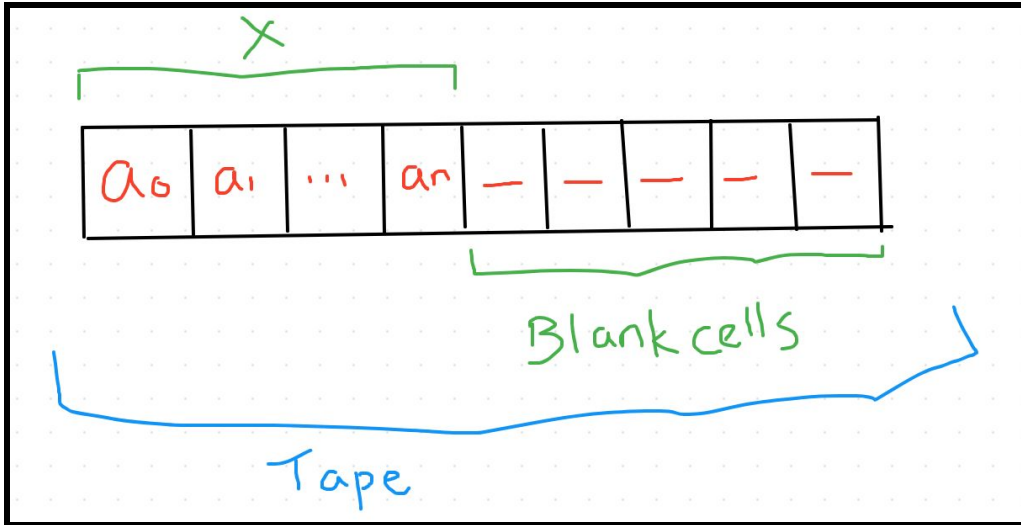


An Informal Description of Turing Machines:

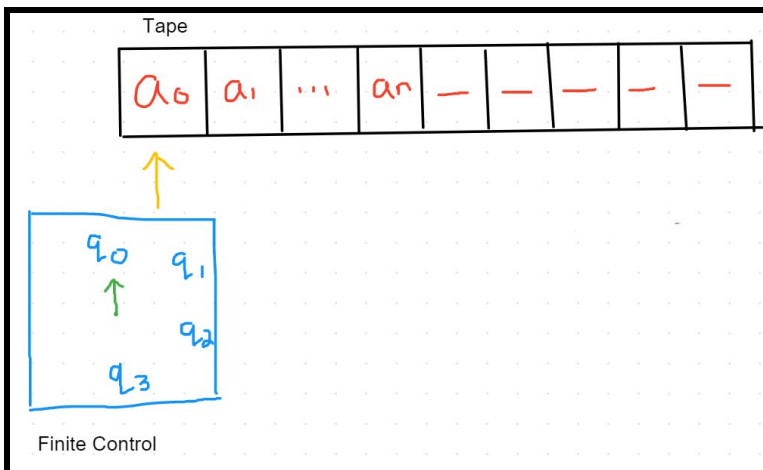
- A **turing machine (TM)** is a mathematical model for what it means to perform a **mechanical computation**. **Mechanical** means that given the representation of an input, we want to produce a representation of the appropriate output by following a finite sequence of steps, each of which can be obviously carried out. **Computation** means that given an input of some sort, we want to produce an appropriate output for that input.
Note: There could be multiple appropriate outputs for the given input.
- Examples of computations:
 - a. Input: $n \in \mathbb{N}$
Output: n^2
In this example, each output is unique.
 - b. Input: An undirected, connected, unweighted Graph, G
Output: A minimal spanning tree (MST) of G
In this example, there could be multiple appropriate outputs.
 - c. Input: $n \in \mathbb{N}$
Output: "Yes" if n is a prime number, and "No" otherwise
- When we're performing a computation, we're trying to solve a **problem**, which is an input-output relation.
- An **instance of a problem** is simply a particular input of that problem.
- A **decision problem** is a problem where the output is True/Yes or False/No.
- We can call an **instance of a decision problem** as either a **Yes-instance** or a **No-instance** depending on output for that instance.
- A **formal language** is a set of strings over some alphabet Σ .
- The subset of strings for which the problem returns Yes is a formal language, and often decision problems are defined as formal languages.
I.e. A decision problem is a set of representations of their Yes-instances.
- E.g.
Suppose we represent natural numbers, n , in binary. The problem is "Is n prime?". The input is a natural number and the output is Yes/No. I can think of the Yes-instances to this problem as a language.
I.e. $L_p = \{10, 11, 101, 111, \dots\}$ These binary strings represent the prime numbers and I can think of the problem "Is n prime" as "Does the binary representation of the number belong to L_p ?"
- A Turing Machine has a 1-way infinite tape divided into cells and it consists of a finite control, which is a box that can be in a finite number of states. This finite control allows the Turing Machine to read what's on the tape and to modify what's on the tape. Each cell of the tape contains 1 symbol from a finite alphabet, including a special blank symbol. At any point in time, the finite control scans exactly one cell of the tape. This is what the Turing Machine does:
Given the present state and the symbol on the tape, it may:
 1. Change its state and then move one cell left or one cell right.
 2. Change the symbol on the present cell and then move one cell left or one cell right.
- Note:** For our purposes, the tape is infinitely long to the right. If we are at the leftmost cell, and we want to move left, we just stay in that cell.
- We start the TM with some input, x , shown below, which is a sequence of finite strings. Every other cell in the tape is a blank cell.



The input string, x , is not allowed to contain any blank cells. This is because for the TM to determine the end of the input, it starts from the leftmost cell and goes to the right until it hits a blank cell. If the input itself contains a blank cell, then the TM would never know when the input ends.

Furthermore, suppose the finite control is in state q_0 .

I.e. This is what the TM looks like at the start.



Given its current state and the input, the TM either changes state or changes the input and it moves to the left or the right. The TM keeps executing these steps until the state that it's in becomes one of two special states, **Qa** or **Qr**.

Qa is the **accept state**.

Qr is the **reject state**.

Then, the TM stops and either accepts the input string, if it's in Q_a , or rejects the input string, if it's in Q_r .

