

# Turing Machine\*

Xiaofeng Gao

Department of Computer Science and Engineering  
Shanghai Jiao Tong University, P.R.China

Algorithm Class @ Shanghai Jiao Tong University

\*Special thanks is given to Prof. Yuxi Fu for sharing his teaching materials.

## What is Effective Procedure


- Methods for addition, multiplication ...
  - ▷ Given  $n$ , finding the  $n$ th prime number.
  - ▷ Differentiating a polynomial.
  - ▷ Finding the highest common factor of two numbers  $HCF(x, y) \rightarrow$  Euclidean algorithm
  - ▷ Given two numbers  $x, y$ , deciding whether  $x$  is a multiple of  $y$ .
- Their implementation requires no ingenuity, intelligence, inventiveness.

## Outline

- 1 Effective Procedures
  - Basic Concepts
  - Computable Function
- 2 Turing Machine
  - Introduction
  - One-Tape Turing Machine
  - Multi-Tape Turing Machine
- 3 TM Variation and TM-Computability
  - TM Variations
  - Computable and Decidable

## Intuitive Definition

An *algorithm* or *effective procedure* is a **mechanical rule**, or **automatic method**, or **programme** for performing some mathematical operations.

Blackbox:      input  $\longrightarrow$    $\longrightarrow$  output

## What is “effective procedure”?

**An Example:** Consider the function  $g(n)$  defined as follows:

$$g(n) = \begin{cases} 1, & \text{if there is a run of exactly } n \text{ consecutive 7's} \\ & \text{in the decimal expansion of } \pi, \\ 0, & \text{otherwise.} \end{cases}$$

Question: Is  $g(n)$  effective?

▷ The answer is unknown  $\neq$  the answer is negative.

**Other Examples:**

- *Theorem Proving* is in general not effective/algorithmic.
- *Proof Verification* is effective/algorithmic.

## Computable Function

When an algorithm or effective procedure is used to calculate the value of a numerical function then the function in question is **effectively calculable** (or **algorithmically computable**, **effectively computable**, **computable**).

**Examples:**

- $HCF(x, y)$  is computable;
- $g(n)$  is non-computable.

## Algorithm

An algorithm is a procedure that consists of a finite set of *instructions* which, given an *input* from some set of possible inputs, enables us to obtain an *output* through a systematic execution of the instructions that *terminates* in a finite number of steps.

## Development of Computation Models

1. Gödel-Kleene (1936): Partial recursive functions.
2. Turing (1936): Turing machines.
3. Church (1936):  $\lambda$ -terms.
4. Post (1943): Post systems.
5. Markov (1951): Variants of the Post systems.
6. Shepherdson-Sturgis (1963): URM-computable functions.

**Church-Turing Thesis:** Each of the above proposals for a characterization of the notion of effective computability gives rise to the **same** class of functions.

- An English student of Church
- Introduced a machine model for effective calculation in “On Computable Numbers, with an Application to the Entscheidungsproblem”, Proc. of the London Mathematical Society, 42:230-265, 1936.
- Turing Machine, Halting Problem, Turing Test



A Turing machine has five components:

1. A finite set  $\{s_1, \dots, s_n\} \cup \{\triangleright, \triangleleft\} \cup \{\square\}$  of **symbols**.
2. A **tape** consists of an infinite number of cells, each cell may store a symbol.

...  ...

3. A **reading head** that scans and writes on the cells.
4. A finite set  $\{q_S, q_1, \dots, q_m, q_H\}$  of **states**.
5. A finite set of **instructions** (specification).

What are necessary for a machine to calculate a function?

- The machine should be able to interpret numbers
- The machine must be able to operate and manipulate numbers according to a set of predefined instructions

and

- The input number has to be stored in an accessible place
- The output number has to be put in an accessible place
- There should be an accessible place for the machine to store intermediate results

The diagram illustrates a Turing Machine ( $M$ ) with a tape and a reading head. The machine's body is a large rectangle labeled  $M$ . Inside it, a smaller rectangle is labeled  $q_1$ , with an arrow pointing to it from the text "Current state display window". Below the machine body is a horizontal tape. The tape is a long rectangle divided into squares. The squares are labeled from left to right:  $\dots$ ,  $s_1$ , an empty square,  $s_3$ ,  $s_2$ , an empty square,  $s_1$ , an empty square,  $s_4$ , and  $\dots$ . A "Reading head" is shown as a small vertical rectangle positioned over the square containing  $s_2$ . An arrow points from the text "Square being scanned" to this square.

