

# Turing Machines

(Based on [Sipser 2006, 2013])

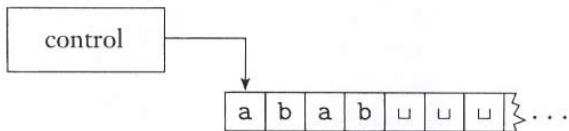
Yih-Kuen Tsay

Department of Information Management  
National Taiwan University

# Turing Machines

- 🌐 Finite and pushdown automata are too restricted to serve as models of general-purpose computers.
- 🌐 A *Turing machine* is similar to a finite automaton but with an *unlimited and unrestricted* memory—an **infinite tape**. It has a tape head that can *read and write* symbols and *move* around on the tape.
- 🌐 A Turing machine can do everything that a real computer (as we know it) can do.
- 🌐 Nonetheless, there are problems that no Turing machines, and hence no real computers (algorithms), can solve.

# Turing Machines (cont.)



**FIGURE 3.1**  
Schematic of a Turing machine

Source: [Sipser 2006]

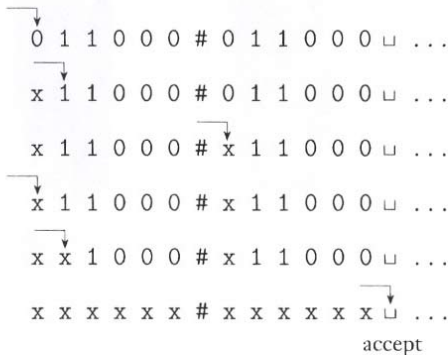
Note: the arrow represents the tape head (or read/write head) that can move around on the infinite tape.

# An Example Turing Machine

Let  $B = \{w\#w \mid w \in \{0, 1\}^*\}$ . A Turing machine  $M_1$  for  $B$  may work as follows:

1. Scan the input to be sure that it contains a single  $\#$  symbol. If not, *reject*.
2. Zig-zag across the tape to corresponding positions on either side of the  $\#$  symbol to check whether these positions contain the same symbol. If they do not, *reject*.  
Cross off symbols as they are checked.
3. When all symbols to the left of  $\#$  have been crossed off, check for any remaining symbols to the right of the  $\#$ . If any symbols remain, *reject*; otherwise, *accept*.

# An Example Turing Machine (cont.)



**FIGURE 3.2**

Snapshots of Turing machine  $M_1$  computing on input  $011000\#011000$

Source: [Sipser 2006]

# Formal Definition of a TM

## Definition (3.3)

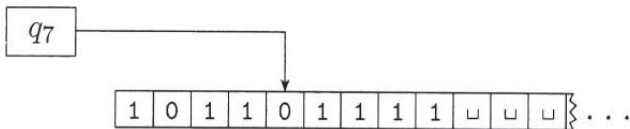
A **Turing machine** is a 7-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ , where  $Q$ ,  $\Sigma$ , and  $\Gamma$  are all finite sets and

1.  $Q$  is the set of states,
2.  $\Sigma$  is the input alphabet, where the *blank* symbol  $\sqcup \notin \Sigma$ ,
3.  $\Gamma$  is the tape alphabet, where  $\sqcup \in \Gamma$  and  $\Sigma \subseteq \Gamma$ ,
4.  $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$  is the transition function,
5.  $q_0 \in Q$  is the start state,
6.  $q_{\text{accept}} \in Q$  is the accept state, and
7.  $q_{\text{reject}} \in Q$  is the reject state.

# Configurations of a TM

- As a TM computes, changes occur in
  - the current state,
  - the current tape contents, and
  - the current head location.
- A setting of these three items is called a **configuration** of the TM.
- We write  $uqv$  to denote the configuration where
  - the current state is  $q$ ,
  - the current tape contents is  $uv$ , and
  - the current head location is the first symbol of  $v$ .(The tape contains only blanks following the last symbol of  $v$ .)

