Theory of Computation

Introduced By:

Prof. Safia Abbas

<u>Text Books:</u>

"***An Introduction to the theory of computation***" by Michael Sipser, 2nd Edition, PWS Publishing Company, Inc. 2006; ISBN: 0-534-95097-3

"***An Introduction to Formal Languages & Automata***" by Peter Linz, 5th Edition, Jones & Bartlett Publishers, Inc. January 2012; ISBN:978-1-4496-1552-9 .

Greibach Normal Form

Definition: **A context-free grammar is in Greibach normal form if all productions are of the form $A \rightarrow ax$, where $a \in T$ and $x \in N^*$.**

Example : $S \rightarrow AB, A \rightarrow aA|bB|b, B \rightarrow b$ is not GNF,

$S \rightarrow aAB|\ bBB|\ bB, A \rightarrow aA|bB|b, B \rightarrow b$ is GNF

Example: Convert $S \rightarrow abSb|aa$ into GNF.

$S \rightarrow aBSB|\ aA, A \rightarrow a, B \rightarrow b$

Theorem: Any context-free grammar with no λ in L(G) has an equivalent grammar G` in GN form.

Pushdown Automata for CFG

Construct a npda that accepts the language generated by grammar with productions

$$S \rightarrow aSbb|a$$

- First we will transform the grammar to the GNF with **λ-free**

$$S \rightarrow aSA|a$$
$$A \rightarrow bB$$
$$B \rightarrow b$$

- The corresponding automaton will have 3 states $\{q_0,q_1,q_f\}$.
- The start symbol S is put in stack by **$\delta(q0, \lambda, z) = \{(q1, Sz)\}$.**
- The derivation rule for final state will be **$\delta(q1, \lambda, z) = \{(qf, \lambda)\}$.**
- The production **$S \rightarrow aSA|a$** will be simulated by removing S and push the SA into the stack or removing S at all.

$$\delta(q1, a, S) = \{(q1, SA), (q1, \lambda)\}$$

- In an analogous manner, the other productions give

$$\delta(q1, b, A) = \{(q1, B)\}$$
$$\delta(q1, b, B) = \{(q1, \lambda)\}$$

Pushdown Automata for CFG

Find the PDA for the grammar:

$S \to aA$,
$A \to aABC | bB | a$,
$B \to b$, $C \to c$

$$\begin{aligned}(q_0, aaabc, z) &\vdash (q_1, aaabc, Sz)\\ &\vdash (q_1, aabc, Az)\\ &\vdash (q_1, abc, ABCz)\\ &\vdash (q_1, bc, BCz)\\ &\vdash (q_1, c, Cz)\\ &\vdash (q_1, \lambda, z)\\ &\vdash (q_f, \lambda, z)\,.\end{aligned}$$

$\delta(q_0, \lambda, z) = \{(q_1, Sz)\}$
$\delta(q_1, \lambda, z) = \{(q_f, \lambda)\}$

$\delta(q_1, a, S) = \{(q_1, A)\}$

$\delta(q_1, a, A) = \{(q_1, ABC), (q_1, \lambda)\}$
$\delta(q_1, b, A) = \{(q_1, B)\}$

$\delta(q_1, b, B) = \{(q_1, \lambda)\}$

$\delta(q_1, c, C) = \{(q_1, \lambda)\}$

Context-free grammars and Pushdown Automata

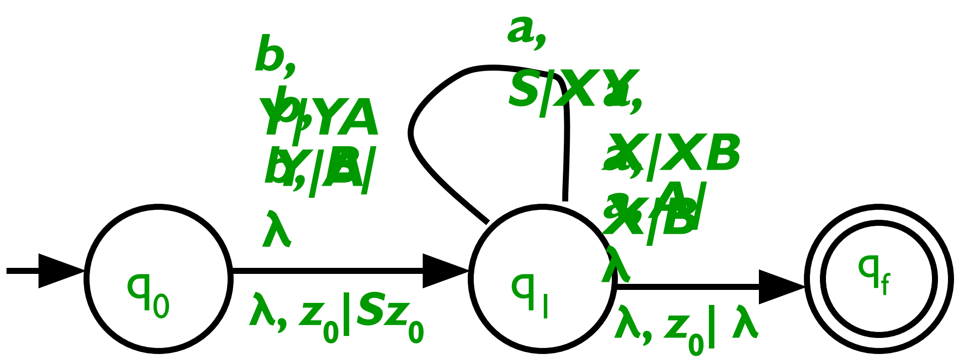
What is going to happen if $L=\{a^{n+1}b^nb^ma^m \mid m, n >= 0\}$

