Chapter 4 : Turing Machine

INCOMPLETE CHA…… NEEDS TO BE Update

Neither finite automata nor pushdown automata can be regarded as truly general models for computers, since they are not capable of recognizing even such simple languages as $\{a^n b^n c^n : n \geq 0\}$.

Turing machines are abstract devices by means of which we can formalize any computation. They stand at the heart of the theory of computation.

Turing machines are finite state automata.
The general setting is the same as with all FSAs :
- a control unit,
- a tape
- a head that reads form or writes onto the tape
- the control unit has states and a transition function to move from one state to another there is one initial state and a set of halting states

The difference:
The head can write on the tape
The head can move in both directions - right and left.
The tape:

- Bound to the left, infinite to the right.
- Consists of cells (also called squares).
- Leftmost cell contains a special symbol (Δ), to indicate the beginning of the tape.
- Each cell except the leftmost may contain a symbol from the alphabet, or a blank (#).
- Two symbols are used to denote the movement of the head to the left: ←, and to the right: →.

Turing machines are designed to satisfy simultaneously these three criteria:
(a) They should be automata; that is, their construction and function should be in the same general spirit as the devices previously studied.
(b) They should be as simple as possible to describe, define formally, and reason about.
(c) They should be as general as possible in terms of the computations they can carry out.

A Turing machine consists of a finite control, a tape, and a head that can be used for reading or writing on that tape.
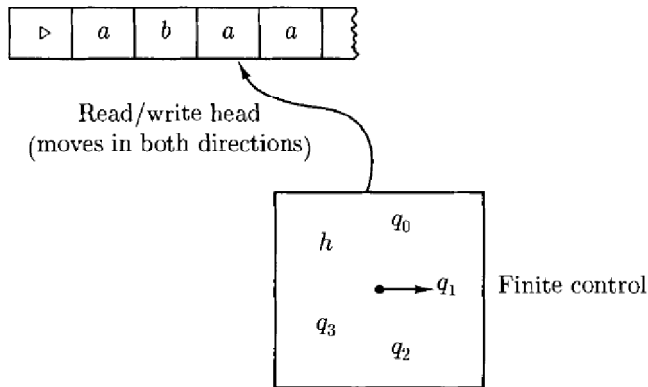
Communication between the two is provided by a single head, which reads symbols from the tape and is also used to change the symbols on the tape.

The control unit operates in discrete steps; at each step it performs two functions in a way that depends on its current state and the tape symbol currently scanned by the read/write head:

(1) Put the control unit in a new state.

(2) Either:

(a) Write a symbol in the tape square currently scanned, replacing the one already there; or

(b) Move the read/write head one tape square to the left or right.

The tape has a left end marked by special symbol denoted by Δ, but it extends infinitely to the right. We will use distinct symbols $\leftarrow$, and $\rightarrow$ to denote the head movement to left or right.

Formal definition

A Turing Machine is a quintuple $(K, \sum, \delta, s, H)$, where:
- K - a finite set of states
- $\sum$ - an alphabet. $\sum$ contains the blank symbol $\square$ and the left end symbol Δ, but does not contain $\leftarrow$ and $\rightarrow$.
- $s \in K$ - the initial state
- $H \subseteq K$ - the set of halting states
- δ - the transition function. This is a function from $(K - H) \times \sum$ to $K \times (\sum È \{\leftarrow, \rightarrow\})$ such that
  a. for all $q \in K - H$, if $\delta(q, \triangleright') = (p, b)$ then $b = \rightarrow$
  If the head is at the left end of the tape, the only possible action is to move one position to the right.
  b. for all $q \in K - H$, and $a \in \sum$, if $\delta(q, a) = (p, b)$ then $b \neq \triangleright'$.
  It is not allowed to write on the tape the left end symbol. Its position is fixed to be the leftmost position of the tape.

The meaning of δ(q , a) = (p, b):
If M (the machine) is in some state *q*, and the head reads an *a* from the input tape, then M enters state p and performs one of the following:

- if *a* is the left end symbol ▷, then M moves its head one position to the right (i.e. b = →)
- if b ε ∑ then M writes b on the tape, however it is not allowed to write ▷
- if b = → then M moves its head to the right, preparing to read the next symbol from the tape
- if b = ← then M moves its head one position to the left, preparing to read the next symbol from the tape

δ is not defined for the halting states. If M enters a halting state, then the operation stops.

Configurations and computations

Formally, the operation of M is described in terms of its configurations.
Configuration: determined by the current state, the state of the tape, and the position of the head.
e.g . $(q_1, ▷ \# a)$ is a configuration. (written informally)
Formal definition of a configuration:
Let M = (K, ∑, δ, s, H) be a Turing machine. A configuration of M is a member of
K x ▷∑* x (∑* (∑ - {#}) ∪ {e})
When M is in a given configuration C1 and we apply the transition function, another configuration C2 is obtained.
We say that C1 immediately yields C2 , C1 ├M C2
Initial **configuration:** initial state, contents of the tape, the head may be in any position
Halting configuration: halting state, any contents of the tape, any position of the head
Computation: defined as a sequence of configurations, each one immediately yielding the next, except for the last configuration

Example
The erasing machine:
Reads a symbol a from the tape, replaces the symbol with a blank, and moves to the next symbol. This is repeated until the next symbol to be read is a blank. Then M stops.

