Turing Machines

Turing Machine Model

- Introduced by Alonzo Church. Church, along with several mathematicians at that time, was looking for the meaning of effective computability.

Turing Machine Model

- Introduced by Alonzo Church. Church, along with several mathematicians at that time, was looking for the meaning of effective computability.
- Machines were known that were effective in computing the solutions for problems. But on a case by case basis.

Turing Machine Model

- Introduced by Alonzo Church. Church, along with several mathematicians at that time, was looking for the meaning of effective computability.
- Machines were known that were effective in computing the solutions for problems. But on a case by case basis.
- Was there a model that could capture the computability of many such machines?

Turing Machine Model

- Introduced by Alonzo Church. Church, along with several mathematicians at that time, was looking for the meaning of effective computability.
- Machines were known that were effective in computing the solutions for problems. But on a case by case basis.
- Was there a model that could capture the computability of many such machines?
- Turing machine is one such model.

Turing Machine Model

- Introduced by Alonzo Church. Church, along with several mathematicians at that time, was looking for the meaning of effective computability.
- Machines were known that were effective in computing the solutions for problems. But on a case by case basis.
- Was there a model that could capture the computability of many such machines?
- Turing machine is one such model.
- We will see examples of equivalent models for computability.

Turing Machine Model

- We will see later that TM is such a strong model that it can even take as part of the input the description of another machine, and simulate the operation of the input machine on some input string.

Turing Machine Model

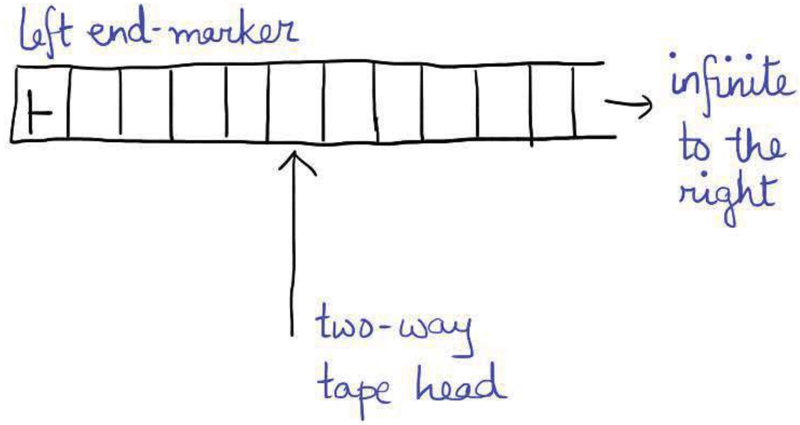
- We will see later that TM is such a strong model that it can even take as part of the input the description of another machine, and simulate the operation of the input machine on some input string.
- This is like taking in a piece of program as input and simulating it.

Turing Machine Model

- We will see later that TM is such a strong model that it can even take as part of the input the description of another machine, and simulate the operation of the input machine on some input string.
- This is like taking in a piece of program as input and simulating it.
- Flipside: We will also see that many problems that try to simulate an input machine are unsolvable by Turing machines.

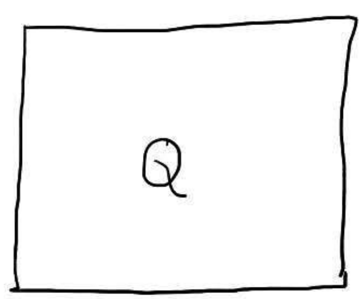
Description of the Turing Machine Model

- Single read-write tape, infinite on the right



- Tape head pointing at current cell, that can move in both directions, but not beyond left end-marker

- Finite set of states
  - Finite control



Finite representation of a TM

- $M = (Q, \Sigma, \Gamma, \vdash, \Box, \delta, s, t, r)$
- Q - finite set of states
- Σ - finite input alphabet, Γ - finite tape alphabet that contains Σ
- $\vdash \in \Gamma - \Sigma$ - left endmarker
- $\Box \in \Gamma - \Sigma$ - blank symbol
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ - transition function
- $s \in Q$ - start state, $t \in Q$ - accept state, $r \neq t \in Q$ - reject state

Transitions

- $\delta(p, a) = (q, b, d)$: If the current state is p and the tape head is scanning a, then write down b in place of a, move tape head towards direction $d \in \{L, R\}$, and transition to state q.