Consider the grammar

- $S \rightarrow aXY$
- $X \rightarrow aXB \mid aB$
- $Y \rightarrow bYA \mid bA$
- $A \rightarrow a$
- $B \rightarrow b$

$L(G) = \{a^{n+1}b^{n+m}a^m \mid m, n > 0\}$

Consider aaabbba



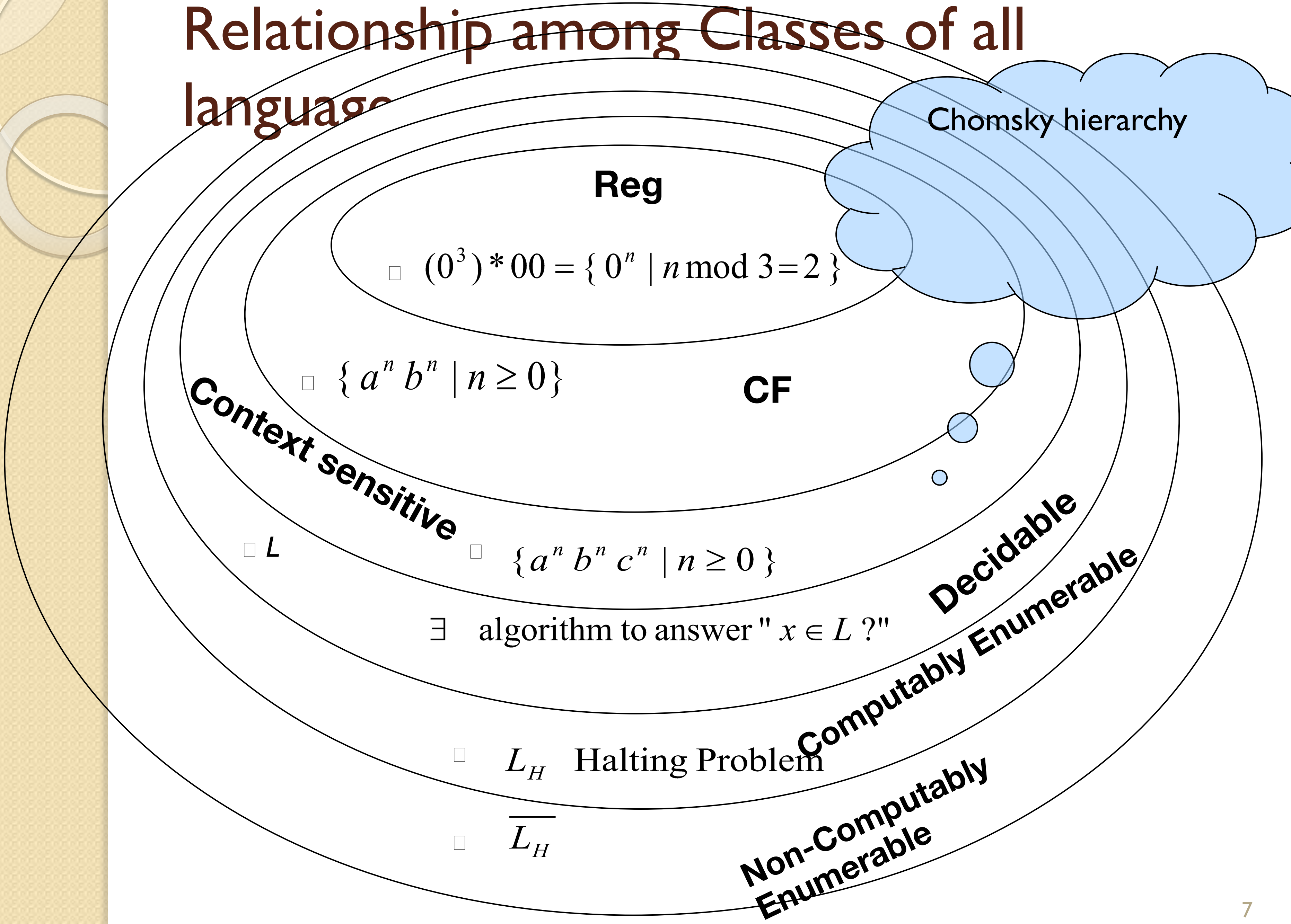
Limitations of CFL

- Not all languages can have context free grammar.
- Example

  Show that the language $L = \{a^nb^nc^n : n >= 0\}$ is not CFL

In order to prove this we use Pumping lemma2, but it is out of our scope.

Relationship among Classes of all language


Chomsky hierarchy
Reg
$(0^3)^*00 = \{ 0^n \mid n \bmod 3 = 2 \}$
$\{ a^n b^n \mid n \geq 0 \}$
CF
Context sensitive
L
$\{ a^n b^n c^n \mid n \geq 0 \}$
$\exists$ algorithm to answer "$x \in L$?"
Decidable
Computably Enumerable
L_H Halting Problem
Non-Computably Enumerable
$\overline{L_H}$

Where do we go from here?

- From the earlier example, we see that not all languages are context-free.
- Is there a mathematical model for even more complex computations?
- There are many.
- We will study one in particular: an automaton called a ***Turing machine***.

Church-Turing thesis

- The definition came in the 1936 papers of ***A. Church*** and ***A. Turing***.
- Church used a notational system called λ-calculus to define algorithms.
- Turing did it with his ‘machines’.
- These two definitions were shown to be equivalent.
- This connection between the informal notion of algorithm and the precise definition has come to be called the ***Church-Turing thesis***.

Intuitive notion of algorithms	equals	Turing machine algorithms

- This thesis provides the definition of algorithm necessary to resolve Hilbert’s tenth problem.

Definition of algorithm

- Hilbert's problems:

  In 1900, mathematician *David Hilbert* delivered a famous address at the "International congress of Mathematicians in Paris."

  He proposed 23 mathematical problems.

- Hilbert's tenth problem:

