Chapter 7: Turing Machines

Dr. Yuan Yao

University of Nottingham Ningbo China (UNNC)

Learning Outcomes

Learning outcomes

At the conclusion of this chapter, the students are expected to be able to:

- Describe the components of a standard Turing machine.
- State whether an input string is accepted by a Turing machine.
- Construct a Turing machine to accept a specific language.
- Trace the operation of a Turing machine transducer given a sample input string.
- Construct a Turing machine to compute a simple function.
- State Turing's thesis and discuss the circumstantial evidence supporting it.
- Understand the Turing Machine halting problem and how to prove it.

Introduction

Models of Computation

- Given the following languages, which models of computations we can use to accept them?
  - $A^nB^m = \{a^nb^m \mid n, m \geq 0\}$
  - $A^nB^n = \{a^nb^n \mid n \geq 0\}$
  - $WW^R = \{ww^R \mid w \in \{a, b\}^*\}$
  - $A^nB^nC^m = \{a^nb^nc^m \mid n, m \geq 0\}$
  - $A^nB^mC^k = \{a^nb^mc^k \mid n, m, k \geq 0\}$
  - $A^nB^nC^n = \{a^nb^nc^n \mid n \geq 0\}$
  - $WW = \{ww \mid w \in \{a, b\}^*\}$

A General Model of Computation

- Both finite automata and pushdown automata are models of computation.
- However, there are languages which are not accepted by these models:
  - An FA cannot accept languages such as:
    - $A^nB^n = \{a^nb^n \mid n \geq 0\}$
    - $WW^R = \{ww^R \mid w \in \{a, b\}^*\}$
  - An PDA cannot accept languages such as:
    - $A^nB^nC^n = \{a^nb^nc^n \mid n \geq 0\}$
    - $WW = \{ww \mid w \in \{a, b\}^*\}$
- However, these tasks can be performed by programs written in general purpose programming languages such as Java, C, or Haskell.
- Therefore, FAs and PDAs do not provide us with the most powerful computational power.
- **Question: Which model of computation (if any) can provide us with the most powerful computational power?**

A General Model of Computation

- In the first part of the 20^{th} century, the most prominent mathematicians and logicians were occupied by the previous question, which is closely related to the following:
  - **What are the limits of algorithmic thinking?**

- To address this question, Alan Turing considered a human computer working with a pen and paper.
  - In those days, the word “computer” referred to people (i.e., human beings) who would work in an office, and perform long computations, using special purpose (and at times, ingenious) computing devices.

A General Model of Computation

- Turing considered the very basic operations performed when a human computer solves a problem algorithmically, by using pen and paper.
- He postulated that the steps a (human) computer takes should include these:
  - Examine an individual symbol on the paper.
  - Erase a symbol or replace it by another.
  - Transfer attention from one symbol to a nearby one.
- Based on that, he came up with his ground-breaking idea regarding how a universal computing machine should work.
- Turing's objective was to demonstrate the inherent **limitations** of algorithmic methods.
- This is why he wanted his device to be able to execute **any** algorithm that a human computer could.

A General Model of Computation

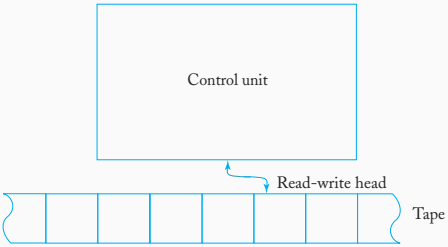
- In simple terms, a Turing machine is a finite state control unit with an unbounded tape attached as a storage device.
- Even though a Turing machine is a very simple machine, it turns out to be very powerful, and has been the basis of languages such as C, Python, Java...
- In fact, so far, no one has been able to find any other effective model of computation more powerful than the Turing machine, although many models have proven to be as powerful as that of Turing machines.
- This has led to Church-Turing Thesis, which claims that Turing machines are the most general types of automata, in principle as powerful as any computers.
  - We will discuss it later.

