CHAPTER 3
The Church-Turing Thesis

Contents

• Turing Machines
  • definitions, examples, Turing-recognizable and Turing-decidable languages

• Variants of Turing Machine
  • Multitape Turing machines, non-deterministic Turing Machines, Enumerators, equivalence with other models

• The definition of Algorithm
  • Hilbert’s problems, terminology for describing Turing machines
Turing Machines (intro)

• So far in our development of theory of computation we have presented several models for computing devices
• **Finite automata** are good models for devices that have a small amount of memory.
• **Pushdown automata** are good models for devices that have an unlimited memory that is usable only in the last in, first out manner of a stack.
• We have shown that some very simple tasks are beyond the capabilities of these models.
• Now we will consider a much more powerful model, first proposed by Alan Turing in 1936, called the **Turing Machine (TM)**.
  • It is similar to a finite automaton but with an *unlimited* and *unrestricted* memory.
  • TM is much more accurate model of a general purpose computer.
  • It can do everything that a real computer can do.
  • But a TM also cannot solve certain problems.
  • There are problems that are beyond the theoretical limits of computation.
Turing Machines (informal)

- The Turing machine model uses an *infinite tape* as its unlimited memory.
- It has a *head* that can read and write symbols and move around on the tape.
- Initially the tape contains only the input string and is blank everywhere else.
- If TM needs to store information, it may write this info on the tape.
- To read the information that it has written, TM can move its head back over it.
- The machine continues computing until it produces an output.
- The output *accept* and *reject* are obtained by entering designated accepting and rejecting states.
- If it does not enter an accepting or a rejecting state, it will go on forever, never halting.

Schematic of a Turing Machine:

- The differences between finite automata and Turing machines.
  - A TM can both write on the tape and read from it.
  - The read-write head can move both to the left and to the right.
  - The tape is infinite.
  - The special states for rejecting and accepting take immediate effect.
Example

• We want to design a TM $M_1$ which accepts if its input is a member of $B$

$$B = \{w\#w : w \in \{0,1\}^*\}.$$ 

Informal description how the TM works on input string $s$.

• Scan the input to be sure that it contains a single # symbol. If not, reject.

• Zig-zag across the tape to corresponding positions on either side of the # symbol to check on whether these positions contain the same symbol. If they do not, reject. Cross off symbols as they are checked to keep track of which symbols correspond.

• 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.

$M_1$ on input 011000#011000

\[ \begin{array}{c}
011000 \\
\hline
11000 \\
\hline
# \\
\hline
011000 \\
\hline
\end{array} \]

\[ \begin{array}{c}
x11000 \\
\hline
\ldots
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]

\[ \begin{array}{c}
\ldots \\
\hline
\end{array} \]
Formal Definition of TMs

- A *Turing machine* (TM) is specified by a 7-tuple 

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

where

- $Q$ is a finite set of states,
- $\Sigma$ is a finite input alphabet not containing $\sqcup$,
- $\Gamma$ is a finite tape alphabet, such that $\sqcup \in \Gamma$, $\Sigma \subset \Gamma$,
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function,
- $q_0 \in Q$ is the start state,
- $q_{\text{accept}} \in Q$ is the accept state, and
- $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{accept}} \neq q_{\text{reject}}$.

- The heart of the definition of a TM is the transition function because it tells us how the machine gets from one step to the next.

$\delta(q, a) = (r, b, L)$ means that when the machine is in a certain state $q$ and head is over a tape square containing a symbol $a$, the machine writes the symbol $b$ replacing the $a$, and goes to state $r$. The third component is either $L$ or $R$ and indicates whether the head moves to the left or right after writing.
How does a TM compute?

• Initially TM receives its input \( w = w_1w_2...w_n \in \Sigma^* \) on the leftmost \( n \) squares of the tape, and the rest of the tape is blank.

• The head starts on the leftmost square of the tape.

• Note that \( \Sigma \) does not contain the blank symbol, so the first blank symbol appearing on the tape marks the end of the input.

• Once TM starts, the computation proceeds according to the rules described by the transition function.

• If TM ever tries to move its head to the left off the left-hand end of the tape, the head stays in the same place for that move, even though the transition function indicates \( L \).

• The computation continues until it enters either accept state or reject state at which point it halts.