# Configurations of a TM (cont.)







**FIGURE 3.4**  
A Turing machine with configuration  $1011q_701111$

Source: [Sipser 2006]



# Configurations of a TM (cont.)

-   $q_0w$  is the *start configuration* on input  $w$ .
-   $uq_{\text{accept}}v$  is an *accepting configuration*.
-   $uq_{\text{reject}}v$  is a *rejecting configuration*.
-  Accepting and rejecting configurations are *halting configurations*.

# Computation of a TM

Configuration  $C_1$  *yields* configuration  $C_2$  if the Turing machine can legally go from  $C_1$  to  $C_2$  in a single step:

1.  $u' a q_i b v'$  yields  $u' q_j a c v'$  if  $\delta(q_i, b) = (q_j, c, L)$ .
2.  $u q_i b v'$  yields  $u c q_j v'$  if  $\delta(q_i, b) = (q_j, c, R)$ .
3.  $q_i b v'$  yields  $q_j c v'$  if  $\delta(q_i, b) = (q_j, c, L)$ .
4.  $q_i b v'$  yields  $c q_j v'$  if  $\delta(q_i, b) = (q_j, c, R)$ .

( $u' a q_i$  is considered equivalent to  $u' a q_i \sqcup$ .)

# Computation of a TM (cont.)

- 🌐 A Turing machine *accepts* input  $w$  if a sequence of configurations  $C_1, C_2, \dots, C_k$  exists where
  1.  $C_1$  the start configuration on  $w$ ,
  2.  $C_i$  yields  $C_{i+1}$ , and
  3.  $C_k$  is an accepting configuration.
- 🌐 The collection of strings that  $M$  accepts is *the language of  $M$* , or *the language recognized by  $M$* , denoted  $L(M)$ .

# Decidable Languages

## Definition (3.5)

A language is **Turing-recognizable** (also called *recursively enumerable*) if some Turing machine recognizes it.

- 🌐 A Turing machine can fail to accept an input by entering the  $q_{\text{reject}}$  state and rejecting, or by looping (not halting).
- 🌐 A machine is called a *decider* if it halts on all inputs. A decider that recognizes some language is said to *decide* the language.

## Definition (3.6)

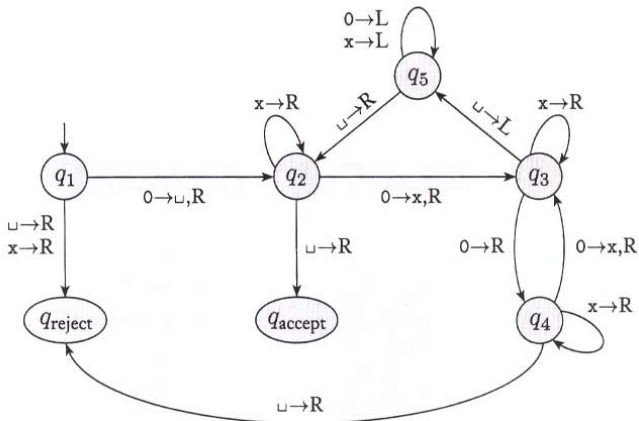
A language is **Turing-decidable**, or simply **decidable** (also called *recursive*), if some Turing machine decides it.

# Example Turing Machines

$A = \{0^{2^n} \mid n \geq 0\}$ . A decider  $M_2$  for  $A$  can be defined to work as follows:

1. Sweep left to right across the tape, crossing off every second 0.
2. If in stage 1 the tape contained a single 0, *accept*.
3. If in stage 1 the tape contained more than one 0 and the number of 0s was odd, *reject*.
4. Return head to the left-hand end of the tape.
5. Go to stage 1.

# Example Turing Machines (cont.)



**FIGURE 3.8**  
State diagram for Turing machine  $M_2$

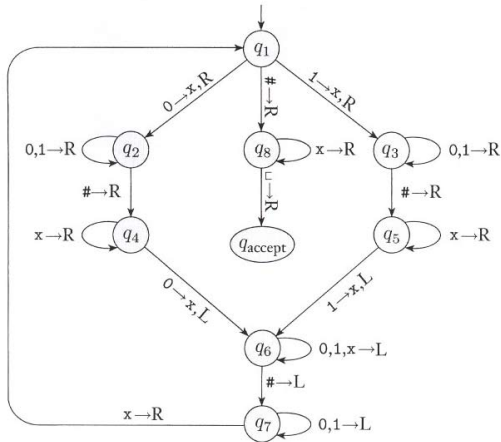
Source: [Sipser 2006]