Note: There is nothing requiring the TM to reach either Q_a or Q_r . It's possible that the TM will keep on going and never entering either Q_a or Q_r . In this case, we say that the **TM loops on the input**.

Note: Not accepting the input is not the same as rejecting the input.

We can define the language accepted by the TM as the set of input strings that lead to Q_a .

A Formal Description of Turing Machines:

- A TM M is a 7-tuple, with $M = (Q, \Sigma, \Gamma, \delta, Q_0, Q_a, Q_r)$ where
 - Q is a finite set of states.
 - Σ is a finite input alphabet. It does not include the blank symbol.
 - Γ is another finite alphabet, which is the set of symbols that can be written on the tape.
 - Note:** Γ includes symbols from Σ , but may also include other symbols, such as the blank symbol. Γ is a superset of Σ . Γ is called the **tape alphabet**.
 - δ is the **transition function** that tells the automation how to move. It defines the moves of the TM.
 - $\delta : (Q - \{Q_a, Q_r\} \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R\})$
 - I.e. δ takes a state that is not Q_a or Q_r and a tape symbol as input and outputs a new state, which may be the same as the old state, the new symbol on the tape head, which may be the same as the old symbol, and the direction in which the tape head should move (left or right).
 - E.g.
 - $\delta(q, a) \rightarrow \delta(p, b, L/R)$
 - Given $\delta(q, a)$ where q is a state that is not Q_a nor Q_r , and a tape symbol “a”, it returns a new state, which may be the same as the old state, a new tape symbol for the tape head, which may be the same as the old tape symbol, and a direction to move, left or right.
 - If M is in state “q” and its state head is scanning the tape cell with symbol “a”, then, M enters state “p” and writes “b” on the tape cell that is being scanned, and moves the tape head one cell to the left or right.
 - Q_0 is the initial state. $Q_0 \in Q$.
 - Q_a is the accept state. $Q_a \in Q$.
 - Q_r is the reject state. $Q_r \in Q - \{Q_a\}$.
- **Configuration/Instantaneous Description (ID):** The configuration or ID of a TM is a snapshot of the TM to describe the current situation of the TM. It provides the following information:
 - Current State
 - Contents of tape, except the trailing blanks
 - Position of the tape head

Given the above 3 pieces of information, we know exactly what will happen next.
 I.e. We will know what will be the next state, the next contents of the tape, and next position of the tape head.

Formally, a configuration is a string of the form “xqy” where $x, y \in \Gamma^*$, s.t. y does not end with a blank, and $q \in Q$.
 x is a string of tape symbols and y is a string of tape symbols not ending in blank symbols.

q represents the current state.

xy are the contents of the tape.

The tape head is positioned over the first cell of string y , or if y is the empty string, then the tape head is positioned over the first of the infinitely many blank strings.

- Consider the relation, \vdash_M , between configurations of the TM M. Let C and C' be configurations.

Informally, think of $C \vdash_M C'$ as "From configuration C, TM M moves to configuration C' in 1 step."

$C \vdash_M C'$ is read as "C yields C'."

Formally, suppose that $C = xqy$.

If $y = ay'$ (i.e. y is not empty and starts with symbol a), and $\delta(q,a) = (p, b, R)$, then

$C \vdash_M C'$ iff $C' = xbp'y'$.

If $y = ay$ and $x = x'c$ (i.e. x is not empty and ends with symbol c), and $\delta(q,a) = (p, b, L)$, then

$C \vdash_M C'$ iff $C' = x'pcby$.

If $y = ay'$ and $x = \epsilon$ (i.e. x is empty), and $\delta(q,a) = (p, b, L)$, then

$C \vdash_M C'$ iff $C' = pby'$.

- We define \vdash_M^* to be the transitive closure of \vdash_M . This means that $C \vdash_M^* C'$ iff
 - $C' = C$
 - There is a finite sequence of configurations C_1, C_2, \dots, C_k , such that $C_1 = C$,

$C_k = C'$ and $C_i \vdash_M C_{i+1}$ for all $i \in 1 \leq i \leq k-1$.

- A TM M accepts string x, $x \in \Sigma^*$, iff $Q_0x \vdash_M^* yQ_{\text{az}}$, where Q_0x is the initial configuration and yQ_{az} is the accepting configuration.

A TM M rejects string x, $x \in \Sigma^*$, iff $Q_0x \vdash_M^* yQ_{\text{rz}}$, where Q_0x is the initial configuration and yQ_{rz} is the rejecting configuration.

If either case happens, then we say that "M halts on x".

If M doesn't halt on x, then we say that "M loops on x".

M loops on x if there is an infinite number of configurations C_1, C_2, \dots , such that

$Q_0x \vdash_M C_1 \vdash_M C_2 \dots$. This is like an infinite loop.

- The language $L(M)$ **accepted/recognized** by TM M is defined as:
 $L(M) = \{x \in \Sigma^* \mid \text{M accepts } x\}$.

- **Note:** If M does not accept x , $x \in \Sigma^*$, this does not mean that M rejects x .
If M does not accept x , there are 2 possibilities:
 1. M rejects x . OR
 2. M loops on x .
- Given a TM M and a string x , $x \in \Sigma^*$, there are 3 possibilities:
 1. M accepts x .
 2. M rejects x .
 3. M loops on x .
- A language L is **recognizable/semi-decidable/recursively enumerable** iff there exists a TM M such that $L(M) = L$. Here, M is called a **recognizer**.
- A language L is **decidable/recursive** iff there exists a TM M such that $L(M) = L$ and M halts on every input. Here, M is called a **decider**.
- If L is a decider, then by definition, it is also a recognizer.
- Since a language is simply a set of strings, we will also say that a set is decidable/recognizable.
- Since a language is simply another way of thinking of a decision problem, we will also say that a decision problem is decidable/recognizable.
- We will create a TM that decides the language L , where
 $L = \{x \in \Sigma^* \mid x \text{ is an even length palindrome}\}$
 $\Sigma = \{0, 1, 2\}$

E.g. $\epsilon, 00, 0220 \in L$

This is a high level description of our TM:

1. If the symbol under the tape head is the blank symbol, then accept it.
Otherwise, "remember" that symbol, replace it with a blank and move 1 cell to the right.
2. While the symbol scanned is not a blank symbol, move right.
3. Move one cell to the left.
4. If the symbol under the tape head is not the same as the one "remembered", then reject it.
Otherwise, replace it with a blank symbol and move 1 cell left.
5. While the symbol scanned is not a blank symbol, move left.
6. Move one cell to the right and go to stage 1.