Turing Machine

The Standard Turing Machine

- A standard Turing machine has unlimited storage in the form of a tape:
  - The tape consists of an infinite number of cells.
  - With each cell capable of storing one symbol.
- The *read-write head* can travel in both directions on the tape, processing one symbol per move.

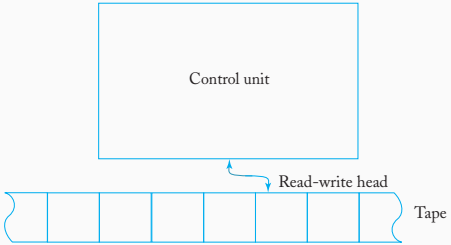
- In a Standard Turing Machine, the tape acts as the:
  - input
  - output
  - and storage medium



The Standard Turing Machine

- A control function causes the machine to change states and possibly overwrite the tape contents.
- Deterministic and non-deterministic Turing machines turn out to be as powerful as each other.

- The input string is surrounded by blanks, so the input alphabet is considered a proper subset of the tape alphabet.



Definition of a Turing Machine

- **Definition 9.1:** A Turing Machine $M = \{Q, \Sigma, \Gamma, \delta, q_0, \square, F\}$ is defined by:
  - Q : a finite set of internal states.
  - Σ : the input alphabet.
  - Γ : the tap alphabet.
  - $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$: the transition function.
  - $\square \in \Gamma$: a special symbol called the blank.
  - $q_0 \in Q$: the initial state.
  - $F \subseteq$ the set of final states.

- In the definition of a Turing machine, we assume that:

$$\Sigma \subseteq \Gamma - \{\square\}$$

Transition function δ

$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

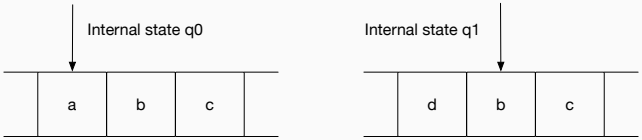
- Input to the transition function δ consists of:
  - the current state of the control unit.
  - and the current tape symbol
- Output of δ consists of :
  - a new state
  - a new tap symbol
  - and location of the next symbol to be read (left or right)
- In general, δ is a partial function, so on some (state, symbol) input combinations it may be undefined.

Example: δ transition

- The following is a sample transition rule in a Turing machine:

$$\delta(q_0, a) = (q_1, d, R)$$

- According to the rule, when:
  - the control unit is in state q_0,
  - and the tape symbol is a
- then, the effect of the transition is as follows:
  - the new state will be q_1,
  - the symbol d replaces a on the tape,
  - and the read-write head moves one cell to the right.



Example: Turing Machine

- Consider the Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ with:

$$Q = \{q_0, q_1\}, \Sigma = \{a, b\}, \Gamma = \{a, b, \square\}, F = \{q_1\}$$

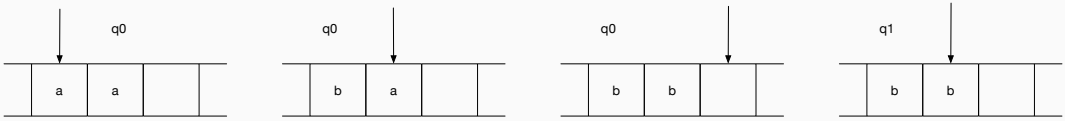
- The transition function δ is given by:

$$\begin{aligned} \delta(q_0, a) &= (q_0, b, R) \\ \delta(q_0, b) &= (q_0, b, R) \\ \delta(q_0, \square) &= (q_1, \square, L) \end{aligned}$$

- Can you find out what this TM does?

