

1 Introduction

Complexity: How fast can a problem be solved by 'computer'.

Computers have limits, how fast can they solve a problem? This section discussed these types of issues.

2 Turing Machines

To figure out how fast they can solve problems, we need to know just what *they* are. The **Turing Machine (TM)** is a simple, powerful, abstract model of computation. It turns out that *Turing Machines* are surprisingly powerful.

- 1) A TM can do practically anything that a conventional computer can do (except it does not have a nice 21 inch display).
- 2) A problem can be solved in polytime on a TM \Leftrightarrow it can be solved in any other reasonable model of computation (ie Java, C++, etc).

2.1 Definition of a Turing Machine

First we define alphabets and strings. Turing Machines compute over strings.

- An alphabet Σ is a finite set of symbols.
- Σ^* is the set of all finite sequences of symbols from alphabet Σ (ie finite strings).
- ϵ is the empty string

A **Turing Machine (TM)** is described by a tuple

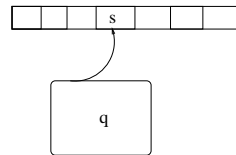
$$(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

where

- 1) Q is a finite set of **states**
- 2) $\Sigma \subset \Gamma$ is a finite set of symbols **the input alphabet** which does not include the blank symbol (b). Usually $\Sigma = \{0, 1\}$.
- 3) Γ is a finite set of symbols (**the tape alphabet**) $b \in \Gamma$.
- 4) $\delta : Q - \{q_{accept}, q_{reject}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function.
- 5) $q_0 \in Q$ is the **start state**
- 6) $q_{accept} \in Q$ is the **accept state**
- 7) $q_{reject} \in Q$ is the **reject state**

The TM consists of

- 1) A 2-way infinite tape. The tape consists of countably infinite sequence of squares. These squares are used to store members of Γ (the tape alphabet).
- 2) a finite state control (FSC) (the CPU of the TM) which is used to compute δ . At any given time, this FSC is in a single state (member of Q).
- 3) a movable read/write head. The read/write head is always scanning a single tape square. It is under the control of the FSC.



2.2 Operation of A Turing Machine

We now describe how Turing Machine M operates

- A **configuration** C of M consists of a complete description of the tape contents of M , the location of the read/write head and the state of the FSC.
- The initial configuration of M on input $x \in \Sigma^*$ is the configuration of M where
 - The 2-way infinite tape consists of all blank symbols except the input string x written in consecutive squares (from left to right).
 - The read/write head is positioned on the left most symbol of x
 - The FSC is in state q_0
- A **halting configuration** is any configuration in which the FSC is in state q_{accept} or q_{reject} . In the first case, M is in an **accepting configuration**, in the second case, it is in a **rejecting configuration**.

We describe one step in the computation of M .

Say M is in configuration C with the read/write head of M positioned over symbol s , the FSC in state q and that $\delta(q, s) = (q', s', h)$, then the subsequent configuration C' is obtained from C as a result of M replacing symbol s under the read/write head with s' , moving the head one square in the direction specified by h (ie LEFT or RIGHT) and setting the state of the FSC to q' .

We write $C \rightarrow C'$ when configuration C' follows from configuration C as described above.

If C is a halting configuration, then we never have $C \rightarrow C'$. That is, the computation has halted.

The **computation of M on input $x \in \Sigma^*$ ($C(M, x)$)** is the possibly infinite sequence of configurations C_0, C_1, \dots where

- C_0 is the initial configuration of M on input x
- $(\forall i \geq 0) C_i \rightarrow C_{i+1}$

If $C(M, x)$ is finite (ie $C(M, x) = C_0, \dots, C_k$) then C_k is a halting configuration. If C_k is an accepting configuration, then we say M **accepts** x . Otherwise, C_k is a rejecting configuration and we say M **rejects** x .