## Example Turing Machines (cont.)

$B = \{w\#w \mid w \in \{0,1\}^*\}$ . A decider  $M_1$  for  $B$  can be defined to work as follows:

1. Scan the input to be sure that it contains a single  $\#$  symbol. If not, *reject*.
2. Zig-zag across the tape to corresponding positions on either side of the  $\#$  symbol to check whether these positions contain the same symbol. If they do not, *reject*.  
Cross off symbols as they are checked.
3. When all symbols to the left of the  $\#$  have been crossed off, check for any remaining symbols to the right of the  $\#$ . If any symbols remain, *reject*; otherwise, *accept*.

# Example Turing Machines (cont.)



**FIGURE 3.10**  
State diagram for Turing machine  $M_1$

Source: [Sipser 2006] (Note: correct, but not conforming to the previous verbal description.)



## Example Turing Machines (cont.)

$C = \{a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 1\}$ . A decider  $M_3$  for  $C$ :

1. Scan the input to be sure that it is a member of  $aa^*bb^*cc^*$  and *reject* if it isn't.
2. Return the head to the left-hand end of the tape.
3. Cross off an  $a$  and scan to the right until a  $b$  occurs. Shuttle between the  $b$ 's and  $c$ 's, crossing off one of each until all  $b$ 's are gone.
4. Restore the crossed off  $b$ 's and repeat Stage 3 if there is another  $a$  to cross off.
5. If all  $a$ 's and  $c$ 's are crossed off, *accept*; otherwise, *reject*.

## Example Turing Machines (cont.)

$$E = \{\#x_1\#x_2\#\cdots\#x_l \mid x_i \in \{0, 1\}^* \text{ and } x_i \neq x_j \text{ (for } i \neq j)\}.$$

1. Place a mark on top of the leftmost tape symbol. If that symbol was not a #, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # occurs before a blank, *accept*.
3. Compare, by zig-zagging, the two strings to the right of the marked #'s. If they are equal, *reject*.
4. Move the second mark to the next # symbol. If not doable, move the first mark to the next # to its right and the second mark to the # after that. If not doable, *accept*.
5. Go to Stage 3.

# Variants of Turing Machines

- 🌐 Alternative definitions of Turing machines abound, including versions with *multiple tapes* or with *nondeterminism*. They are called *variants* of the Turing machine model.
- 🌐 The original model and its reasonable variants all have the same power—*they recognize the same class of languages*.
- 🌐 To show that two models are equivalent, we simply need to show that we can *simulate* one by the other.

# Multitape Turing Machines

- 🌐 A *multitape Turing machine* is like an ordinary Turing machine with several tapes.
- 🌐 Each tape has its own head for reading and writing. Initially the input appears on tape 1 and the others start out blank.
- 🌐 The transition function is changed to allow for reading, writing, and moving the heads on all the tapes simultaneously. Formally,

$$\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k,$$

where  $k$  is the number of tapes.

# Multitape Turing Machines (cont.)

## Theorem (3.13)

*Every multitape Turing machine has an equivalent single-tape Turing machine.*

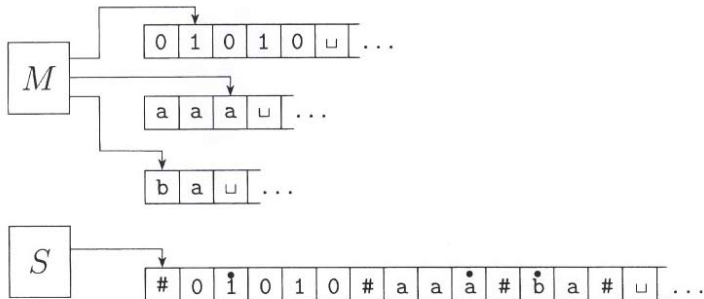
A single tape TM  $S$  can simulate a  $k$ -tape  $M$ :

1.  $S$  “formats” its tape to represent all  $k$  tapes of  $M$ :

$$\# \overset{\bullet}{w}_1 \overset{\bullet}{w}_2 \cdots \overset{\bullet}{w}_n \# \overset{\bullet}{\square} \# \overset{\bullet}{\square} \# \cdots \#$$

2. To simulate a single move of  $M$ ,  $S$  scans its tape to determine the symbols under the virtual heads. Then  $S$  makes a second pass to update the tapes according to  $M$ 's transition function.
3. Whenever a virtual head is moved to the right onto a  $\#$ ,  $S$  writes a blank symbol on this tape cell and shifts the tape contents from this cell one unit to the right.