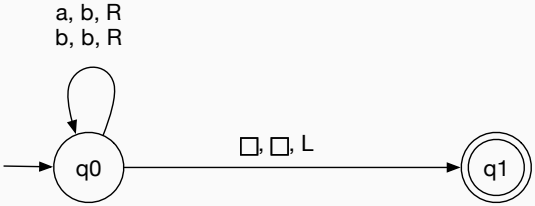
Example: Turing Machine

- The machine starts in q_0 and, as long as it reads a's, it will replace them with b's and continue moving to the right, but b's will not be modified.
- When a blank is found, the control unit switches states to the final state q_1 and moves one cell to the left.
- The followings show a sequence of moves for the given Turing machine with the initial content aa:



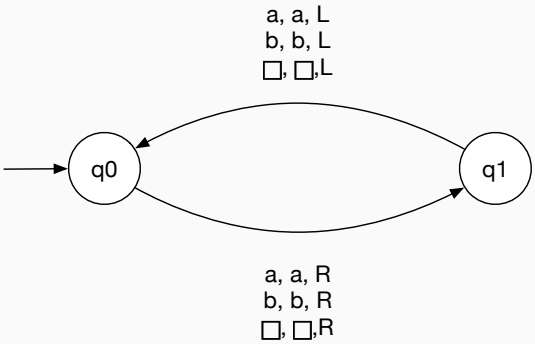
Transition Graphs for Turing Machines

- As before, we can use transition graphs to represent Turing machines.
- We label the edges of the graph with three items:
  - the current tape symbol,
  - the symbol that replaces it,
  - and the direction in which the read-write head is to move.
- The Turing machine in the previous slides can be represented as follows:



Another Example of Turing Machines

- Given the following transition graph of a Turing Machine, what happens if the input string is *ab* (starting from the initial state q_0)?

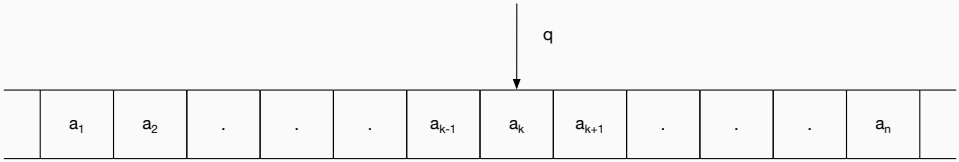


Another Example of Turing Machines

- A Turing machine is said to halt whenever it reaches a configuration for which δ is not defined.
- It is possible for a Turing machine to never halt on certain inputs.
- It should be clear that this machine will run forever, regardless of the initial information on its tape, with the read-write head moving alternately right then left but making no modifications to the tape.
- In analogy with programming terminology, we say that the Turing machine is in an infinite loop.

Instantaneous Description

- Here, as in the case of PDAs, the most convenient way to exhibit a sequence of configurations of a Turing machine is based on the idea of an *instantaneous description*.
- Any configuration is completely determined by:
  - the current state of the control unit,
  - the contents of the tape,
  - and the position of the read-write head.



Instantaneous Description

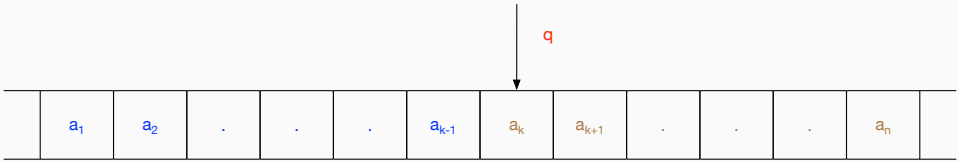
- A tape can be divided into three parts:
  - The current symbol (or cell), e.g., a_k.
  - All cells to the left, e.g., $a_1 a_2 \dots a_{k-1}$.
  - All cells to the right, e.g., $a_{k+1} \dots a_n$.

- An instantaneous description is in the form of $x_1 q x_2$, where
  - $x_1 = a_1 \dots a_{k-1}$
  - q is the current state
  - $x_2 = a_k \dots a_n$

Instantaneous Description

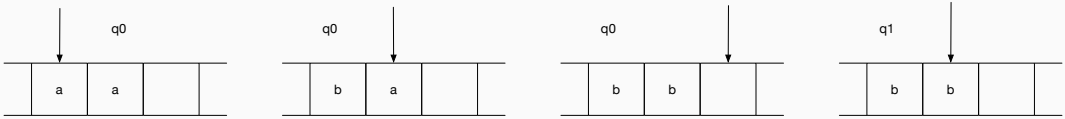
- An instantaneous description of a machine in state q with the tape depicted in the figure below is as follows:

$$a_1 a_2 \ldots a_{k-1} q a_k a_{k+1} \ldots a_n$$



Instantaneous Description

- Write down the instantaneous description for the following Turing machine state.