This is a formal description of our TM:

- States of M :
 - Q_0 : Initial State
 - Q_a/Q_r : Accept/Reject State
 - $[Q_1, a] \forall a \in \Sigma$: This means, keep moving right and the first symbol is a .
 - $[Q_2, a] \forall a \in \Sigma$: This means, I have reached the right end and the symbol remembered is a .
 - Q_3 : Keep moving left.
- Hence, $Q = \{Q_0, Q_a, Q_r, Q_3\} \cup \{[Q_1, a] \mid a \in \Sigma\} \cup \{[Q_2, a] \mid a \in \Sigma\}$
- $\Sigma = \{0, 1, 2\}$
 - $\Gamma = \{0, 1, 2, \sqcup\}$

$$\delta(q_0, a) = \begin{cases} (q_a, \sqcup, R) & \text{if } a = \sqcup \\ ([q_1, a], \sqcup, R), & \text{o.w.} \end{cases}$$

I.e. We start at the initial state, Q_0 , and our symbol is a . If a is the blank symbol, then we enter the accept state, Q_a . Otherwise, we enter state Q_1 , remember a , and replace it with a blank symbol and move 1 cell to the right.

$$\delta([q_1, b], a) = \begin{cases} ([q_1, b], a, R) & \text{if } a \neq \sqcup \\ ([q_2, b], \sqcup, L) & \text{if } a = \sqcup \end{cases}$$

I.e. If a is not the blank symbol, we just keep moving right. If a is the blank symbol, we enter state Q_2 and move one cell left.

$$\delta([q_2, b], a) = \begin{cases} (q_r, \sqcup, L) & \text{if } a \neq b \\ (q_3, \sqcup, L) & \text{if } a = b \end{cases}$$

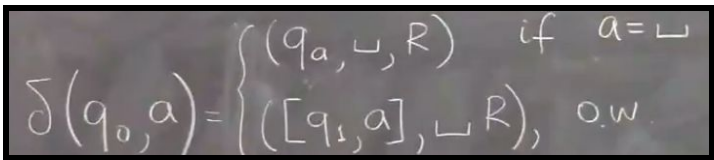
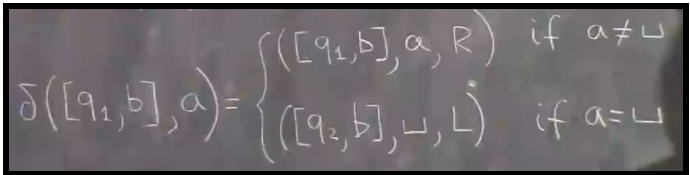
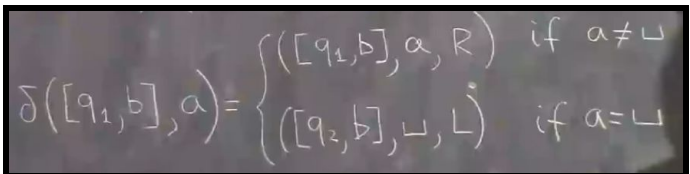
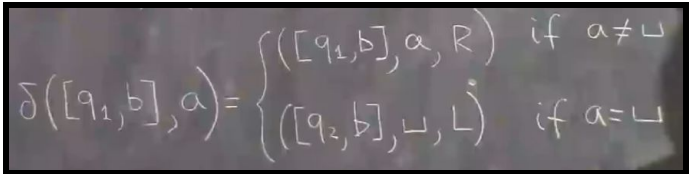
I.e. If a is not equal to b , then we enter the reject state, Q_r . Otherwise, we enter state Q_3 , replace a with the blank symbol, and move left.

$$\delta(q_3, a) = \begin{cases} (q_3, a, L) & \text{if } a \neq \sqcup \\ (q_0, \sqcup, R) & \text{if } a = \sqcup \end{cases}$$

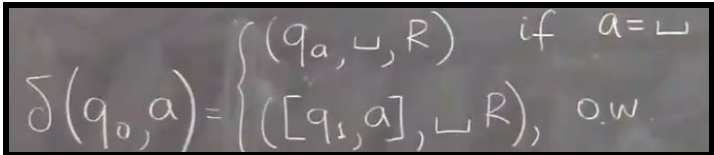
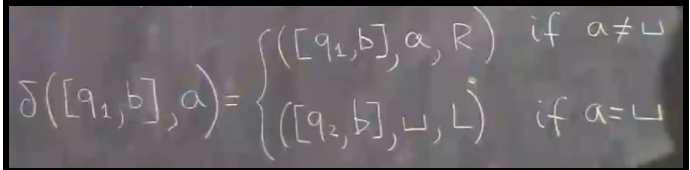
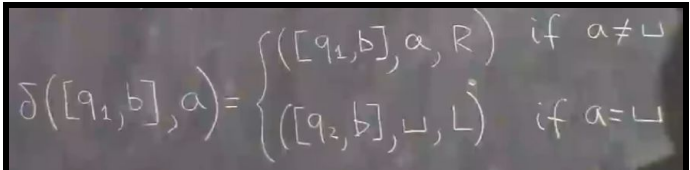
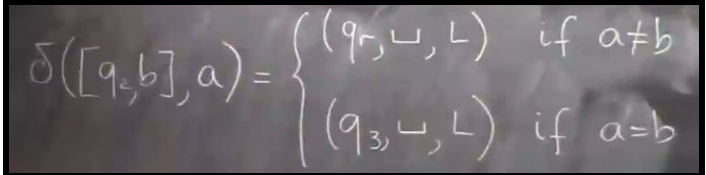
I.e. If a is not the blank symbol, then we keep moving left. If a is the blank symbol, we enter state Q_0 and move 1 cell to the right.