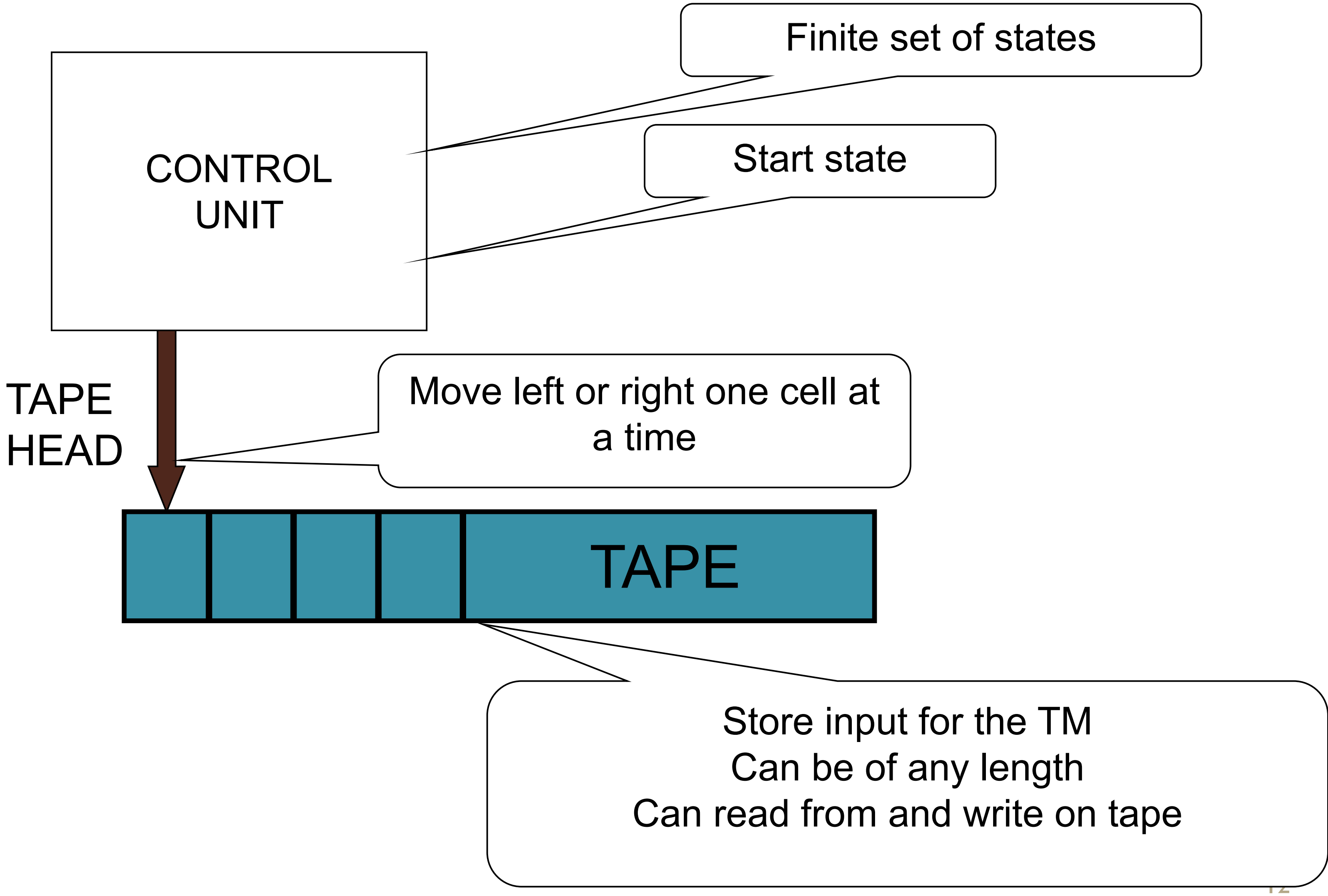
  Design an algorithm "a process according to which it can be determined by a finite number of operations" that tests whether a multi-variable polynomial has an integral root.

Hilbert's tenth problem

it is algorithmically unsolvable.

- A ***polynomial*** is a sum of terms, where each ***term*** is a product of certain variables and a constant called a ***coefficient.***
- $6 \cdot x \cdot x \cdot x \cdot y \cdot z \cdot z = 6x^3yz^2$ is a term with coefficient 6.
- $6x^3yz^2 + 3xy^2 - x^3 - 10$ is a polynomial with four terms over the variables *x,y,* and *z.*
- A ***root*** of a polynomial is an assignment of values to variables so that the value of the polynomial is 0. That polynomial has a root *x=5, y=3, z=0.*
- This root is ***integral*** since all the variables are assigned integer values.
- Some polynomials have an integral root and some do not.
- So, Hilbert's tenth problem was to devise an algorithm that tests whether a polynomial has an integral root.
- Proving that an algorithm does not exist requires having a clear definition of algorithm. Progress on the tenth problem had to wait for that definition.

Structure of TM

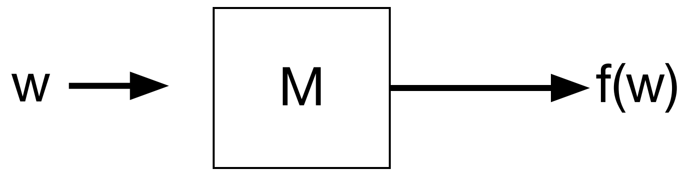


What does a TM do?