## One-Tape Turing Machine

The input data

$$\triangleright s_1^1 \dots s_{i_1}^1 \square \dots \square s_1^k \dots s_{i_k}^k \triangleleft \square \dots$$

The reading head may write a symbol, move left, move right.

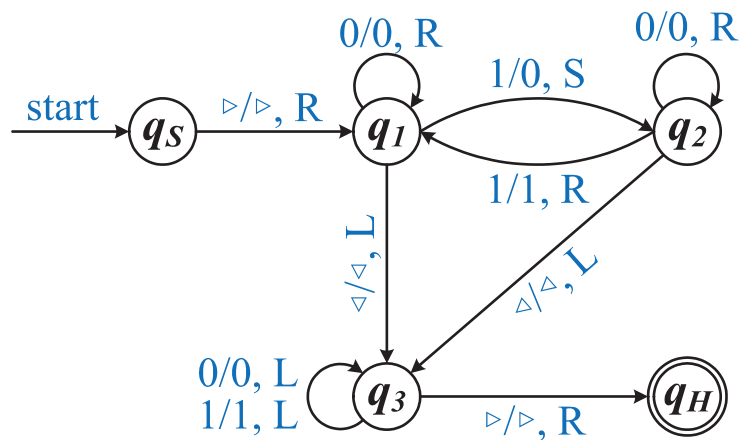
An instruction is of the form:

$$\langle q_i, s_j \rangle \rightarrow \langle q_l, s_k, L/R/S \rangle,$$

which means when reads  $s_j$  with state  $q_i$ , the machine will turn to state  $q_l$ , replace  $s_j$  with  $s_k$ , and turn one cell to the left.

The direction can be  $L$ ,  $R$ , or  $S$ , meaning move to left, right, or stay at the current position.

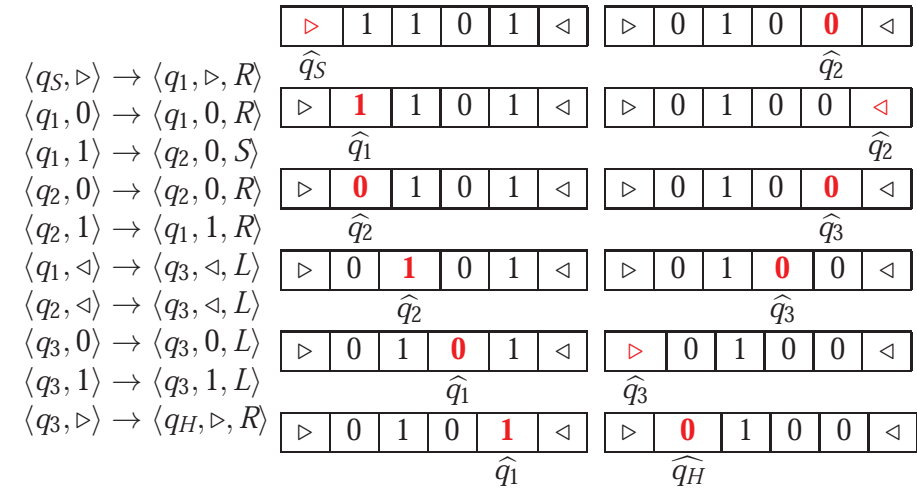
## State Transition Diagram



$M$ 's action is to work from left to right along the tape, replacing alternate 1's by the symbol 0's.

## An Example

Given a Turing machine  $M$  with the alphabet  $\{0, 1\} \cup \{\triangleright, \square, \triangleleft\}$ .