Transitions

- $\delta(p, a) = (q, b, d)$: If the current state is p and the tape head is scanning a, then write down b in place of a, move tape head towards direction $d \in \{L, R\}$, and transition to state q.
- δ is such that $\vdash$ is never overwritten: for any $p \in Q$ there must be a $q \in Q$ such that $\delta(p, \vdash) = (q, \vdash, R)$

Transitions

- $\delta(p, a) = (q, b, d)$: If the current state is p and the tape head is scanning a, then write down b in place of a, move tape head towards direction $d \in \{L, R\}$, and transition to state q.
- δ is such that $\vdash$ is never overwritten: for any $p \in Q$ there must be a $q \in Q$ such that $\delta(p, \vdash) = (q, \vdash, R)$
- Once t or r states are entered, then they cannot be left: for all $b \in \Gamma$ there exist $c, c' \in \Gamma, d, d' \in \{L, R\}$ such that
  (i) $\delta(t, b) = (t, c, d)$
  (ii) $\delta(r, b) = (r, c', d')$.

Configurations

- At any time: $y\square^\omega$ is a semi-infinite string on the read/write tape; $y \in \Gamma^*$ is finite.

Configurations

- At any time: $y\Box^{\omega}$ is a semi-infinite string on the read/write tape; $y \in \Gamma^*$ is finite.
- **Configuration**: $\alpha \in Q \times \{y\Box^{\omega} | y \in \Gamma^*\} \times \mathbb{N}$ gives a snapshot of the TM currently.
  Configuration $\alpha = (p, z, n)$ denotes that the TM is in state p, has z on its read/write tape and the tape head is at the n^{th} position

Configurations

- At any time: $y\Box^\omega$ is a semi-infinite string on the read/write tape; $y \in \Gamma^*$ is finite.
- **Configuration**: $\alpha \in Q \times \{y\Box^\omega | y \in \Gamma^*\} \times \mathbb{N}$ gives a snapshot of the TM currently.
  Configuration $\alpha = (p, z, n)$ denotes that the TM is in state p, has z on its read/write tape and the tape head is at the n^{th} position
- **Start configuration** on input $x \in \Sigma^*$ is unique: $(s, \vdash x\Box^\omega, 0)$

Configurations

- **Next configuration relation** $\rightarrow^1_M$ for a string $z \in \Gamma^*$: let z_n be the n^{th} symbol, and $s^n_b(z)$ be the string obtained from z by replacing z_n by the alphabet $b \in \Gamma$.

Configurations

- **Next configuration relation** $\rightarrow_M^1$ for a string $z \in \Gamma^*$: let z_n be the n^{th} symbol, and $s_b^n(z)$ be the string obtained from z by replacing z_n by the alphabet $b \in \Gamma$.
- If $\delta(p, z_n) = (q, b, L)$, then $(p, z, n) \rightarrow_M^1 (q, s_b^n(z), n-1)$.

Configurations

- **Next configuration relation** $\rightarrow_M^1$ for a string $z \in \Gamma^*$: let z_n be the n^{th} symbol, and $s_b^n(z)$ be the string obtained from z by replacing z_n by the alphabet $b \in \Gamma$.
- If $\delta(p, z_n) = (q, b, L)$, then $(p, z, n) \rightarrow_M^1 (q, s_b^n(z), n - 1)$.
- If $\delta(p, z_n) = (q, b, R)$, then $(p, z, n) \rightarrow_M^1 (q, s_b^n(z), n + 1)$.

Configurations

- **Reflexive transitive closure** $\rightarrow_M^*$ of $\rightarrow_M^1$ defined inductively.