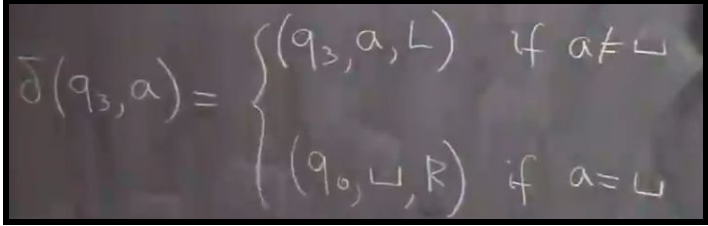
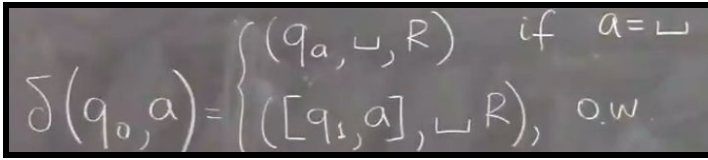
Example of our TM on the string 0110:

Note: To describe what the TM does on the given string, I will use a series of configurations shown below:

Configurations	Description
Q00110	<p>We're starting at the initial configuration, Q0. Based on the transition function shown below</p>  <p>since $a = 0$, we enter state $[Q1,0]$, replace a with \sqcup and move 1 cell to the right.</p>
$\sqcup[Q1,0]110$	<p>We're now in state $[Q1,0]$. Based on the transition function shown below</p>  <p>since $a = 1$, we just move 1 cell to the right.</p>
$\sqcup1[Q1,0]10$	<p>We're still in state $[Q1,0]$. Based on the transition function shown below</p>  <p>since $a = 1$, we just move 1 cell to the right.</p>
$\sqcup11[Q1,0]0$	<p>We're still in state $[Q1,0]$. Based on the transition function shown below</p>  <p>since $a = 0$, we just move 1 cell to the right.</p>
$\sqcup110[Q1,0]$	<p>We're still in state $[Q1,0]$. Based on the transition function shown below</p>

	$\delta([q_1, b], a) = \begin{cases} ([q_1, b], a, R) & \text{if } a \neq \sqcup \\ ([q_2, b], \sqcup, L) & \text{if } a = \sqcup \end{cases}$ <p>since $a = \sqcup$, we enter state $[Q2, 0]$ and move 1 cell left.</p>
$\sqcup 11[Q2, 0]0\sqcup$	<p>We're now in state $[Q2, 0]$. Based on the transition function shown below</p> $\delta([q_2, b], a) = \begin{cases} (q_2, \sqcup, L) & \text{if } a \neq b \\ (q_3, \sqcup, L) & \text{if } a = b \end{cases}$ <p>since $a = b$ ($0 = 0$), we enter state $Q3$, replace a with \sqcup and move 1 cell left.</p>
$\sqcup 1Q31\sqcup$	<p>We're now in state $Q3$. Based on the transition function shown below</p> $\delta(q_3, a) = \begin{cases} (q_3, a, L) & \text{if } a \neq \sqcup \\ (q_0, \sqcup, R) & \text{if } a = \sqcup \end{cases}$ <p>since $a = 1$, we just move 1 cell left.</p>
$\sqcup Q311\sqcup$	<p>We're still in state $Q3$. Based on the transition function shown below</p> $\delta(q_3, a) = \begin{cases} (q_3, a, L) & \text{if } a \neq \sqcup \\ (q_0, \sqcup, R) & \text{if } a = \sqcup \end{cases}$ <p>since $a = 1$, we just move 1 cell left.</p>
$Q3\sqcup 11\sqcup$	<p>We're still in state $Q3$. Based on the transition function shown below</p> $\delta(q_3, a) = \begin{cases} (q_3, a, L) & \text{if } a \neq \sqcup \\ (q_0, \sqcup, R) & \text{if } a = \sqcup \end{cases}$

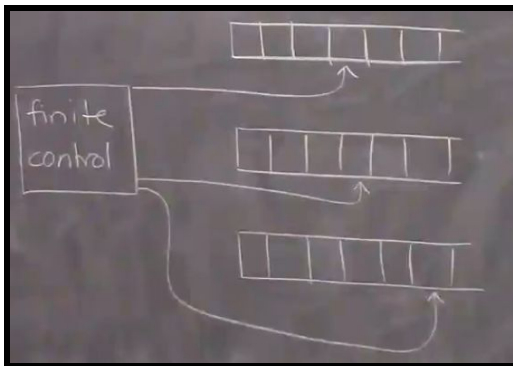
	since $a = \sqcup$, we enter state Q_0 and move 1 cell right.
$\sqcup Q_0 11 \sqcup$	<p>We're now in state Q_0. Based on the transition function shown below</p>  $\delta(q_0, a) = \begin{cases} (q_a, \sqcup, R) & \text{if } a = \sqcup \\ ([q_1, a], \sqcup, R), & \text{o.w.} \end{cases}$ <p>since $a = 1$, we enter state $[Q_1, 1]$, replace a with \sqcup and move 1 cell to the right.</p>
$\sqcup \sqcup [Q_1, 1] 1 \sqcup$	<p>We're now in state $[Q_1, 1]$. Based on the transition function shown below</p>  $\delta([q_1, b], a) = \begin{cases} ([q_1, b], a, R) & \text{if } a \neq \sqcup \\ ([q_2, b], \sqcup, L) & \text{if } a = \sqcup \end{cases}$ <p>since $a = 1$, we just move 1 cell to the right.</p>
$\sqcup \sqcup 1 [Q_1, 1] \sqcup$	<p>We're still in state $[Q_1, 1]$. Based on the transition function shown below</p>  $\delta([q_1, b], a) = \begin{cases} ([q_1, b], a, R) & \text{if } a \neq \sqcup \\ ([q_2, b], \sqcup, L) & \text{if } a = \sqcup \end{cases}$ <p>since $a = \sqcup$, we enter state $[Q_2, 1]$ and move 1 cell left</p>
$\sqcup \sqcup [Q_2, 1] 1 \sqcup$	<p>We're now in state $[Q_2, 1]$. Based on the transition function shown below</p>  $\delta([q_2, b], a) = \begin{cases} (q_2, \sqcup, L) & \text{if } a \neq b \\ (q_3, \sqcup, L) & \text{if } a = b \end{cases}$ <p>since $a = b$ ($1 = 1$), we enter state Q_3, replace a with \sqcup and move 1 cell left.</p>
$\sqcup Q_3 \sqcup$	<p>We're now in state Q_3. Based on the transition function shown below</p>

	 <p>since $a = \sqcup$, we enter state $Q0$ and move 1 cell right.</p>
$\sqcup \sqcup Q0 \sqcup$	<p>We're now in state $Q0$. Based on the transition function shown below</p>  <p>since $a = \sqcup$, we enter the state Q_{accept}.</p>
$\sqcup \sqcup \sqcup Q_{accept}$	<p>We're now in state Q_{accept}.</p>

Note: The bolded parts are the states.