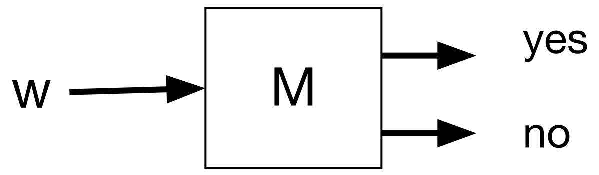
- Determine if an input x is in a language.
  - That is answer, if the answer of a problem P for the instance x is "yes".
- Compute a function
  - Given an input x, what is f(x)?
- Generates the internal set of computations

How Machines are Used/usesof TM

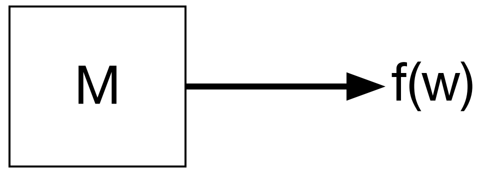
- To specify *functions or sets*
  - Transducer - maps string to string



  - Acceptor - ``recognizes" or ``accepts" a set of strings
    - M *accepts* strings which cause it to enter a final state



  - Generator - ``generates" a set of strings
    - M generates all values generated during computation (ex. neuro lang. generator)



How does a TM work?

- At the beginning,
  - A (TM) is in the *start state (initial state)*
  - its tape head points at the first cell
  - The tape contains **€**, following by input string, and the rest of the tape contains **€**.

How does a TM work?

- For each move, a TM
  - reads the symbol under its tape head
  - According to the *transition function* on the symbol read from the tape and its current state, the TM:
    - write a symbol on the tape
    - move its tape head to the left or right one cell or not
    - changes its state to the *next state*

When does a TM stop working?

- A (TM) stops working,
  - when it gets into the special state called halt state. (halts)
    - The output of the TM is on the tape.
  - when the tape head is on the leftmost cell and is moved to the left. (hangs)
  - when there is no *next state*. (hangs)

The differences between finite automata and Turing machines.

- A Turing machine 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.

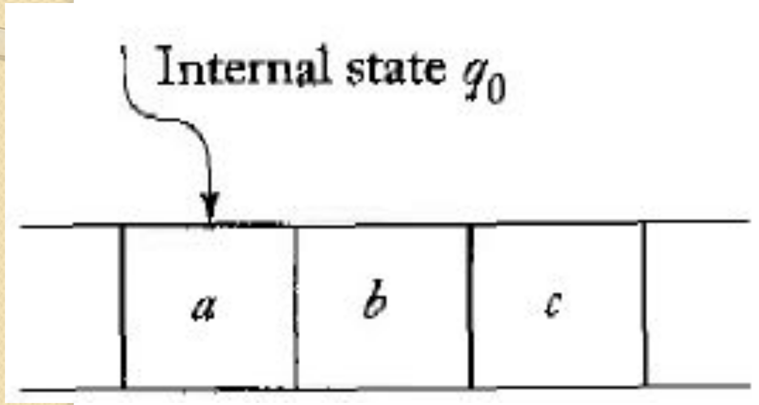
How to define deterministic TM (DTM)

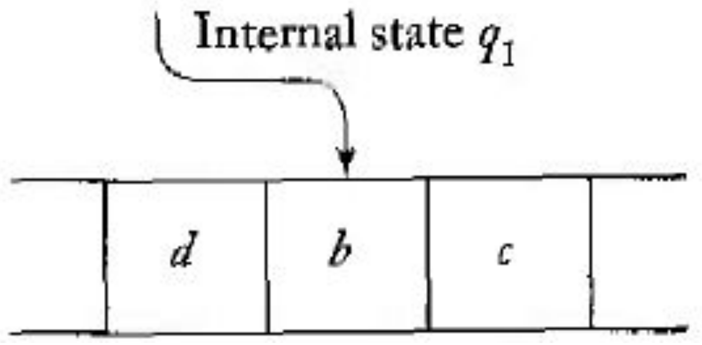
A standard Turing Machine M is defined by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, € , F),$$

- Q is a set of internal states.
- Σ is the input alphabet. We assume $\Sigma \subseteq \Gamma - \{€\}$
- Γ is a finite set of symbols called the tape alphabet.
- δ is the transition function $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- $€ \in \Gamma$ is a special symbol called the blank
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is a set of final states.