## Multi-Tape Turing Machine

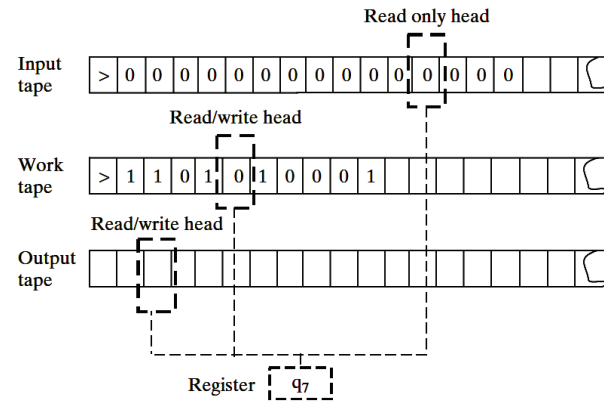
A multi-tape TM is described by a tuple  $(\Gamma, Q, \delta)$  containing

- A finite set  $\Gamma$  called **alphabet**, of symbols. It contains a blank symbol  $\square$ , a start symbol  $\triangleright$ , and the digits 0 and 1.
- A finite set  $Q$  of **states**. It contains a start state  $q_{start}$  and a halting state  $q_{halt}$ .
- A **transition function**  $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times L, S, R^k$ , describing the rules of each computation step.

**Example:** A 2-Tape TM will have transition function (also named as **specification**) like follows:

$$\begin{aligned} \langle q_s, \triangleright, \triangleright \rangle &\rightarrow \langle q_1, \triangleright, R, R \rangle \\ \langle q_1, 0, 1 \rangle &\rightarrow \langle q_2, 0, S, L \rangle \end{aligned}$$

## Computation and Configuration



Computation, configuration, initial/final configuration

## Preparation

To recognize palindrome we need to check the input string, output 1 if the string is a palindrome, and 0 otherwise.

Initially the input string is located on the first tape like " $\triangleright 0110001 \triangleleft \square\square\square\dots$ ", strings on all other tapes are " $\triangleright\square\square\square\dots$ ".

The head on each tape points the first symbol " $\triangleright$ " as the starting state, with state mark  $q_s$ .

In the final state  $q_F$ , the output of the  $k^{th}$  tape should be " $\triangleright 1 \triangleleft \square$ " if the input is a palindrome, and " $\triangleright 0 \triangleleft \square$ " otherwise.

## A 3-Tape TM for the Palindrome Problem

A **palindrome** is a word that reads the same both forwards and backwards. For instance:

**ada**, **anna**, **madam**, and **nitalarbralatin**.

**Requirement:** Give the specification of  $M$  with  $k = 3$  to recognize palindromes on symbol set  $\{0, 1, \triangleright, \triangleleft, \square\}$ .

## A 3-Tape TM for the Palindrome Problem

$Q = \{q_s, q_h, q_c, q_l, q_t, q_r\}$ ;  $\Gamma = \{\square, \triangleright, \triangleleft, 0, 1\}$ ; two work tapes.

**Start State:**

$\langle q_s, \triangleright, \triangleright, \triangleright \rangle \rightarrow \langle q_c, \triangleright, \triangleright, R, R, R \rangle$

**Begin to copy:**

$\langle q_c, 0, \square, \square \rangle \rightarrow \langle q_c, 0, \square, R, R, S \rangle$

$\langle q_c, 1, \square, \square \rangle \rightarrow \langle q_c, 1, \square, R, R, S \rangle$

$\langle q_c, \triangleleft, \square, \square \rangle \rightarrow \langle q_l, \square, \square, L, S, S \rangle$

**Return back to the leftmost:**

$\langle q_l, 0, \square, \square \rangle \rightarrow \langle q_l, \square, \square, L, S, S \rangle$

$\langle q_l, 1, \square, \square \rangle \rightarrow \langle q_l, \square, \square, L, S, S \rangle$

$\langle q_l, \triangleright, \square, \square \rangle \rightarrow \langle q_t, \square, \square, R, L, S \rangle$

**Begin to compare:**

$\langle q_t, \triangleleft, \triangleright, \square \rangle \rightarrow \langle q_r, \triangleright, 1, S, S, R \rangle$

$\langle q_t, 0, 1, \square \rangle \rightarrow \langle q_r, 1, 0, S, S, R \rangle$

$\langle q_t, 1, 0, \square \rangle \rightarrow \langle q_r, 0, 0, S, S, R \rangle$