# Multitape Turing Machines (cont.)



**FIGURE 3.14**  
Representing three tapes with one

Source: [Sipser 2006]

# Nondeterministic Turing Machines

- 🌐 A **nondeterministic Turing machine** is defined in the expected way.
- 🌐 The transition function of a nondeterministic TM has the form

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

- 🌐 The computation of a nondeterministic TM is a tree whose branches correspond to different possibilities for the machine.
- 🌐 If some branch of the computation leads to the accept state, the machine accepts its input.

# Nondeterministic Turing Machines (cont.)

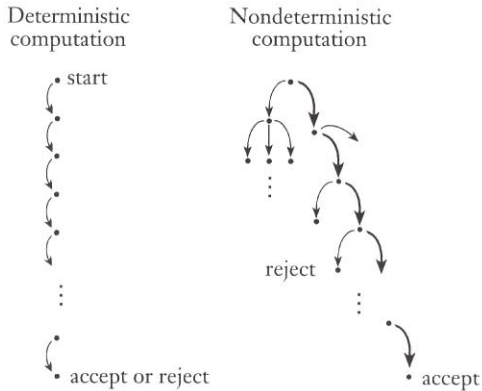


FIGURE 1.28

Deterministic and nondeterministic computations with an accepting branch

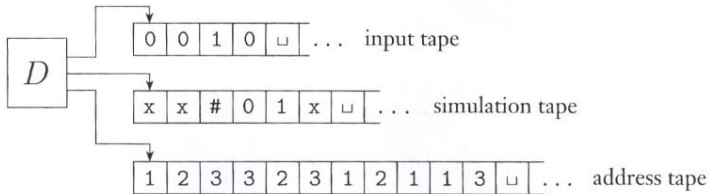


## Theorem (3.16)

*Every nondeterministic TM has an equivalent deterministic TM.*

- 🌐 The idea is to have a deterministic TM  $D$  try all possible branches of the given nondeterministic TM  $N$ 's computation.
- 🌐  $D$  searches, in a breadth first manner,  $N$ 's computation tree for an accepting configuration.




# Nondeterministic Turing Machines (cont.)



**FIGURE 3.17**  
Deterministic TM  $D$  simulating nondeterministic TM  $N$

Source: [Sipser 2006]

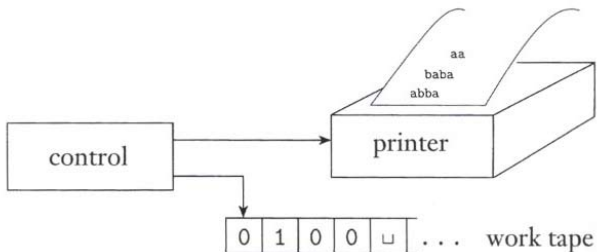
$D$  has three tapes:

-  Tape 1 always contains the input string and is never altered.
-  Tape 2 maintains a copy of  $N$ 's tape on some branch of its nondeterministic computation.
-  Tape 3 keeps track of  $D$ 's location in  $N$ 's nondeterministic computation tree.

# Enumerators

- Some people use the term **recursively enumerable** language for Turing-recognizable language.
- An *enumerator* is a Turing machine with an attached printer. Every time the Turing machine wants to add a string to the output list, it sends the string to the printer.
- The language enumerated by an enumerator  $E$  is the collection of all the strings that  $E$  eventually prints out.
- Moreover,  $E$  may generate the strings of the language in any order, possibly with repetitions.