Example 1. $PAL = \text{even length palindromes} = \{yy^r \mid y \in \{0, 1\}^*, \text{ where } y^r \text{ means } y \text{ spelled backwards}\}$

State	Symbol	Action
q_0	0	print b , R , q_1
q_1	0 or 1	print 0 or 1 resp, R , q_1
q_1	b	print b , L , q_2
q_2	0	print b , L , q_3
q_2	1 or b	q_{reject}
q_3	0 or 1	print 0 or 1 resp, L , q_3
q_3	b	print b , R, q_0
q_0	1	print b , R , q_4
q_4	0 or 1	print 0 or 1 resp, R , q_4
q_4	b	print b , L , q_5
q_5	1	print b , L , q_3
q_5	0 or b	q_{reject}
q_0	b	q_{accept}

The idea is that the TM initially scans the leftmost symbol of the input. If that symbol is a 0 , then the 0 is erased, and the head moves right repeatedly (one square at a time) until it reaches the b to the right end of the input. It then moves left one square, and checks to see if that symbol is 0 . If it is not 0 , then the machine halts and rejects. If it is 0 , then it erases that 0 and moves all the way back left to the first b . It then moves right one square, and goes back to the initial state.

Initially, if the scanned square is b , then the machine halts and accepts. If the scanned square is 1 , then the machine proceeds as above, except it checks for a 1 at the the right end of the input, instead of a 0 .

3 What Turing Machines Compute

We associate languages and functions with Turing Machines. We say what it means for a language to be decidable/semi-decidable. We associate functions with some Turing Machines and say what it means for a function to be computable.

3.1 Languages and Turing Machines

We now describe what it is that a Turing Machine actually computes...

Let M be a TM with input alphabet Σ

Definition 1. $L(M) = \{x \in \Sigma^* | M \text{ accepts } x\}$. $L(M)$ is the language accepted by M or the language recognized by M . If M halts on all inputs then M is a decider.

Note: If M is not a decider then M may not halt on some inputs $x \notin L(M)$.

Say that we are given an arbitrary language $L \subseteq \Sigma^*$, then

Definition 2. L is recognizable (or semi-decidable) if there is some TM M such that $L(M) = L$ (we say that M recognizes or accepts L). If M is a decider then we say that L is decidable and that M decides L .

Example 2. By the above example, PAL is recognizable, it is semi-decidable and the machine described in the example decides PAL. It also recognizes PAL.

3.2 Functions and Turing Machines

Turing Machines can do more than determine if $x \in L$ for some language L . They can compute functions.

For Σ_1, Σ_2 alphabets,

- $f : \Sigma_1^* \rightarrow \Sigma_2^* \cup \{\uparrow\}$ is a **partial function**.
- If $f(x) \in \Sigma_2^*$ then we say that f is **defined at x** .
- Otherwise, we say that $f(x)$ is **undefined** and write $f(x) = \uparrow$ or $f(x) \uparrow$.
- If f is defined on all of Σ_1^* then we say that f is a **total function**.

Say M is a Turing Machine for which either

- 1) $C(M, x)$ is infinite **or**
- 2) $C(M, x)$ halts in a configuration C' where C' is in the accepting state, the tape consists of all blanks except for a string y and the read/write head scans the leftmost symbol of y .

For such nicely behaved Turing Machines, we write ϕ_M for the function computed by M . In the first case above, we write $\phi_M(x) = \uparrow$. In the second case, we write $\phi_M(x) = y$.

If f is a function such that $f = \phi_M$ for some TM M then we say that f is **computable**.

Example 3. *Add 1 in binary. ...bb1001011bb... is initial tape string, and the head is initially scanning the leftmost nonblank symbol Here $\Gamma = \{0, 1, b\}$, $\Sigma = \{0, 1\}$ and $Q = \{q_0, q_1, q_2, q_{\text{accept}}, q_{\text{reject}}\}$. The δ is specified by the following table:*

State	Symbol	Action
q_0	0	print 0, R, q_0
q_0	1	print 1, R, q_0
q_0	b	print b, L, q_1
q_1	0	print 1, L, q_2
q_1	1	print 0, L, q_1
q_1	b	print 1, L, q_2
q_2	0	print 0, L, q_2
q_2	1	print 1, L, q_2
q_2	b	print b, R, q_{accept}

This machine operates as follows: It uses state q_0 to scan to the rightmost symbol (the low order bit) of the input. Next it uses state q_1 to add one to the input in a right to left scan. We continue in state q_1 until we have nothing more carries. State q_2 finishes the right to left scan of the input. At the end of the execution, the read/write head is scanning the leftmost symbol of the binary representation of $i + 1$ (if the tape initially held the binary representation of i).

By our conventions above, the total function $f(i) = i + 1$ is computable (where i and $i + 1$ are represented in binary). The machine M described in this example has $\phi_M(x) = y$ where x is the binary representation of a number i and y is the binary representation of $i + 1$. We say that M computes f .