Configurations

- **Reflexive transitive closure** $\rightarrow_M^*$ of $\rightarrow_M^1$ defined inductively.
- $\alpha \rightarrow_M^0 \alpha$,

Configurations

- **Reflexive transitive closure** $\rightarrow^{*}_{M}$ of $\rightarrow^{1}_{M}$ defined inductively.
- $\alpha \rightarrow^{0}_{M} \alpha$,
- $\alpha \rightarrow^{n+1}_{M} \beta$ if $\alpha \rightarrow^{n}_{M} \gamma \rightarrow^{1}_{M} \beta$ for some γ,

Configurations

- **Reflexive transitive closure** $\rightarrow^*_M$ of $\rightarrow^1_M$ defined inductively.
- $\alpha \rightarrow^0_M \alpha$,
- $\alpha \rightarrow^{n+1}_M \beta$ if $\alpha \rightarrow^n_M \gamma \rightarrow^1_M \beta$ for some γ,
- $\alpha \rightarrow^*_M \beta$ if $\alpha \rightarrow^n_M \beta$ for some $n \geq 0$

Acceptance and Rejection

- TM *accepts* $x \in \Sigma^*$: $(s, \vdash x\Box^\omega, 0) \rightarrow^*_M (t, y, n)$ for some y, n. This set is denoted as $L(M)$ for TM M.

Acceptance and Rejection

- TM *accepts* $x \in \Sigma^*$: $(s, \vdash x\Box^\omega, 0) \rightarrow^*_M (t, y, n)$ for some y, n. This set is denoted as $L(M)$ for TM M.
- TM *rejects* $x \in \Sigma^*$: $(s, \vdash x\Box^\omega, 0) \rightarrow^*_M (r, y, n)$ for some y, n.

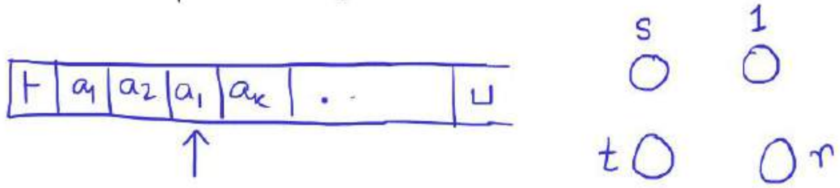
Acceptance and Rejection

- TM *accepts* $x \in \Sigma^*$: $(s, \vdash x\Box^\omega, 0) \rightarrow^*_M (t, y, n)$ for some y, n. This set is denoted as $L(M)$ for TM M.
- TM *rejects* $x \in \Sigma^*$: $(s, \vdash x\Box^\omega, 0) \rightarrow^*_M (r, y, n)$ for some y, n.
- TM *halts* on $x \in \Sigma^*$ if it either accepts or rejects x. It may *loop*.

Design a TM that accepts even length strings.

Language is a <u>regular set</u>

Algorithm: As the tape head moves from $\vdash$ to the rest of the input, the finite control remembers the parity of the input so far.



Transition function:

	$\vdash$	a_1	a_2	$\dots$	a_k	$\sqcup$
s	$(1,\vdash,R)$	$(1,a_1,R)$	$(1,a_2,R)$		$(1,a_k,R)$	$(r,\sqcup,R)$
1	$(r,\vdash,R)$	(s,a_1,R)	(s,a_2,R)		(s,a_k,R)	$(t,\sqcup,R)$
t	$(t,\vdash,R)$	(t,a_1,R)	(t,a_2,R)		(t,a_k,R)	$(t,\sqcup,R)$
r	$(r,\vdash,R)$	(r,a_1,R)	(r,a_2,R)		(r,a_k,R)	$(r,\sqcup,R)$

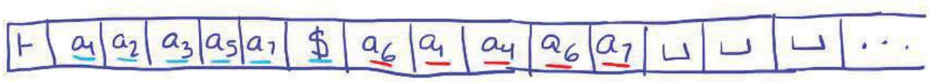
Design a TM that accepts a string if it is of odd length and the middle element is a $ symbol

Language is (a) Not a regular set
(b) CFL

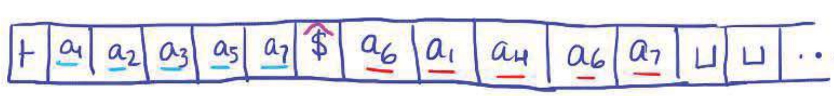
Algorithm:

Step 1: Check if input string is of odd length.