$\langle q_t, 0, 0, \square \rangle \rightarrow \langle q_t, 0, \square, R, L, S \rangle$

$\langle q_t, 1, 1, \square \rangle \rightarrow \langle q_t, 1, \square, R, L, S \rangle$

**Ready to terminate:**

$\langle q_r, \triangleleft, \triangleright, \square \rangle \rightarrow \langle q_h, \triangleright, \triangleleft, S, S, S \rangle$

$\langle q_r, 0, 1, \square \rangle \rightarrow \langle q_h, 1, \triangleleft, S, S, S \rangle$

$\langle q_r, 1, 0, \square \rangle \rightarrow \langle q_r, 0, \triangleleft, S, S, S \rangle$

## Language System

Let  $\Sigma = \{a_1, \dots, a_k\}$  be the set of symbols, called **alphabet**.

A **string (word)** from  $\Sigma$  is a sequence  $a_{i_1}, \dots, a_{i_n}$  of symbols from  $\Sigma$ .

$\Sigma^*$  is the set of all words/strings from  $\Sigma$ . (**Kleene Star**)

For example, if  $\Sigma = \{a, b\}$ , we have

$$\Sigma^* = \{a, b\}^* = \{\Lambda, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, \dots\}.$$

$\Lambda$  is the **empty string**, that has no symbols. ( $\varepsilon$ )

 $\{0, 1, \square, \triangleright\}$  vs. Larger Alphabets

Suppose  $M$  has  $k$  tapes with the alphabet  $\Gamma$ .

A symbol of  $M$  is encoded in  $\tilde{M}$  by a string  $\sigma \in \{0, 1\}^*$  of length  $\log |\Gamma|$ .

A state  $q$  in  $M$  is turned into a number of states in  $\tilde{M}$

- $q$ ,
- $\langle q, \sigma_1^1, \dots, \sigma_1^k \rangle$  where  $|\sigma_1^1| = \dots = |\sigma_1^k| = 1$ ,
- $\dots$ ,
- $\langle q, \sigma_{\log |\Gamma|}^1, \dots, \sigma_{\log |\Gamma|}^k \rangle$ , the size of  $\sigma_{\log |\Gamma|}^1, \dots, \sigma_{\log |\Gamma|}^k$  is  $\log |\Gamma|$ .

 $\{0, 1, \square, \triangleright\}$  vs. Larger Alphabets

**Fact:** If  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  is computable in time  $T(n)$  by a TM  $M$  using the alphabet set  $\Gamma$ , then it is computable in time  $4 \log |\Gamma| T(n)$  by a TM  $\tilde{M}$  using the alphabet  $\{0, 1, \square, \triangleright\}$ .

 $\{0, 1, \square, \triangleright\}$  vs. Larger Alphabets

To simulate one step of  $M$ , the machine  $\tilde{M}$  will

- 1 use  $\log |\Gamma|$  steps to read from each tape the  $\log |\Gamma|$  bits encoding a symbol of  $\Gamma$ ,
- 2 use its state register to store the symbols read,
- 3 use  $M$ 's transition function to compute the symbols  $M$  writes and  $M$ 's new state given this information,
- 4 store this information in its state register, and
- 5 use  $\log |\Gamma|$  steps to write the encodings of these symbols on its tapes.

## $\{0, 1, \square, \triangleright\}$ vs. Larger Alphabets

**Example:**  $\{0, 1, \square, \triangleright\}$  vs. English Alphabets

M's tape: 

>	m	a	c	h	i	n	e										
---	---	---	---	---	---	---	---	--	--	--	--	--	--	--	--	--	--

$\tilde{M}$ 's tape: 

>	0	1	1	0	1	0	0	0	0	1	0	0	0	1	1			
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	--	--

## Single-Tape vs. Multi-Tape

- The basic idea is to interleave  $k$  tapes into one tape.
- The first  $n + 1$  cells are reserved for the input.