Variants of TMs:

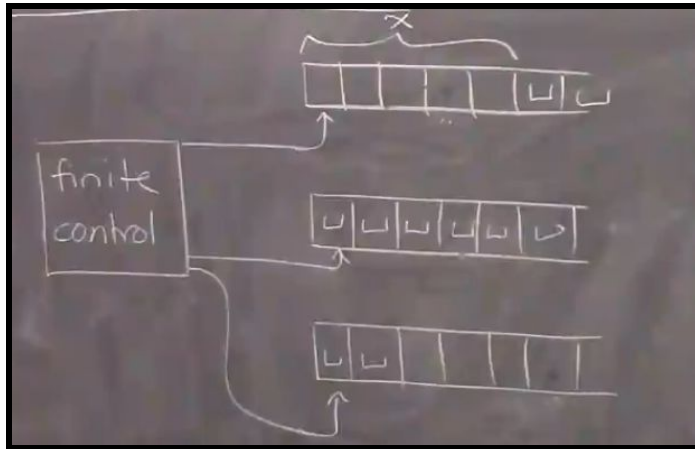
- **Multi-tape TM:**
- Looks like this:



There is one control machine and there are k tapes. Each of these k tapes has its own tape head.

- This is how multi-tape TMs work:
 - Each tapehead starts at the leftmost cell of their respective tapes.
 - The input, X , is at the leftmost cell(s) of the first tape. Every other cells, including cells in the other tapes, are blanks.

i.e.



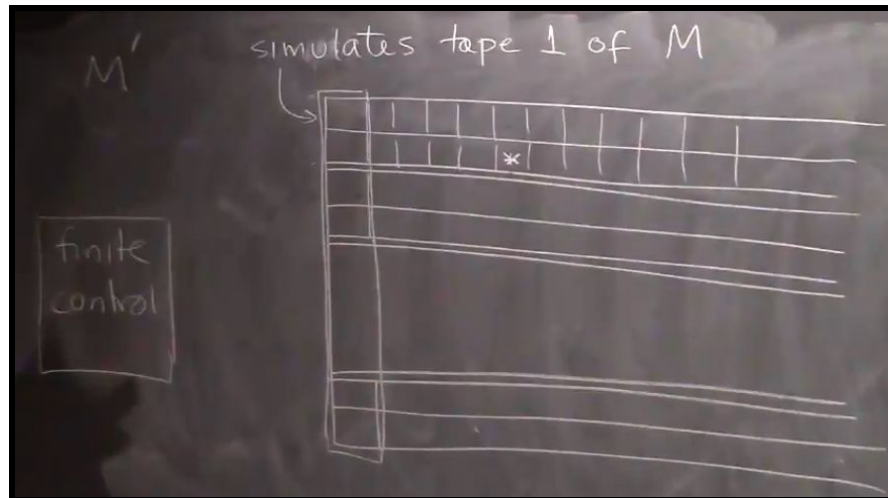
- The control machine is at some state and it scans each of the k tapes.
- Then, based on the machine's state and what it is currently scanning, it performs an action on that tape.
- Hence, each action is independent of other tapes.
- **Theorem 2.1:** If M is a multi-tape (k -tape) TM, there exists a basic TM M' such that $L(M') = L(M)$.

i.e. I can simulate a multi-tape TM with a basic TM.

Furthermore, if M accepts/rejects input x in m moves, then M' accepts/rejects x in $O(m^2)$ moves.

Proof that $L(M') = L(M)$:

- This is what M' will look like:



It will have one tape, but divided up into $2k$ tracks. It will have 2 tracks for each tape in M . In other words, every 2 tracks in M' corresponds to 1 tape in M .

The first track in every pair of tracks will simulate the corresponding tape of M .

The second track in every pair of tracks is blank everywhere, except for in 1 square, where it has a star. That star represents the location of the tape head of the corresponding tape in M .

- Remember M 's state.
- Head starts from the leftmost cell of M' .

- Scan the multitape of M' to the right, remembering in the state of M' all the symbols corresponding to the stars until you have seen all k stars.
- Using the state transition function of M , we determine the symbols to be written on the cell of each tape under its head. Remember them in our state.
- M' scans its multitape (single) tape to the left performing the appropriate action at each star. M' accepts/rejects if M does.

Proof that M' takes $O(m^2)$ moves:

If M takes m moves, its heads are at most m cells apart.

The right moves of M' to simulate one move of M takes at most m steps.

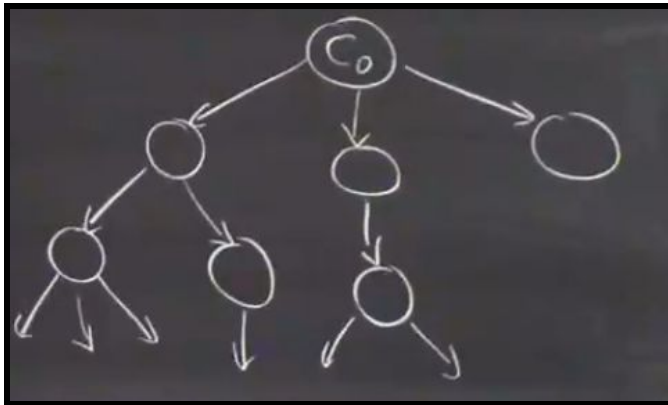
The left moves of M' to simulate one move of M takes at most $(m + 2k)$ steps. It takes $m+2k$ steps because we might need to move 1 cell right before moving $m+1$ cells to the left. That's 2 extra steps. Since there are k tapes in M , at most, there would be $2k$ extra steps. Hence, we get $m+2k$ steps.