# Enumerators (cont.)



**FIGURE 3.20**  
Schematic of an enumerator

Source: [Sipser 2006]

# Enumerators (cont.)

## Theorem (3.21)

*A language is Turing-recognizable if and only if some enumerator enumerates it.*

To recognize the language enumerated by  $E$ , a TM  $M$  works as follows:

1. Run  $E$ . Every time that  $E$  outputs a string, compare it with the input  $w$ .
2. If  $w$  appears in the output of  $E$ , *accept*.

## Enumerators (cont.)

To enumerate the language recognized by  $M$ , an enumerator  $E$  works as follows:

1. Repeat Steps 2 and 3 for  $i = 1, 2, 3, \dots$
2. Run  $M$  for  $i$  steps on each input,  $s_1, s_2, \dots, s_i$ .
3. If any computations accept, print out the corresponding  $s_j$ .

# Hilbert's Tenth Problem

- 🌐 A *polynomial* is a sum of terms, where each term is a product of variables and a constant.
- 🌐 For example,  $6x^3yz^2 + 3xy^2 - x^3 - 10$  is a polynomial with four terms over variables  $x$ ,  $y$ , and  $z$ .
- 🌐 Let  $D = \{p \mid p \text{ is a polynomial with an integral root}\}$ .
- 🌐 **Hilbert's tenth problem** (rephrased): “Is there an algorithm for determining  $D$ ?”
- 🌐 Proving that no algorithm exists for a particular task requires a precise definition of algorithm.



# Hilbert's Tenth Problem: The Original Statement

**10. Determination of the solvability of a Diophantine equation.** Given a diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.

Note: the kind of process that Hilbert looked after is “effective procedure” and is nowadays referred to as “computer algorithm” or simply “algorithm.”

# “Effective” Procedures

A procedure  $M$  is considered effective if the following hold:

1.  $M$  contains a finite number of exact instructions (each being expressed with a finite number of symbols);
2.  $M$  will, if carried out without error, always produce the desired result in a finite number of steps;
3.  $M$  can (in practice or in principle) be carried out by a human being unaided by any machinery save paper and pencil;
4.  $M$  demands no insight or ingenuity on the part of the human being carrying it out.

Note: excerpted from “The Church-Turing Thesis” of *Stanford Encyclopedia of Philosophy*.

# The Definition of Algorithm

- 🌐 All models of a general-purpose computer turn out to be at best equivalent in power to the Turing machine, as long as they satisfy certain reasonable requirements.
- 🌐 This has an important philosophical corollary: Even though there are many different computational models, *the class of algorithms that they describe is unique*.
- 🌐 The **Church-Turing thesis** says that *the intuitive notion of an algorithm corresponds to the formal definition of a Turing machine*.

# Describing Turing Machines

Three possible levels of detail:

- 🌐 A *formal description* spells out in full the Turing machine's states, transition function, and so on.
- 🌐 In an *implementation description*, we use natural language prose to describe the way that the Turing machine moves its head and the way that it stores data on its tape.
- 🌐 In a *high-level description*, we use natural language prose to describe an algorithm, ignoring the implementation model.

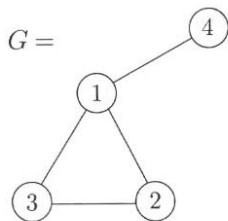
# An Example High-Level Description

Let  $A = \{\langle G \rangle \mid G \text{ is a connected undirected graph}\}$ . The following is a high-level description of a TM  $M$  that decides  $A$ :

$M =$  “On input  $\langle G \rangle$ , the encoding of a graph  $G$ :

1. Select the first node of  $G$  and mark it.
2. Repeat Step 3 until no new nodes are marked.
3. For each node in  $G$ , mark it if it is attached by an edge to a node that is already marked.
4. Scan all the nodes of  $G$  to determine whether they all are marked. If they are, *accept*; otherwise, *reject*.”

# An Example High-Level Description (cont.)



$\langle G \rangle =$

$(1, 2, 3, 4) ((1, 2), (2, 3), (3, 1), (1, 4))$

**FIGURE 3.24**

A graph  $G$  and its encoding  $\langle G \rangle$

Source: [Sipser 2006]