How does the TM work??





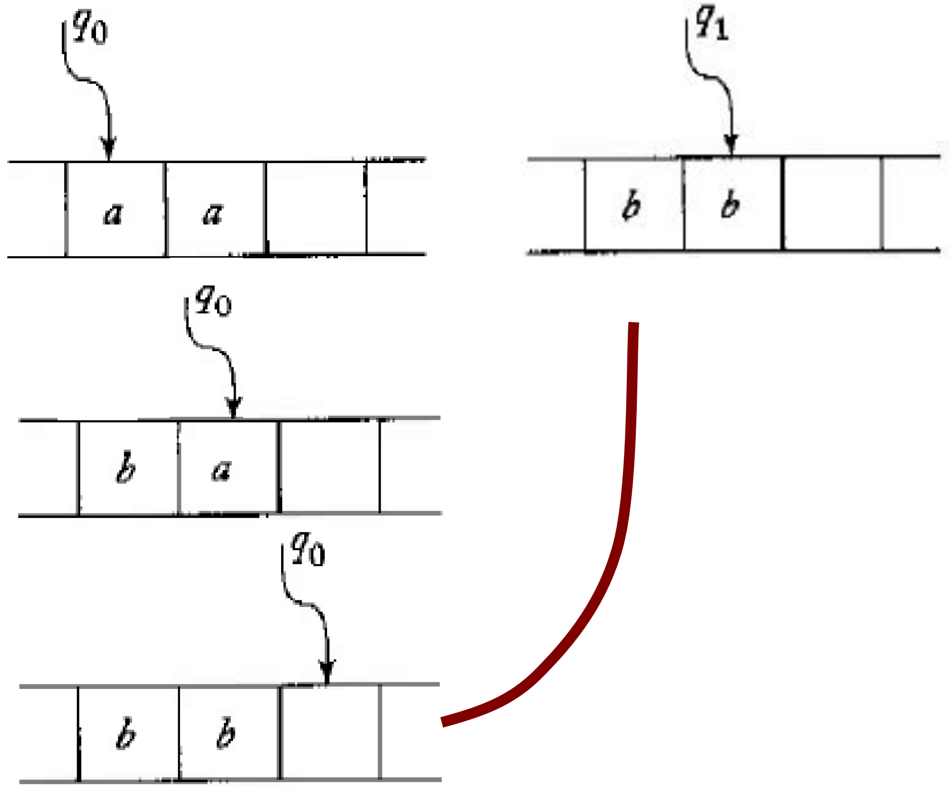
$\delta(q0, a) = (q1, d, R)$

How does the TM work???

- Example: consider the Turing machine defined by

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

$$\delta(q_0, a) = (q_0, b, R),$$
$$\delta(q_0, b) = (q_0, b, R),$$
$$\delta(q_0, \square) = (q_1, \square, L).$$