Hence, one move of M takes $(m) + (m+2k)$ or $2(m+k)$ moves of M' . Therefore, m moves of M takes $O(m^2)$ moves of M' .

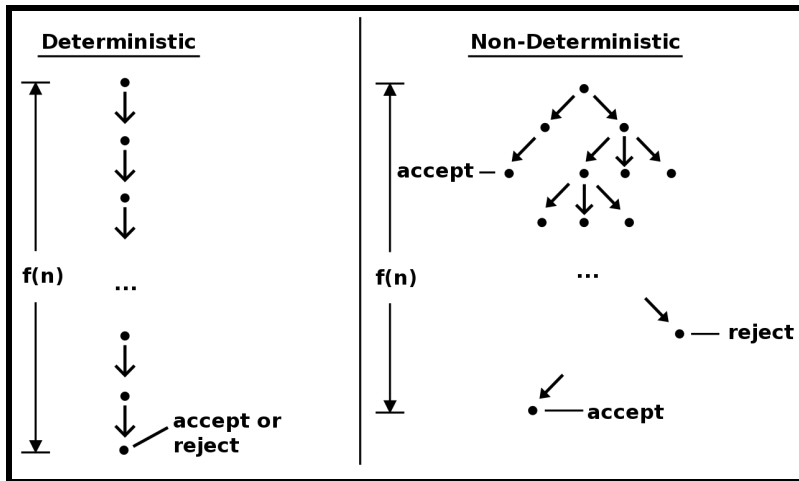
Non-deterministic TM (NTM):

- In a regular TM, the transition function is defined as $\delta(q, a) \rightarrow \delta(p, b, L/R)$.
- In a NTM, the transition function is defined as $\delta(q, a) \rightarrow \delta\{(p, b, L/R)\}$.
- In a regular TM, the configurations were in a linear structure.
I.e. $C_0 \vdash C_1 \vdash \dots$

With NTMs, the configurations are in a tree-like structure.



Here's a side by side comparison of configurations for TM and NTM.



- A NTM M accepts input x iff there exists a computation path in the computation tree of M on x ending in a configuration in the accept state.
I.e. As long as there exists one computation path that leads to the accepting state, we say that the NTM accepts the input.
- **Theorem 3.1:** A language L is recognized by a TM iff L is recognized by a NTM.

Proof:

(\Rightarrow)

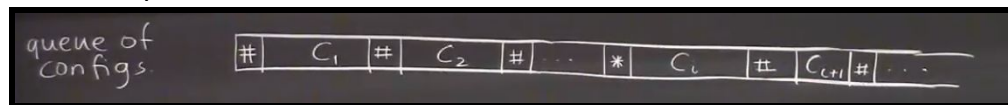
A TM is a special case of a NTM. Hence, if L is recognized by a TM, then it is also recognized by a NTM.

(\Leftarrow)

(High level idea)

- Do a BFS of the computation tree, until you discover a configuration in the accept state, in which case we accept.
- We will use a multi-tape (2 tape) TM to simulate the NTM.
- In the multi-tape, the first tape will be a queue of configurations of the NTM. We will use a special symbol, $\#$, to separate the configurations. However, one of the configurations will have a star, $*$, in front of it instead of a $\#$. The star represents the configuration currently being looked at. Furthermore, configurations to the left of the star have already been looked at.

The first tape is shown below.



The second tape will be a work tape.

- Recall that configurations are of the form xqy where q is a state and xy is the contents of the NTM tape that we are simulating. Suppose that $y = ay'$.
Hence, $xqy = xqay'$.
- The TM that's simulating the NTM scans the configuration C_i until it finds the state. Then, it consults the transition function of the NTM.

Suppose this is the transition function of the NTM.

$$\delta(q, a) \rightarrow p_1, a_1, D_1$$

$$\delta(q, a) \rightarrow p_2, a_2, D_2$$

- Then, the TM will create 2 copies of configuration C_i into the work tape. It makes 2 copies because there are 2 possibilities in the transition function. Then, it will go into each copy of the configuration in the work tape and change it based on its corresponding possibility of the transition function.
I.e. Copy i will be changed based on the i^{th} possibility of the transition function.
- The TM will continue scanning to the right until it hits a blank symbol. Once it sees the blank symbol, it copies the contents of the work tape to the end of the queue. The work tape is now empty.
- Then, the TM goes back, looking for the star. It replaces the star with $\#$ and looks for the next $\#$, at which it will change the $\#$ to a star and repeats for the next configuration.
- If the TM ever creates a configuration in the work tape that contains the accept state, it stops and accepts.
- **Note:** The reason why we are doing a BFS instead of a DFS is because the DFS strategy goes all the way down one branch before backing up to explore other branches. If the TM were to explore the tree in this manner, it could go forever down one infinite branch and miss an accepting configuration on some other branch. Hence we design the TM to explore the tree by using BFS instead. This strategy explores all branches to the same depth before going on to explore any branch to the next depth. This method guarantees that the TM will visit every node in the tree until it encounters an accepting configuration.
- **Note:** Non-determinism does not add any powers to regular TMs.
- Suppose that a NTM M has an accepting computation path in m moves. Furthermore suppose that the maximum number of choices of M is k . How many moves does it take a simulating TM M' to accept?

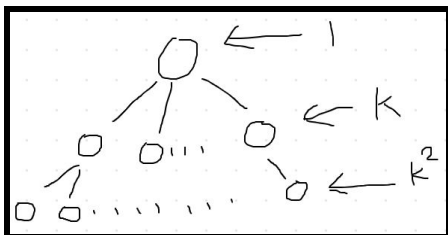
Solution:

For the root of the tree, there is 1 choice.

For the "row" after the root, there are k choices.

For the "row" after that, there are k^2 choices.

And so on.



Hence, the number of configurations explored by M' before finding the accepting configuration of M is $1 + k^2 + k^3 + \dots + k^m$. This is $O(k^m)$.