Step 2: Finding the middle element:



Finite control remembers that last blue mark could not be matched with a red mark.



Middle element detected and marked.

Step 3: Check if middle element is \$.

This can be done using finite control: transition to accept state 't' if $\hat{\$}$ is read on the current cell.

Algorithm in words:

1. Check that the string is of odd length: Using the finite control determine the parity of the length of the string in one pass.
2. Determine the middle element: Starting from $\vdash$ mark the first unmarked alphabet in Σ by 'underlining'. Remember this _ marking was done and move right till the rightmost unmarked alphabet in Σ. Mark it by 'overlining'. Remember this ‾ marking and move left till you hit $\vdash$. Repeat above step as long as possible. If at some point the very next cell after a _ marking contains an alphabet marked with ‾, the the previous cell contains the middle element: Remember this using a special state in the finite control and move left.

4. Checking that the middle element is \$: If you are in the special state and the current alphabet is \$, then go to accept state.
5. Anything not working out in steps 1-4, go to reject state 'r'.

Design a TM that writes a copy of the input string after the input string. Eg. If x is the string, in the end the tape contains xx

This is NOT a decision problem - TM can still do this operation.

Algorithm:

- Starting from $\vdash$, when you see first unmarked alphabet $a \in \Sigma$, mark it by rewriting with $\underline{a}$. In finite control, remember a was seen (Σ is finite).
- Move right till you see $\sqcup$ for the first time. Rewrite with $\bar{a}$.
- Continue till there are no unmarked alphabets in the tape. (All are marked with $\_$ or $\bar{\ }$.
- Move left till $\vdash$.
- In one pass to the right, rewrite each $\underline{a}$ by a and each $\bar{a}$ by a.

There is a TM that accepts the language $\{a^n b^n c^n | n \geq 0\}$

The language is not a CFL.

Algorithm:

Step 1: Check if input string is of the form $a^* b^* c^*$.

Step 2: When $\sqcup$ is seen for the first time, it is rewritten with $\dashv$ (right end-marker, good practice, defines right endpoint of string written on the tape)

Step 3: Move left, mark first c, with $\hat{c}$
then first b, with $\hat{b}$
then first a, with $\hat{a}$

Step 4: When $\vdash$ is hit, move right
mark first a, with $\hat{a}$
first b, with $\hat{b}$
first c, with $\hat{c}$

Step 5: Continue making left and right passes.
In each pass, 1 a, b, c gets marked.
If in any pass, this does not happen - reject
If all alphabets are seen to be marked
in a pass - accept.

There is a TM that accepts the language $\{ww | w \in \Sigma^*\}$

Language is not a CFL.

Algorithm:-

Step 1: Check the parity of the input string

Step 2: Mark the first half of the string with $\_$
second half of the string with $\bar{\ }$

Step 3: Move left till $\vdash$ is reached.
Move right to the first alphabet marked with $\_$.
Unmark it (rewrite $\underline{a}$ to a)
In finite control remember a was seen.