The Language Accepted by a Turing Machine

- Turing machines can be viewed as language accepters.
- **Definition** Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ be a Turing Machine. Then the languages accepted by M is:

$$L(M) = \{w \in \Sigma^* \mid q_0 w \vdash^* x_1 q_f x_2\}$$

  where $q_f \in F$ and $x_1, x_2 \in \Gamma^*$
- The language accepted by a Turing machine is the set of all strings which cause the machine to halt in a final state, when started in its standard initial configuration.

The Language Accepted by a Turing Machine

- A string is rejected by M if:
  - The machine halts in a non-final state, or
  - The machine never halts.

- M is also said to halt starting from some initial configuration $x_1 q_i x_2$ if

$$x_1 q_i x_2 \vdash^* y_1 q_j a y_2$$

  for any q_j and a, for which $\delta(q_j, a)$ is undefined.

The Language Accepted by a Turing Machine

- In the textbook, only the non-empty strings $w \in \Sigma^+$ are allowed. This has been done only for technical reasons to make some subsequent arguments easier and less tedious. We consider the more general case of $w \in \Sigma^*$, i.e., we allow the empty string as well.
- Note that, by allowing $w \in \Sigma^*$, we have excluded any string which included blank symbols.

- The previous definition indicates that the input w is written on the tape with blanks on either side.
- Why we exclude blanks from the input?

The Language Accepted by a Turing Machine

- The reason for excluding blanks from the input now becomes clear:
  - It assures us that all the input is restricted to a well-defined region of the tape, bracketed by blanks on the right and left.
  - Without this convention, the machine could not limit the region in which it must look for the input; no matter how many blanks it saw, it could never be sure that there was not some nonblank input somewhere else on the tape.

Example: Languages Accepted by Turing Machines

- Consider the Turing Machine $M = (\{q_0, q_1\}, \{0, 1\}, \{0, 1, \square\}, \delta, q_0, \square, F)$ where δ is defined as follows:

$$\begin{aligned} \delta(q_0, 0) &= (q_0, 0, R) \\ \delta(q_0, \square) &= (q_1, \square, R) \end{aligned}$$

- What is the language $L(M)$?

Example: Languages Accepted by Turing Machines

- Turing machines are more powerful than the previous classes of automata that we have studied (i.e., FAs and PDAs).

- Construct Turing machines for the following languages:
  - $A^n B^m$: which is regular.
  - $A^n B^n (n \geq 1)$: which is not regular but context-free.
  - $A^n B^n C^n (n \geq 1)$: which is not context-free.

Example: Languages Accepted by Turing Machines

- $A^nB^n (n \geq 1)$: which is not regular, but context-free.
  1. Find the leftmost a, replace it with some new symbol.
  2. Find the leftmost b, replace it with another symbol.
  3. Repeat step 1 and step 2.
  4. If the above repetitive process halts, and no a's and b's left, then the string is accepted. Otherwise, the string is rejected.

Example: Languages Accepted by Turing Machines

- $A^n B^n C^n (n \geq 1)$: which is not context-free.
- This question can be solved easily using the previous idea, however, the actual program is tedious.

Turing Machines as Transducers

Turing Machines as Transducers

- So far, we have focused on machines as language accepters, because in formal languages, accepters are quite adequate.
- On the other hand, the primary purpose of a computer is to transform input into output, i.e., it acts as a transducer.
- If we want to model computers using Turing machines, we must look at this aspect more closely.
- A *Turing machine transducer* implements a function that treats the original contents of the tape as its input and the final contents of the tape as its output.