Furthermore, if the number of moves of M is m , then the length of the configuration is $m+1$. This is because m symbols may have been changed, plus 1 for the state. This is $O(m)$.

Lastly, since we copy each configuration to the work tape and back, the amount of work done for each configuration is proportional to its length.

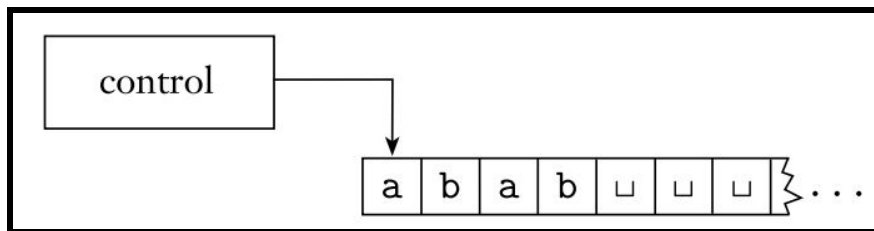
Putting everything together, the total number of moves by M' is $O(m \cdot k^m)$ which is $2^{O(m)}$.

Textbook Notes:

- **Introduction to Turing Machines:**

- The Turing Machine (TM) was invented by Alan Turing in 1936.
- The Turing Machine model uses an infinite tape as its unlimited memory. It has a tape 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 the machine needs to store information, it may write this information on the tape. To read the information that it has written, the machine can move its head back over it. The machine continues computing until it decides to produce an output. The outputs accept and reject are obtained by entering designated accepting and rejecting states. If it doesn't enter an accepting or a rejecting state, it will go on forever, never halting.

I.e. A schematic of a TM



- Example of a TM:

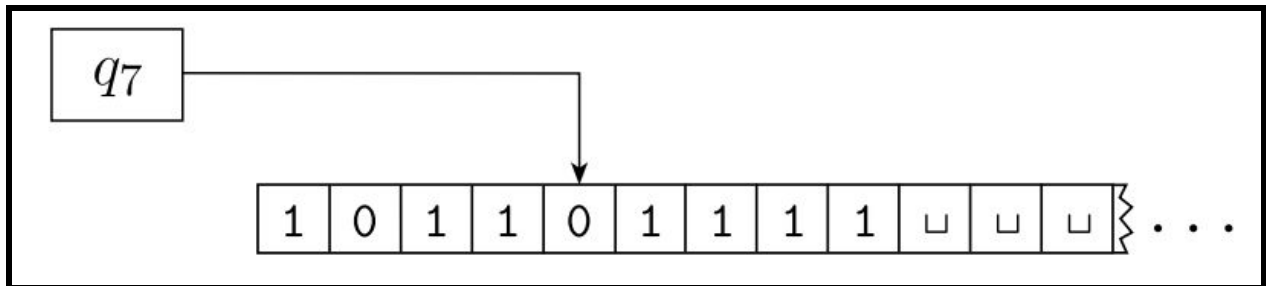
Let's introduce a Turing machine M_1 for testing membership in the language $B = \{w\#w \mid w \in \{0,1\}^*\}$. We want M_1 to accept if its input is a member of B and to reject otherwise. Here's a high level summary of M_1 's algorithm on input string x :

- 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, or if no $\#$ is found, 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."
- **FORMAL DEFINITION OF A TM:**
- The heart of the definition of a Turing machine is the transition function δ because it tells us how the machine gets from one step to the next. For a Turing machine, δ takes the form: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$. That is, when the machine is in a certain state q and the head is over a tape square containing a symbol a , and if $\delta(q, a) = (r, b, L)$, 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. In this case, the L indicates a move to the left.
- A **TM** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, Q_0, Q_{\text{accept}}, Q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states.
 2. Σ is the input alphabet not containing the blank symbol \sqcup .
 3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$.
 4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function.
 5. $Q_0 \in Q$ is the start state.
 6. $Q_{\text{accept}} \in Q$ is the accept state.
 7. $Q_{\text{reject}} \in Q$ is the reject state, where $Q_{\text{reject}} \neq Q_{\text{accept}}$.
- A Turing machine $M = (Q, \Sigma, \Gamma, \delta, Q_0, Q_{\text{accept}}, Q_{\text{reject}})$ computes as follows. Initially, M receives its input $w = w_1w_2 \dots w_n \in \Sigma^*$ 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 Σ does not contain the blank symbol, so the first blank appearing on the tape marks the end of the input. Once M has started, the computation proceeds according to the rules described by the transition function. If M 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 the accept or reject states, at which point it halts. If neither occurs, M goes on forever.
 - As a Turing machine 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 Turing machine. Configurations often are represented in a special way. For a state q and two strings u and v over the tape alphabet Γ , we write uqv 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 . The tape contains only blanks following the last symbol of v .

E.g.

1011Q70111 represents the configuration when the tape is 101101111, the current state is Q_7 , and the head is currently on the second 0, as shown below:



- We say that configuration C_1 **yields** configuration C_2 if the Turing machine can legally go from C_1 to C_2 in a single step. We define this notion formally as follows:
Suppose that we have a, b , and c in Γ , as well as u and v in Γ^* and states Q_i and Q_j . In that case, $uaQibv$ and $uQjacv$ are two configurations. We say that $uaQibv$ yields $uQjacv$ if in the transition function $\delta(Q_i, b) = (Q_j, c, L)$. That handles the case where the Turing machine moves leftward. For a rightward move, we say that $uaQibv$ yields $uacQjv$ if $\delta(Q_i, b) = (Q_j, c, R)$.
- Special cases occur when the head is at one of the ends of the configuration.
For the left-hand end, the configuration $Qibv$ yields $Qjcv$ if the transition is left-moving because we prevent the machine from going off the left-hand end of the tape, and it yields $cQjv$ for the right-moving transition.
For the right-hand end, the configuration uaQ_i is equivalent to $uaQ_i\sqcup$ because we

assume that blanks follow the part of the tape represented in the configuration. Thus we can handle this case as before, with the head no longer at the right-hand end.