Step 4: Move right till first alphabet marked $\bar{\ }$ is hit.
Check if the unmarked version is same as the alphabet remembered in the finite control.
Unmark it (rewrite $\bar{a}$ to a)

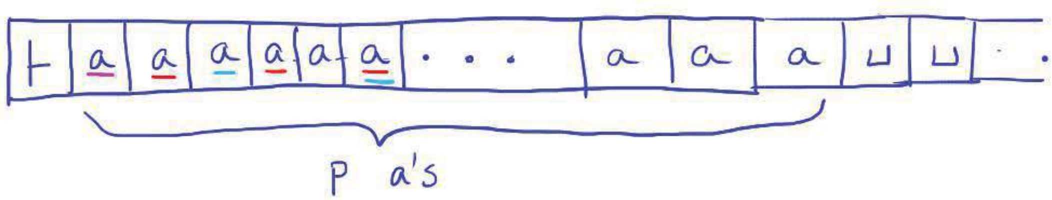
Step 5: Repeat steps 3 and 4 till every alphabet gets unmarked or some mismatch has been detected for rejection.

If no rejection condition has arisen, accept the string.

There is a TM that accepts the language $\{a^p | p \text{ is prime }\}$

The language is not a CFL

Algorithm: Implementing Eratosthenes' sieve:



Step 1: - Mark the first a in the input.

Step 2: - Choose the first unmarked a. Suppose it is at the i^{th} position. Mark all a's in positions that are multiples of i (If a ⊔ is encountered, all multiples are marked)

Step 3: - Continue Step 2 till all a's are marked.

STEP 2: For an i, marking all positions that are multiples of i.

- Mark the i^{th} a as $\hat{a}$. Move to $\vdash$.
- Move right, mark a as a' till $\hat{a}$ is hit (remember in finite control)
- Move left to $\vdash$. For a' to the left match an a after $\hat{a}$ (using markings) and mark this a as a'.
- Upon reaching $\hat{a}$ match to an a, and mark it with $\underset{\vee}{\underline{a}}$.
- Move left to previous $\hat{a}$ or $\underset{\vee}{\underline{a}}$ and match a's in between to a's after $\underset{\vee}{\underline{a}}$ - mark these with a'. At $\underset{\vee}{\underline{a}}$, matched a will be marked with $\underset{\vee}{\underline{a}}$. Continue till $\sqcup$ reached.
- In one pass, unmark all a' to a, and mark all $\hat{a}$ and $\underset{\vee}{\underline{a}}$ to $\underline{a}$.

R.E and Recursive Sets

- A TM may accept, reject or loop on an input string $x \in \Sigma^*$.

R.E and Recursive Sets

- A TM may accept, reject or loop on an input string $x \in \Sigma^*$.
- A TM is *total* if it halts on all inputs: for any input it either accepts or rejects the input.

R.E and Recursive Sets

- A TM may accept, reject or loop on an input string $x \in \Sigma^*$.
- A TM is *total* if it halts on all inputs: for any input it either accepts or rejects the input.
- A set of strings L is:
  (i) *recursively enumerable* (r.e) if $L = L(M)$ where M is a TM.
  (ii) *co-r.e* if $\overline{L}$ is r.e.
  (iii) *recursive* if $L = L(M)$ where M is a total TM (always halts on all inputs).

Decidability and Semidecidability

- A property P of strings in Σ^* is a subset of Σ^*.

Decidability and Semidecidability

- A property P of strings in Σ^* is a subset of Σ^*.
- A property P is *decidable* if the set of strings with P is a recursive set. Eg. The TM for the language $\{ww | w \in \Sigma^*\}$

Decidability and Semidecidability

- A property P of strings in Σ^* is a subset of Σ^*.
- A property P is *decidable* if the set of strings with P is a recursive set. Eg. The TM for the language $\{ww | w \in \Sigma^*\}$
- P is *semidecidable* if the set of strings with P is r.e.

Decidability and Semidecidability

- A property P of strings in Σ^* is a subset of Σ^*.
- A property P is *decidable* if the set of strings with P is a recursive set. Eg. The TM for the language $\{ww | w \in \Sigma^*\}$
- P is *semidecidable* if the set of strings with P is r.e.
- Can you think of an example of a semidecidable set? Will see examples of such properties which are semidecidable but not decidable, etc.