the input is an “yes” instance and rejects otherwise.

An algorithm for an optimization problem is simply a TM that computes the appropriate function from strings to strings.

TMs with Multiple Strings

Definition 4. A k -string TM, where $k \geq 1$ is an integer, is like a (ordinary) TM except that the machine can read/write on k strings at the same time.

That is, δ is a function from $K \times \Sigma^k$ to $(K \cup \{h, \text{yes}, \text{no}\}) \times (\Sigma \times \{\leftarrow, \rightarrow, -\})^k$.

Thus, a transition is written as:

$$\delta(q, \sigma_1, \dots, \sigma_k) = (p, \rho_1, D_1, \dots, \rho_k, D_k).$$

Initially, all strings start with a \triangleright .

The input is in the first string and the output can be read from the last string (if the machine computes a function).

Configurations and “yields” are straightforward generalizations.

Time Complexity of a TM

The time complexity of a TM M on a input is the number of steps (transitions) to halting.

We say M operates **within time bound** $f(n)$ if, for any string x , the time required by M on x is at most $f(|x|)$.

If $L \subset (\Sigma - \{\sqcup\})^*$ is decided by a **multistring** TM operating in time $f(n)$, then we say $L \in TIME(f(n))$, i.e., belongs to complexity class $TIME(f(n))$.

Example: PALINDROME $\in TIME(O(n))$.

Robustness of TMs

Theorem 3. *Given any k -string TM M operating within time $f(n)$, we can construct a TM M' operating within time $O(f(n)^2)$ and such that, for any input x , $M(x) = M(x')$.*

M has the same “power” of M' (may be with a polynomial loss in efficiency).

Proof

M' simulates M as follows:

1. It simulates the k strings by concatenating them (with special delimiters) as:

$\triangleright w'_1 u_1 \# w'_2 u_2 \# \dots w'_k u_k \# \#$.

2. The first symbol of w'_i is has a special symbol $\#'$ and the last symbol is “marked”.

3. To simulate a move of M , M' scans **twice** its string from left to right and back. The first scan gathers information on the k currently scanned symbols in M and in the second scan changes the information based on the corresponding transitions of M .

4. If “overflow” occurs in any substring $\# \dots \#$, a (marked) blank is inserted.

Time complexity of M' is $O(k^2 f(|x|)^2)$ for an input of length x , where $f(|x|)$ is the time complexity of M .

TMs with Input/Output

Definition 5. Let k -string ($k > 2$) TM with I/O is an (ordinary) k -string TM where

1. the first (input) string is read-only and cursor never goes “outside” the input.
2. the last string is write-only (cursor never moves left).

Theorem 4. For any k -string TM M operating within time bound $f(n)$ there is a $(k+2)$ -string TM M' with I/O which operates within time bound $O(f(n))$.

Space Complexity

Definition 6. Suppose that, for a k -string TM and an input x :

$(s, \triangleright, x, \dots, \triangleright, \epsilon) \xrightarrow{M^*} (H, w_1, u_1, \dots, w_k, u_k)$
where $H = \{h, \text{yes}, \text{no}\}$. Then the space required by M on input x is $\sum_{i=1}^k |w_i u_i|$.

If M is a TM with I/O, then space required is $\sum_{i=2}^{k-1} |w_i u_i|$.

A TM M operates within space bound $f(n)$, if for any input x , M requires space at most $f(|x|)$.

Let L be a language. If there is a TM with I/O that can decide L and operates within space bound $f(n)$ we say $L \in SPACE(f(n))$.

Example: $PALINDROMES \in SPACE(\log n)$.

Church-Turing Thesis

“Intuitive notion of algorithms = Turing machine algorithms”

“Turing machines can implement arbitrary algorithms”

A Stronger Thesis:

“ Any reasonable model of algorithms and their time bound can be implemented by a TM within a polynomial (loss) of efficiency”

Random Access Machines

An array of **registers** each of which can contain an arbitrarily large integer.

Instruction set: READ, STORE, LOAD, ADD, JUMP, JZERO, HALF, HALT, ...

Syntax: Instruction [operand]

A program counter.

Three modes of addressing: direct, register direct, register indirect.

A RAM program consists of a finite sequence of such instructions.

Input is a finite sequence of integers (in binary) contained in input array of input registers.

Output is contained in a special register (accumulator).

Assume that all arithmetic operation of RAM takes a single step.

TM can simulate RAM

Let Π be a RAM program and let D be a set of finite sequences of integers.

Let ϕ be a function from D to integers.

We say that Π computes ϕ if, for any $I \in D$,

the program, on input I , halts with $\phi(I)$ in the accumulator.

Let the binary representation of $I = (i_1, i_2, \dots, i_n)$ denoted by $b(I)$ be the string $b(i_1); \dots; b(i_n)$.

TM M computes ϕ if, for any $I \in D$, $M(b(I)) = b(\phi(I))$.

Theorem 5. *If a RAM program Π computes a function ϕ in time $f(n)$, then there is a 7-string TM M which computes ϕ in time $O(f(n)^3)$.*

Proof

1st string: Input, never overwritten.

2nd string: Register contents – sequence of (separated) strings of the form $b(i) : b(r_i)$.