Turing Machines as Transducers

- A function is *Turing-computable* if it can be carried out by a Turing machine capable of processing all values in the function domain.
- **Definition:** A function f with domain D is said to be Turing-computable if there exists some Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ such that, for all $w \in D$:

$$q_0 w \vdash^* q_f f(w)$$

for any input $w \notin D$, the machine M does not halt in a final state.

Turing Machines as Transducers

- Turing machines are the most powerful model of computation as transducers as well.

- In fact, all the common mathematical functions are Turing-computable, e.g.:
  1. Arithmetic operators, Exponentiation, Integer logarithm;
  2. Comparison;
  3. String manipulation;
  4. Etc.

Example: Turing Machine as Transducers

- Given two positive integers x and y in unary notation, i.e., $\{1\}^+$, separated by a single zero, the Turing machine below computes $x + y$ which ends with a single zero:
  - For example, 111111011 means $6 + 2$, and the result should be 111111110
  - The transducer $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, \square\}, \delta, q_0, \square, \{q_4\})$ where δ is as follows:

$$\begin{aligned}
\delta(q_0, 1) &= (q_0, 1, R), \quad \delta(q_0, 0) = (q_1, 1, R) \\
\delta(q_1, 1) &= (q_1, 1, R), \quad \delta(q_1, \square) = (q_2, \square, L) \\
\delta(q_2, 1) &= (q_3, 0, L), \quad \delta(q_3, 1) = (q_3, 1, L) \\
\delta(q_3, \square) &= (q_4, \square, R)
\end{aligned}$$

- Derive the result for the input string “111011” by using the instantaneous description.

Example: Turing Machine as Transducers

- Derive the result for the input string “111011” by using the instantaneous description:

$$
\begin{aligned}
q_0111011 &\vdash 1q_011011 \vdash 11q_01011 \vdash 111q_0011 \vdash 1111q_111 \\
&\vdash 11111q_11 \vdash 111111q_1\square \vdash 11111q_21 \vdash 1111q_310 \\
&\vdash 111q_3110 \vdash 11q_31110 \vdash 1q_311110 \vdash q_3111110 \\
&\vdash q_3\square111110 \vdash q_4111110
\end{aligned}
$$

Example: Turing Machine as Transducers

- Design a Turing machine that copies strings of 1's. More precisely, find a machine that performs the computation

$$q_0 w \vdash q_f ww$$

  for any $w \in \{1\}^*$

- To solve the problem, we implement the following intuitive process:
  1. Replace every 1 by an x.
  2. Find the rightmost x and replace it with 1.
  3. Travel to the right end of the current nonblank region and create a 1 there.
  4. Repeat Steps 2 and 3 until there are no more x's.

- **Practice:** Implement a TM to solve this problem.

Combining Turing Machines

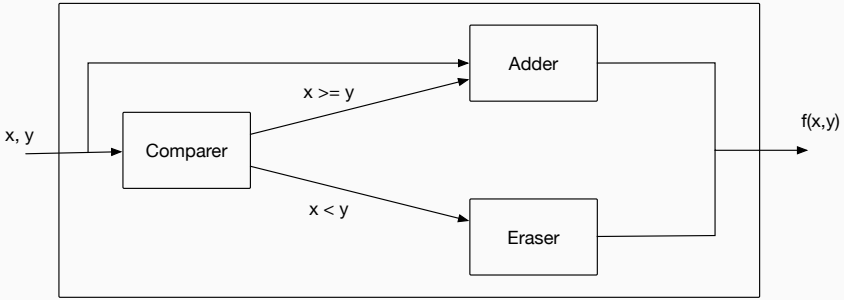
- By combining Turing Machines that perform simple tasks, complex algorithms can be implemented.
- For example, assume the existence of:
  1. a Turing machine to compare two numbers (comparer),
  2. one to add two numbers (adder),
  3. and one to erase the input (eraser)
- Design a Turing machine that computes the function:

$$f(x,y) = \begin{cases} x+y & \text{if} \quad x \geq y \\ 0 & \text{if} \quad x < y \end{cases}$$

Combining Turing Machines

- Design a Turing machine that computes the function:

$$f(x,y) = \begin{cases} x+y & \text{if } x \geq y \\ 0 & \text{if } x < y \end{cases}$$



Universal Turing Machines

- The TMs that we have studied so far have been special-purpose computers capable of executing a single algorithm.