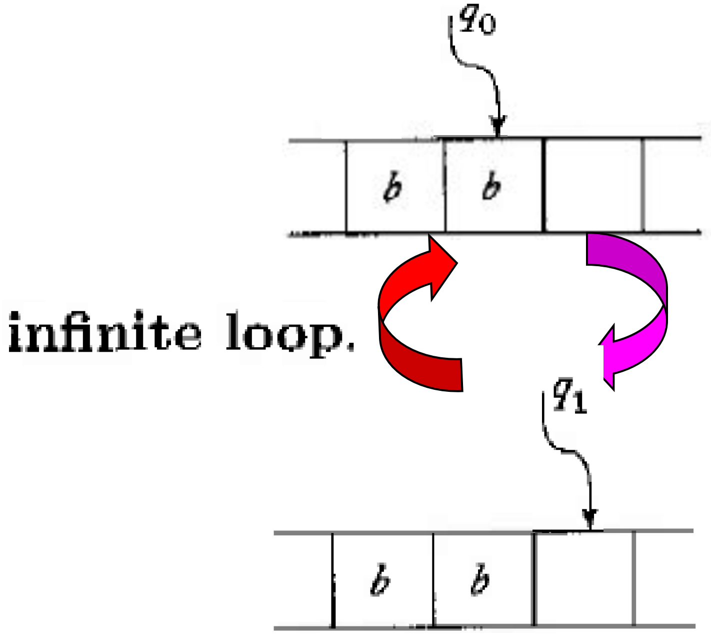


Turing machine that never halts

- Example:

Consider the following Turing machine:

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



Standard Turing machines

- The Turing machine has a tape that is unbounded in both directions, allowing any number of left and right moves.
- The Turing machine is deterministic in the sense that **δ** defines at most one move for each configuration.
- There is no special input file. Similarly, there is no special output device. Whenever the machine halts, some or all of the contents of the tape may be viewed as output.

Configuration

Definition

- Let $T = (Q, \Sigma, \Gamma, \delta, s)$ be a DTM.