3rd string: current value of program counter

4th string: current register address

5th string: operand 1

6th string: operand 2

7th string: result/output

The states of M are subdivided into m groups, where m is the number of instructions in Π ; each group implements one instruction.

Time Complexity

Lemma 1. *After t th step of a RAM program computation on input I , the contents of any register have length at most $t + \ell(I) + \ell(B)$.*

Proof: By induction on t .

Easy to see that it holds for operations such as LOAD, STORE, READ etc.

Consider an arithmetic instruction e.g., ADD.

The length of the result is $\leq 1 + t - 1 + \ell(I) + \ell(B)$.

Simulating each instruction of Π takes $O((f(n))^2)$ time where n is the length of the input.

Decoding current instruction and the constants it contains can be done in constant time.

Fetching the value of the registers involved takes $O((f(n))^2)$ time:

(the second string contains $O(f(n))$ pairs each of length $O(f(n))$).

The computation involves arithmetic functions on integers of length $O(f(n))$ and can be done in $O(f(n))$.

Nondeterministic Turing Machines (NTM)

A NTM $N = (K, \Sigma, \Delta, s)$ is like an ordinary TM except that instead of a transition function we have a transition **relation** Δ .

$$\Delta \subset (K \times \Sigma) \times [(K \cup \{h, \text{yes}, \text{no}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}].$$

Many (or none) next steps for each state-symbol combination.

"yields" is a relation.

Definition 7. We say that a NTM N **decides** a language L if for any $x \in \Sigma^*$, the following is true: $x \in L$ iff $(s, \triangleright, x) \rightarrow^{N^*} (\text{yes}, w, u)$ for some strings w and u .

N **decides** L in time $f(n)$ if N decides L , and for any $x \in \Sigma^*$, if $(s, \triangleright, x) \rightarrow^{N^k} (q, w, u)$, then $k \leq f(|x|)$.

NTIME

The set of languages decided by NTMs within time f is called $NTIME(f(n))$.

NP is the set of languages that can be decided by a NTM in polynomial time.

$$P \subseteq NP.$$

Example: HAMILTONIAN CYCLE $\in NP$.

Can be decided by a (2-string) NTM in $O(n^2)$ time.

Simulating NTMs by TMs

Theorem 6. *Let L be decided by a NTM N in time $f(n)$. Then it is decided by a 3-string TM M in time $O(c^{f(n)})$, where $c > 1$ is some constant depending on N . That is,*

$$NTIME(f(n)) \subseteq \cup_{c>1} TIME(c^{f(n)}).$$

Proof. Let $N = (K, \Sigma, \Delta, s)$.

For each $(q, \sigma) \in K \times \Sigma$, consider

$$C_{q,\sigma} = \{(q', \sigma', D) : (q, \sigma), (q', \sigma', D) \in \Delta\}.$$

Let $d = \max_{q,\sigma} |C_{q,\sigma}|$ — “degree of nondeterminism”.

Assume $d > 1$.

M considers all paths in the computation tree in order of increasing length and simulates N on each path.

M maintains computation paths in its second string.

M uses the third string to generate computation paths.

The total time is bounded by $\sum_{t=1}^{f(n)} (O(d))^t = (O(d))^{f(n)}$

Space Complexity of NTMs

Definition 8. Given an k -string NTM with input and output $N = (K, \Sigma, \Delta, s)$ we say that N decides language L within space $f(n)$ if N decides L and moreover, for any $x \in (\Sigma - \{\sqcup\})^*$, if $(s, \triangleright, x, \dots, \triangleright, \epsilon) \rightarrow^{N^*} (q, w_1, u_1, \dots, w_k, u_k)$, then

$$\sum_{j=2}^{k-1} |w_j u_j| \leq f(|x|).$$

Let L be a language. If there is a NTM with I/O that can decide L and operates within space bound $f(n)$ we say $L \in NSPACE(f(n))$.

Example: REACHABILITY $\in NSPACE(O(\log n))$.

Universal Turing Machines

Takes as input a description of another TM M and an input x for M and simulates the behavior of M on that machine, i.e.,

$$U(M; x) = M(x).$$

We encode the symbols and states of TM M by integers (in binary).

Let $M = (K, \Sigma, \delta, s)$. Then we assume

$\Sigma = \{1, 2, \dots, |\Sigma|\}$ and $K = \{|\Sigma| + 1, |\Sigma| + 2, \dots, |\Sigma| + |K|\}$

s is $|\Sigma| + 1$.

Special things like $\rightarrow, \leftarrow, -, yes, no$ are encoded by integers from $|K| + |\Sigma| + 6$.

All numbers will be in binary with $\lceil \log(|K| + |\Sigma| + 6) \rceil$ bits.

The description of M will start with $|K|$, then $|\Sigma|$ and then δ .

U uses 2-strings to simulate M .

Undecidability

Problems that have no algorithms or languages that are not recursive.

Implied by the fact that there are more languages than TMs.

HALTING Problem: Given the description of a TM M and its input x , will M halt on x ?

$$H = \{M; x : M(x) \neq \nearrow\}$$

Theorem 7. *H is undecidable, i.e., not recursive.*

All r.e. languages can be **reduced** to H , i.e., if we can solve HALTING we can decide **any** r.e. language.

Thus *HALTING* is a **complete** problem for the class of r.e. languages.