- The **start configuration** of M on input w is the configuration Q0w, which indicates that the machine is in the start state Q0 with its head at the leftmost position on the tape.
- In an **accepting configuration**, the state of the configuration is Qaccept.
- In a **rejecting configuration**, the state of the configuration is Qreject.
- Accepting and rejecting configurations are **halting configurations** and do not yield further configurations.
- A Turing machine M accepts input w if a sequence of configurations C1, C2, ... , Ck exists, where:
 1. C1 is the start configuration of M on input w,
 2. each Ci yields Ci+1, and
 3. Ck is an accepting configuration
- The collection of strings that M accepts is called **the language of M** or **the language recognized by M** and is denoted L(M).
- We call a language **Turing-recognizable** if some Turing machine recognizes it.
- When we start a Turing machine on an input, three outcomes are possible: accept, reject or **loop**. By **loop** we mean that the machine simply does not halt. Looping may entail any simple or complex behavior that never leads to a halting state.
- A Turing machine M can fail to accept an input by entering the Qreject state and rejecting or by looping. Sometimes distinguishing a machine that is looping from one that is merely taking a long time is difficult. For this reason, we prefer Turing machines that halt on all inputs; such machines never loop. These machines are called **deciders** because they always make a decision to accept or reject. A **decider** that recognizes some language also is said to **decide** that language. We call a language **Turing-decidable** or simply decidable if some Turing machine decides it.
- **VARIANTS OF TURING MACHINES:**
 1. **MULTI-TAPE TURING MACHINES:**
 - A **multi-tape 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 some or all of the tapes simultaneously. Formally, the transition function is defined as:

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

where k is the number of tapes.

- The expression $\delta(Q_i, a_1, \dots, a_k) = (Q_j, b_1, \dots, b_k, L, R, \dots, L)$ means that if the machine is in state Q_i and heads 1 through k are reading symbols a_1 through a_k , the machine goes to state Q_j , writes symbols b_1 through b_k , and directs each head to move left or right, or to stay put, as specified.
- **Theorem:** Every multi-tape Turing machine has an equivalent single-tape Turing machine.
- **Corollary:** A language is Turing-recognizable iff some multi-tape Turing machine recognizes it.

Proof:

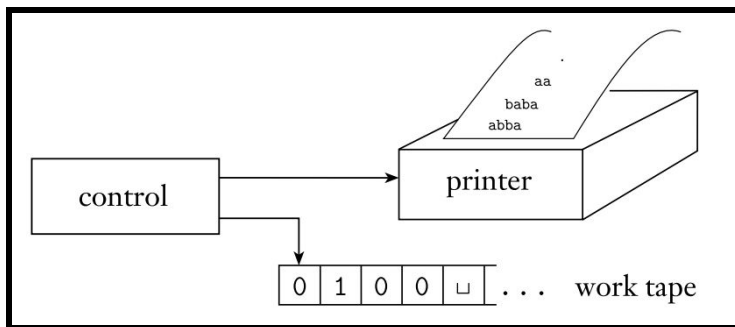
A Turing-recognizable language is recognized by an ordinary (single tape) Turing machine, which is a special case of a multi-tape Turing machine. That proves one direction of this corollary. The other direction follows from the above theorem that every multi-tape Turing machine has an equivalent single-tape Turing machine.

2. NONDETERMINISTIC TURING MACHINES:

- **Theorem:** Every nondeterministic Turing machine has an equivalent deterministic Turing machine.
- **Corollary:** A language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it.
- **Corollary:** A language is decidable if and only if some nondeterministic Turing machine decides it.

- ENUMERATORS:

- Loosely defined, an **enumerator** is a Turing machine with an attached printer. The Turing machine can use that printer as an output device to print strings. Every time the Turing machine wants to add a string to the list, it sends the string to the printer. I.e.



- An enumerator E starts with a blank input on its work tape. If the enumerator doesn't halt, it may print an infinite list of strings. The language enumerated by E is the collection of all the strings that it eventually prints out. Moreover, E may generate the strings of the language in any order, possibly with repetitions.
- More specifically, an **enumerator** for a language $L \subseteq \Sigma^*$ is a multitape TM E_L that, if started with all tapes being blank, lists on Tape 1 every element of L preceded and followed by a special symbol $\#$, where $\# \notin \Sigma$. The computation of E_L started in its initial state with all tapes blank is a sequence of configurations $C_1 \vdash C_2 \vdash \dots \vdash C_i \vdash \dots$, such that:
 1. If $i < j$ then the string contained in Tape 1 in C_i is a prefix of that in C_j ; and
 2. $\#x\#$ is a substring of the string contained in Tape 1 in some C_i if and only if $x \in L$.
- **Theorem:** A language is Turing-recognizable if and only if some enumerator enumerates it.

Proof:

\Rightarrow

First we show that if we have an enumerator E that enumerates a language A , a TM M recognizes A .

The TM M works in the following way.

$M =$ "On input w :

1. Run E . Every time that E outputs a string, compare it with w .
2. If w ever appear in the output of E , accept."

Clearly, M accepts those strings that appear on E 's list.

\leq

If TM M recognizes a language A , we can construct the following enumerator E for A .

Say that s_1, s_2, s_3, \dots is a list of all possible strings in Σ^* .

$E =$ "Ignore the input.

Repeat the following for $i = 1, 2, 3, \dots$.

- a. Run M for i steps on each input, s_1, s_2, \dots, s_i .
- b. If any computations accept, print out the corresponding s_j ."

If M accepts a particular string s , eventually it will appear on the list generated by E . In fact, it will appear on the list infinitely many times because M runs from the beginning on each string for each repetition of step 1.

Here, we're defining A as the set of all strings that are accepted by M . We may not know what these strings are given by an arbitrary M , so we run M on every possible string for every possible number of steps and see which are accepted.

Naming Conventions For The Rest of The Course:

- p, q are states
- a, b, c (letters near the beginning of the alphabet) $\in r$ (They're tape symbols.)
- v, w, x, y, z (letters near the end of the alphabet) $\in r^*$ (They're strings over the alphabet r .)
- \sqcup denotes the blank symbol.

Note:

- A **string** is a finite set of symbols, where each symbol belongs to an **alphabet** denoted by Σ .
- The set of all strings that can be constructed from an alphabet Σ is Σ^* .
- ϵ is the empty string.