The alphabet would be: ∑ = {a, #, ▷}

How many states?

q_0 - initial state, where M reads an a.
q_1 - where M erases the a.

h - a halting state

Transition function:

state (q)	symbol (σ)	Transition δ(q , σ)
q0	a	(q1, #)
q0	#	(h, #)
q0	▷·	(q0, →)
q1	a	(q0, a)
q1	#	(q0, →)
q1	Δ	(q1, →)

Tracing the operation when M is in q0, and the input tape is: Δ a a □

(The underline shows the position of the head)

1. The first thing to be done is to move the head one position to the right.

Rule:

| q0 | ▷· | (q0, →) |

Result: ▷· <u>a</u> a #

2. When in q0 and a on the tape, M enters q1 and replaces a with a blank

Rule:

| q0 | a | (q1, #) |

Result: ▷· <u>#</u> a #

3. When in q1 and blank on the tape, M enters q0 and moves the head one position to the right

Rule:

| q1 | □ | (q0, →) |

Result: Δ □ <u>a</u> □

4. When in q0 and **a** on the tape, M enters q1 and replaces **a** with a blank

Rule:

| q0 | a | (q1, □) |

Result: Δ □ <u>□</u> □

5. When in q1 and blank on the tape, M enters q0 and moves the head one position to the right

Rule:

| q1 | □ | (q0, →) |

Result: Δ □ □ <u>□</u>

6. When in q0 and **blank** on the tape, M enters h (the halting state)

| q0 | □ | (h, □) |

The operation stops.

Note that some of the transitions are never to be performed, but they are specified because the transition relation has to be a function, otherwise the transition will not be a function.

1. **Problem**

Consider a machine that scans to the left until it finds a blank, and then stops

$\sum = \{a, \square, \Delta\}$

$K = \{q0, h\}$

$H = \{h\}$

$s = q0$

Transition function:

state q	symbol σ	Transition δ(q , σ)
q0	a	(q0, ←)
q0	□	(h, □)
q0	Δ	(q0, →)

Trace the operation if **M** is in **q0,** and the tape is Δ **a a** □

Will the machine stop? (the answer is: no)

Exam-like questions and problems

1. Describe informally what a Turing machine is
2. Give the formal definition of a Turing machine
3. What are the differences between simple FSAs and a Turing machine?
4. What is the only possible action when the head is at the left end symbol of the tape?
5. Is it allowed to write the left end symbol on the tape?

Computing with Turing Machines

Introduction

FSAs : Stop when the input has been consumed. If the state is final - the string is accepted, if not - the string is rejected.

Turing machines: the tape is infinite.

Halting states - to stop the machine.

Two types of halting states: y ('yes') and n ('no')

If the machine arrives at a halting state 'y', the string is accepted.

If the machine arrives at a halting state 'n', the string is rejected.

(Note that the machine may never arrive at a halting state)

Definition: Let M = (K, $\sum$, δ, s, H) be a Turing machine. such that H = {y,n} consists of two distinguished halting states.

Accepting configuration: Any halting configuration whose state component is "y"

Rejecting configuration: Any halting configuration whose state component is "n"

M accepts an input w ϵ (∑ - {□, Δ})* if (s, Δ□w) yields an accepting configuration.

M rejects an input w ϵ (∑ - {□, Δ})* if (s, Δ□w) yields a rejecting configuration.

Recursive and recursively enumerable languages

There are three possible outcomes of executing a Turing machine over a given input. The Turing machine may

- Halt and accept the input
- Halt and reject the input
- Never halt

Definition: A Turing machine decides a language L if for any string the following is true:

- If the string is in L, the machine accepts it.
- If the string is not in L, the machine rejects it.

Definition: A language L is called recursive, if there is a Turing machine that decides it,

i.e. accepts every string of the language and rejects every string (over the same alphabet) that is not in the language.

Theorem: The class of recursive languages is closed under complement.

Given a Turing machine and a language L, it might or it might not decide L, and there is no obvious way to tell whether it does.

Definition: A Turing machine semidecides (recognizes) a language if the following is true:

- If the string is in L, the machine accepts it.
- If the string is not in L, the machine either rejects it or never halts.

Definition: A language is called recursively enumerable, if there is a Turing machine that semidecides it.

(Strings in the language are accepted, strings that are not in the language may be rejected or may cause the Turing machine to go into an infinite loop.)

Each Turing machine M defines a recursively enumerable language: the set of all strings accepted by M.

Thus there is a class of languages, such that given a string we can tell whether it is in the language or not - these are the recursive languages.

There is another class of languages, such that given a string that is in the language, we can determine this fact. If the string is not in the language we can say nothing - the Turing machine never halts. These are the recursively enumerable languages.

All recursive languages are recursively enumerable - we can always make the rejecting state 'n' a nonhalting state from which the machine never halts.

However, there are recursively enumerable languages that are not recursive, as shown in the next section.

Theorem: If a language and its complement are both recursively enumerable, then both are recursive.

Proof:

Suppose a language L is recursively enumerable. That means there exists a Turing machine T1 that, given any string of the language, halts and accepts that string.

Now let's also suppose that the complement of L, -L = {w: w Ï L}, is recursively enumerable. That means there is some other Turing machine T2 that, given any string of -L, halts and accepts that string.

Clearly, any string (over the appropriate alphabet) belongs to either L or -L. Hence, any string will cause either T1 or T2 (or both) to halt. We construct a new Turing machine that emulates both T1 and T2, alternating moves between them. When either one stops, we can tell (by whether it accepted or rejected the string) to which language the string belongs.

Thus, we have constructed a Turing machine that, for each input, halts with an answer whether or not the string belongs to L. Therefore L and -L are recursive languages.