- We can consider a “universal” Turing machine, which can execute a program stored in its memory:
- It receives an input string that specifies both:
  1. the algorithm it is to execute
  2. and the input that is to be provided to the algorithm.

- Think of your personal computer:
  1. Instead of buying a separate computer for each software, we would rather have one machine that can run any arbitrary software.
  2. We encode the software in 0s and 1s (i.e., binary format) and provide it to our machines.

The Church-Turing Thesis

The Church-Turing Thesis

- To say that the TM is a general model of computation implies that:
  - "any algorithmic procedure that can be carried out at all, by a human computer or a team of humans or an electronic computer, can be carried out by a TM"

- The statement was formulated by Alonzo Church in the 30's.
  - It is referred to as Church's thesis or the Church-Turing thesis.
  - **It isn't a statement that can be proved**.
  - But there is a lot of evidence for it.

- An acceptance of the Church-Turing Thesis leads to a definition of an algorithm:
  - An algorithm for a function $f : D \to R$ is a Turing machine M, which given any $d \in D$ on its tape, eventually halts with the correct answer $f(d) \in R$ on its tape.

The Church-Turing Thesis

- The nature of the model makes it seem that a *TM* can execute any algorithm a human can.
- Apparent enhancements to the TM have been shown not to increase its power.
- Other models of computation proposed have either been less powerful or equivalent to Turing machines.

The Church-Turing Thesis

- No one has ever suggested any kind of **effective** computation that cannot be implemented on a TM:
  - Pay attention to the word “effective”.
  - For those who are interested, see (e.g.):
    - https://en.wikipedia.org/wiki/Hypercomputation
    - Hava T. Siegelmann, “Neural and Super-Turing Computing”, Minds and Machines 13: 103–114, 2003.

- From now on, we will consider that, by definition, an “algorithmic procedure” is what a Turing machine can do.

Computability and Decidability

- **Definition:** a function f on a certain domain is said to be computable if there exists a Turing machine that computes the value of f for all arguments in its domain.
- A function is uncomputable if no such Turing machine exists.
- If we simplify the problem to “yes” or “no” questions, then we talk about a problem being **decidable** or **undecidable**.
- We say that a problem is **decidable** if there exists a Turing machine that gives the correct answer for every statement in the domain of the problem.

The Turing Machine Halting Problem

- Given the description of a Turing machine M and an input w, does M, when started in the initial configuration, perform a computation that eventually halts?
- This is a famous problem that cannot be solved by TM.
- How to prove it?

The Turing Machine Halting Problem

Proof by contradiction:
Assume we have a procedure (TM) *H* which takes a program (TM) *P* as its input and answers true if *P* halts on all its possible input and false otherwise. Given the procedure *H*, we could construct another program *H'* such that:

```
Program H'()
      If H(H') then
            Loop
      else
            Halt;
```

The Turing Machine Halting Problem

```
Program H'()
      If H(H') then
            Loop
      else
            Halt;
```

If H' will terminate on all its input, *i.e.*, $H(H')$ returns true, then H' enters an infinite loop. If H' will not terminate, then $H(H')$ returns false, which will then lead H' to a halting state. Contradiction.

The Chomsky Hierarchy

Type	Languages	Form of productions	Accepting Device
3	Regular	$A \to aB, A \to \lambda$	Finite Automata
2	Context-Free	$A \to \alpha$	Pushdown Automata
1	Context-Sensitive	$\alpha \to \beta$, with $\lvert\beta\rvert \geq \lvert\alpha\rvert$	Linear Bounded Automata
0	Unrestricted	$\alpha \to \beta$	Turing Machine

Table 1: The Chomsky Hierarchy

The Chomsky Hierarchy