M's 3 work tapes:

Tape 1: 

c	o	m	p	i	e	t	e	l	y								
---	---	---	---	---	---	---	---	---	---	--	--	--	--	--	--	--	--

Tape 2: 

r	e	p	l	a	c	e	d										
---	---	---	---	---	---	---	---	--	--	--	--	--	--	--	--	--	--

Tape 3: 

m	a	c	h	i	n	e	s										
---	---	---	---	---	---	---	---	--	--	--	--	--	--	--	--	--	--

Encoding this in one tape of  $\tilde{M}$ :

1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
c	r	m	o	ê	a	m	p	c	ê	l	h	i	a	i	e	c	n

- Every symbol  $a$  of  $M$  is turned into two symbols  $a, \hat{a}$  in  $\tilde{M}$ , with  $\hat{a}$  used to indicate head position.

## Single-Tape vs. Multi-Tape

Define a single-tape TM to be a TM that has one read-write tape.

**Fact:** If  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  is computable in time  $T(n)$  by a TM  $M$  using  $k$  tapes, then it is computable in time  $5kT(n)^2$  by a single-tape TM  $\tilde{M}$ .

## Single-Tape vs. Multi-Tape

The outline of the algorithm:

The machine  $\tilde{M}$  places  $\triangleright$  after the input string and then starts copying the input bits to the imaginary input tape. During this process whenever an input symbol is copied it is overwritten by  $\triangleright$ .

$\tilde{M}$  marks the  $n + 2$ -cell,  $\dots$ , the  $n + k$ -cell to indicate the initial head positions.

$\tilde{M}$  Sweeps  $kT(n)$  cells from the  $(n + 1)$ -th cell to right, recording in the register the  $k$  symbols marked with the hat  $\hat{\_}$ .

$\tilde{M}$  Sweeps  $kT(n)$  cells from right to left to update using the transitions of  $M$ . Whenever it comes across a symbol with hat, it moves right  $k$  cells, and then moves left to update.

## Unidirectional Tape vs. Bidirectional Tape

Define a bidirectional Turing Machine to be a TM whose tapes are infinite in both directions.

**Fact:** If  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  is computable in time  $T(n)$  by a bidirectional TM  $M$ , then it is computable in time  $4T(n)$  by a TM  $\tilde{M}$  with one-directional tape.

## Unidirectional Tape vs. Bidirectional Tape

Let  $H$  range over  $\{L, S, R\}$  and let  $-H$  be defined by

$$-H = \begin{cases} R, & \text{if } H = L, \\ S, & \text{if } H = S, \\ L, & \text{if } H = R. \end{cases}$$

$\tilde{M}$  contains the following transitions:

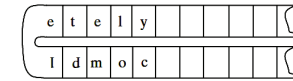
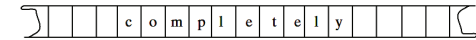
$$\begin{aligned} \langle \bar{q}, (\triangleright, \triangleright) \rangle &\rightarrow \langle \underline{q}, (\triangleright, \triangleright), R \rangle \\ \langle \underline{q}, (\triangleright, \triangleright) \rangle &\rightarrow \langle \bar{q}, (\triangleright, \triangleright), R \rangle \end{aligned}$$

$$\begin{aligned} \langle \bar{q}, (a, b) \rangle &\rightarrow \langle \bar{q}', (a', b), H \rangle \text{ if } \langle q, a \rangle \rightarrow \langle q', a', H \rangle \\ \langle \underline{q}, (a, b) \rangle &\rightarrow \langle \underline{q}', (a, b'), -H \rangle \text{ if } \langle q, b \rangle \rightarrow \langle q', b', H \rangle \end{aligned}$$

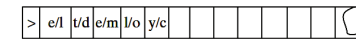
## Unidirectional Tape vs. Bidirectional Tape

- The idea is that  $\tilde{M}$  makes use of the alphabet  $\Gamma \times \Gamma$ .

$M$ 's tape is infinite in both directions:



$\tilde{M}$  uses a larger alphabet to represent it on a standard tape:



- Every state  $q$  of  $M$  is turned into  $\bar{q}$  and  $\underline{q}$ .

## TM-Computable Function

What does it mean that a TM computes a (partial)  $n$ -ary function  $f$ ?

Let  $M$  be a TM and  $a_1, \dots, a_n, b \in \mathbb{N}$ . When computation  $M(a_1, \dots, a_n)$  converges to  $b$  if  $M(a_1, \dots, a_n) \downarrow$  and  $r_1 = b$  in the final configuration. We write  $M(a_1, \dots, a_n) \downarrow b$ .