  A configuration of T is an element of

$$(\Gamma)^* \times Q \times (\Gamma) \times (\Gamma)^*$$

- Can be written as
  - (q,l,a,r) or
  - $x_1 q a x_2$

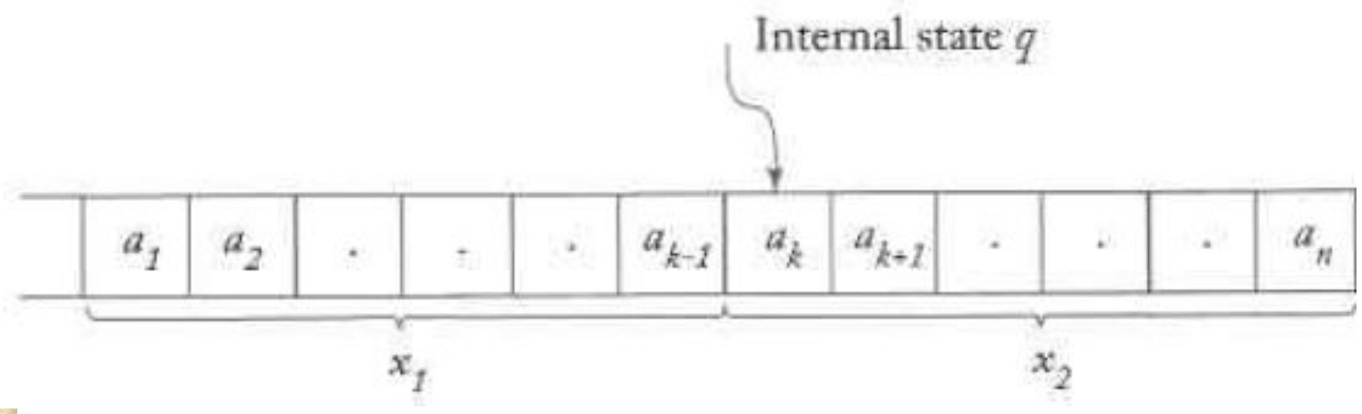
Current state

string to the right of tape head

symbol under tape head

string to the left of tape head

Turing machine



$$x_1 q x_2$$

or

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

Turing machine

A move from one configuration to another will be denoted by $\vdash$. Thus,

if $$\delta(q_1, c) = (q_2, e, R),$$

then the move

$$abq_1cd \vdash abeq_2d$$

Turing machine

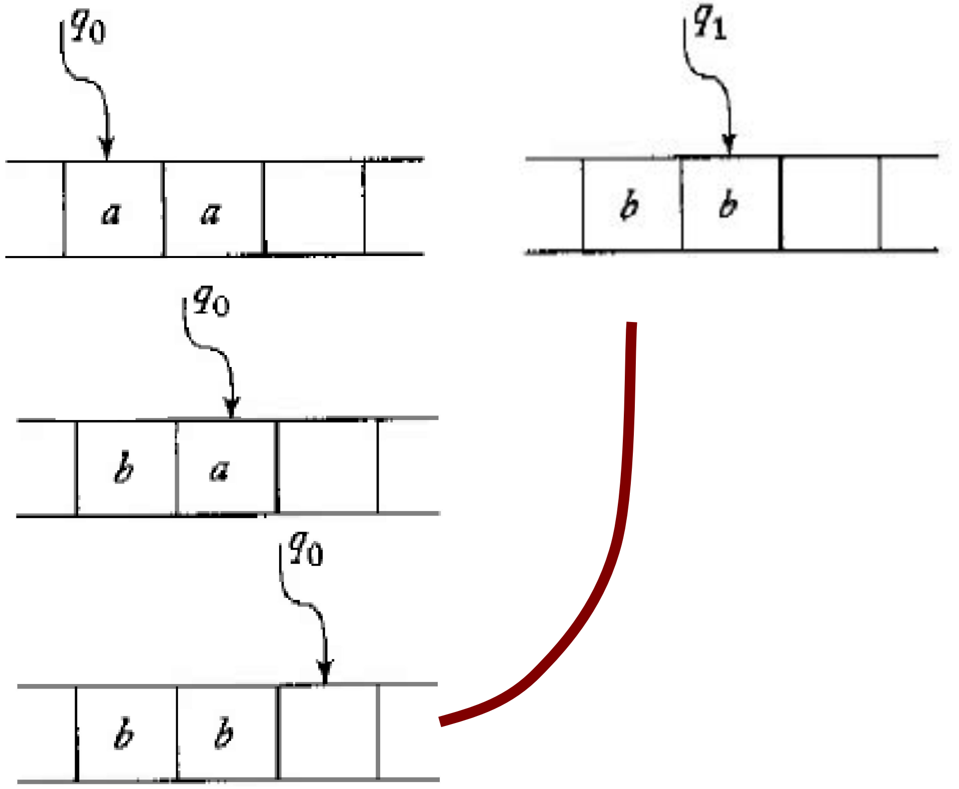
Example:

$$\delta(q_0, a) = (q_0, b, R),$$
$$\delta(q_0, b) = (q_0, b, R),$$
$$\delta(q_0, \square) = (q_1, \square, L).$$

$$q_0aa \vdash bq_0a \vdash bbq_0\square \vdash bq_1b$$

or

$$q_0aa \overset{*}{\vdash} bq_1b.$$



Turing machines

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

then a move

$$a_1 \cdots a_{k-1} q_1 a_k a_{k+1} \cdots a_n \vdash a_1 \cdots q_2 a_{k-1} b a_{k+1} \cdots a_n$$

Is possible if and only if the following transition is exist

$$\delta(q_1, a_k) = (q_2, b, L)$$

Turing machines

- Let M = $(Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ be a Turing machine then the language accepted by M is defined as

$$L(M) = \left\{ w \in \Sigma^+ : q_0 w \overset{*}{\vdash} x_1 q_f x_2 \text{ for some } q_f \in F, x_1, x_2 \in \Gamma^* \right\}$$

- When w is not in $L(M)$ one of two things may happen:

  - the machine may halts in non-final state.

  - the machine enters into an infinite loop and never halts

Turing machines

- M is said to halt starting from some initial configuration $x_1 q_i x_2$ $x_1 q_i x_2 \overset{*}{\vdash} y_1 q_j a y_2$ wher $\delta(q_j, a)$ is defined.
- We said that $x_1 q_i x_2$ yields $y_1 q_j a y_2$ in one or more steps.
- The configuration of the machine M that never halts will be presented by

$$x_1 q x_2 \overset{*}{\vdash} \infty,$$

- The sequence of configuration leading to a halt state called a ***computation***

Thanks for listning