• If neither occurs, TM goes on forever.

control
\[ \text{a} \quad \text{a} \quad \text{b} \quad \text{b} \quad \text{...} \]
Acceptance of Strings and the Language of TM

A configuration $C$ of the TM.

For a state $q$ and two strings $u$ and $v$ over the tape alphabet $\Gamma$ we write $uv^q$ for the configuration where the current state is $q$, the current tape contents is $uv$, and the current head location is the first symbol of $v$.

Let $a, b, c \in \Gamma$, $u, v \in \Gamma^*$, $q_i, q_j \in Q$.

We say that configuration

$uaq_i bv$ yields

\[
\begin{cases} 
uaq_j acv & \text{if } \delta(q_i, b) = (q_j, c, L) \\
uacq_j v & \text{if } \delta(q_i, b) = (q_j, c, R)
\end{cases}
\]

Note that

$q_i bv$ yields

\[
\begin{cases} 
q_j cv & \text{if } \delta(q_i, b) = (q_j, c, L) \\
cq_j v & \text{if } \delta(q_i, b) = (q_j, c, R)
\end{cases}
\]

and that $uaq_i$ is equivalent to $uaq_i \quad$ and we can handle this as before.
Acceptance of Strings and the Language of TM (cont.)

• The **start configuration** of TM on input \( w \) is \( q_0w \).

• In an **accepting configuration** the state is \( q_{\text{accept}} \).

• In an **rejecting configuration** the state is \( q_{\text{reject}} \).

\( \{ \text{Halting configurations} \} \)

• A Turing machine TM **accepts input** \( w \) if a sequence of configurations \( C_1, C_2, \ldots, C_k \) exists where
  - \( C_1 \) is the start configuration of TM on input \( w \),
  - each \( C_i \) yields \( C_{i+1} \) and
  - \( C_k \) is an accepting configuration.

• If \( L \) is a set of strings that TM accepts, we say that \( L \) is the **language of** TM and write \( L = L(TM) \).

• We say TM **recognizes** \( L \) or TM **accepts** \( L \).

• A language is **Turing-recognizable** if some TM recognizes it.

• For a TM three outcomes are possible on an input: it may **accept**, **reject** or **loop**.

• **Deciders** are TMs that always make a decision to accept or reject the input.

• A language is **Turing-decidable** or simply **decidable** if it is accepted by a decider.
Example 1.
• A TM $M_2$ which decides the language $A = \{0^{2^n} : n \geq 0\}$.

Higher-level description.

$M_2$ = “On input string $w$:
1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, accept.
3. If in stage 1 the tape contained more than a single 0 and the number of 0’s was odd, reject.
4. Return the head to the left-hand end of the tape.
5. Go to stage 1.”

Formal description.

$M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$

$Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$

$\Sigma = \{0\}$

$\Gamma = \{0, x, \sqcup\}$

Start state

Run $M_2$ on input 0000 and 000
Example 2.

- A TM $M_1$ which decides the language $B = \{w\#w : w \in \{0,1\}^*\}$. For higher-level description see slide #4.

**Formal description.**

$$M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$$

$$Q = \{q_1, ..., q_{14}, q_{\text{accept}}, q_{\text{reject}}\}$$

$$\Sigma = \{0,1,\#\}$$

$$\Gamma = \{0,1,\#, x, \sqcup\}$$
• **Example 3:** A TM solving the *element uniqueness problem*. It is given a list of strings over \{0,1\} separated by #s and its job is to accept if all strings are different. The language is

\[ E = \{ \# x_1 \# x_2 \# \ldots \# x_l : x_i \in \{0,1\}^* \; \forall i \in \{1,\ldots,l\}, \; x_i \neq x_j \; \text{for} \; i \neq j \}. \]

• TM M3 works by comparing \( x_1 \) with \( x_2 \) through \( x_l \), then by comparing \( x_2 \) with \( x_3 \) through \( x_l \), and so on.

**Higher-level description:**

M3="On input \( w \):

1. Place a mark on top of the leftmost tape symbol. If that symbol was blank, *accept*. If it was a #, continue with the next stage. Otherwise, *reject*.

2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only \( x_1 \) was present, so *accept*.

3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.

4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. If no # is available for the rightmost mark, all the strings have been compared, so *accept*.

5. Go to stage 3."

(In the actual implementation, the machine has two different symbols, \# and \#, in its tape alphabet).