

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 1. 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 . The 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 2. *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.