- $M$  **TM-computes**  $f$  if, for all  $a_1, \dots, a_n, b \in \mathbb{N}$ ,

$$M(a_1, \dots, a_n) \downarrow b \text{ iff } f(a_1, \dots, a_n) = b$$

- Function  $f$  is **TM-computable** if there is a Turing Machine that TM-computes  $f$ .
- (We abbreviate “TM-computable” to “computable”)



## Function Defined by Program

Given any program  $P$  and  $n \geq 1$ , by thinking of the effect of  $P$  on initial configurations of the form  $a_1, \dots, a_n, 0, 0, \dots$ , there is a unique  $n$ -ary function that  $P$  computes, denoted by  $f_P^{(n)}$ .

$$f_P^{(n)}(a_1, \dots, a_n) = \begin{cases} b, & \text{if } P(a_1, \dots, a_n) \downarrow b, \\ \text{undefined}, & \text{if } P(a_1, \dots, a_n) \uparrow. \end{cases}$$

## Decidable Predicate and Decidable Problem

Suppose that  $P(x_1, \dots, x_n)$  is an  $n$ -ary predicate of natural numbers. The **characteristic function**  $c_P(\mathbf{x})$ ,

$$f_M^{(n)}(a_1, \dots, a_n) = \begin{cases} 1, & \text{if } P(\mathbf{x}) \text{ holds,} \\ 0, & \text{if otherwise.} \end{cases}$$

The predicate  $P(\mathbf{x})$  is **decidable** if  $c_P$  is computable; it is **undecidable** otherwise.

## Predicate and Decision Problem

The value of a predicate is either 'true' or 'false'.

The answer of a *decision problem* is either 'yes' or 'no'.

**Example:** Given two numbers  $x, y$ , check whether  $x$  is a multiple of  $y$ .

Input:  $x, y$ ;

Output: 'Yes' or 'No'.

The operation amounts to calculation of the function

$$f(x, y) = \begin{cases} 1, & \text{if } x \text{ is a multiple of } y, \\ 0, & \text{if otherwise.} \end{cases}$$

Thus the property or predicate ' $x$  is a multiple of  $y$ ' is **algorithmically** or **effectively decidable**, or just **decidable** if function  $f$  is computable.

## Computability on other Domains

Suppose  $D$  is an object domain. A **coding** of  $D$  is an explicit and **effective injection**  $\alpha : D \rightarrow \mathbb{N}$ . We say that an object  $d \in D$  is **coded** by the natural number  $\alpha(d)$ .

A function  $f : D \rightarrow D$  extends to a numeric function  $f^* : \mathbb{N} \rightarrow \mathbb{N}$ . We say that  $f$  is computable if  $f^*$  is computable.

$$f^* = \alpha \circ f \circ \alpha^{-1}$$

## Example

Consider the domain  $\mathbb{Z}$ . An explicit coding is given by the function  $\alpha$  where

$$\alpha(n) = \begin{cases} 2n, & \text{if } n \geq 0, \\ -2n - 1, & \text{if } n < 0. \end{cases}$$

Then  $\alpha^{-1}$  is given by

$$\alpha^{-1}(m) = \begin{cases} \frac{1}{2}m, & \text{if } m \text{ is even,} \\ -\frac{1}{2}(m+1), & \text{if } m \text{ is odd.} \end{cases}$$

## Example (Continued)

Consider the function  $f(x) = x - 1$  on  $\mathbb{Z}$ , then  $f^* : \mathbb{N} \rightarrow \mathbb{N}$  is given by

$$f^*(x) = \begin{cases} 1 & \text{if } x = 0 \text{ (i.e. } x = \alpha(0)), \\ x - 2 & \text{if } x > 0 \text{ and } x \text{ is even (i.e. } x = \alpha(n), n > 0), \\ x + 2 & \text{if } x \text{ is odd (i.e. } x = \alpha(n), n < 0). \end{cases}$$

It is a routine exercise to write a program that computes  $f^*$ , hence  $x - 1$  is a computable function on  $\mathbb{Z}$ .