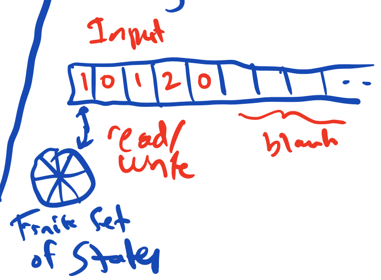
CSE 431
2025
#1

Turing Machines



Q Finite set of states

Σ input alphabet (finite)

Γ tape alphabet (finite)

$\sqcup \in \Gamma \setminus \Sigma$

blank

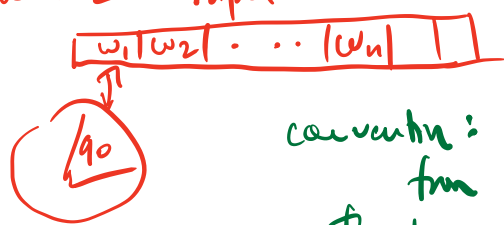
$q_0 \in Q$ start state

$q_{acc} \in Q$ accept state

$q_{rej} \in Q$ reject state

Transition function $\delta : \underbrace{Q}_{\text{old state}} \times \underbrace{\Gamma}_{\text{scanned symbol}} \to \underbrace{Q}_{\text{new state}} \times \underbrace{\Gamma}_{\text{new symbol}} \times \underbrace{\{L, R\}}_{\text{move left or right}}$

$w \in \Sigma^*$ input $|w| = n$



convention: if move from left end of the tape has an L just stay there.

Formalizing Computation: Configurations

- tape contents
- state
- head position

Assume $Q \cap \Gamma = \emptyset$ without loss of generality

So we can understand configuration

configuration $\in \Gamma^* \, Q \, \Gamma \, \Gamma^*$

Γ^*: contents of tape to left of read head

Q: state

Γ: currently scanned symbol

Γ^*: contents of tape to right of read head up to further of:
- last non-blank
- last cell TM has looked at.

typical: $uqav$

TM Operation: $M=(Q,\Sigma,\Gamma,\delta,q_0,q_a,q_r)$ Configuration Yields in one step

$\delta(q,a)=(p,b,R)$: $uqav \vdash_M ubpv$ if $v \neq \varepsilon$ empty string

$ubp\sqcup$ if $v=\varepsilon$ ↑ blank

$\delta(q,a)=(p,b,L)$: $ucqav \vdash_M upcbv$

↑ not at left end of tape

$qav \vdash_M pbv$

Yields $\vdash_M^*$: Two configs C, D

$C \vdash_M^* D$ iff $\exists C_0, C_1, \ldots, C_t$ for some $t \geq 0$ s.t.

$C=C_0, D=C_t$ $(C=C_0 \vdash_M C_1 \vdash_M \cdots \vdash_M C_t=D)$

Start configurations on input w : $q_0 w$ if $w \neq \varepsilon$

$q_0 \sqcup$ if $w=\varepsilon$

Convention: always write $q_0 w$

$uq_{acc}v$ is an **accepting configuration**

$uq_{rej}v$ is a **rejecting configuration**

M **accepts** w iff $q_0 w \vdash_M^*$ an accepting configuration

rejects w iff $q_0 w \vdash_M^*$ a rejecting configuration

$L(M)=\{w \in \Sigma^* : q_0 w \vdash_M^* uq_{acc}v$ for some $u,v \in \Gamma^*\}$

is **language recognized** by M

L is **Turing-recognizable** iff $\exists$ TM M s.t. $L=L(M)$

M is a **decider** iff for all $w \in \Sigma^*$, M accepts w or M rejects w.

L is **decidable** iff $\exists$ decider TM M s.t. $L=L(M)$

Turing Machines

Recall $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$

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

configurations, $\vdash_M$, $\vdash_M^*$

yields in one step — yields

M accepts w iff $q_0 w \vdash_M^* u q_{acc} v$ $\quad u, v \in \Gamma^*$

rejects w iff $q_0 w \vdash_M^* u q_{rej} v$ $\quad u, v \in \Gamma^*$

M is a decider iff $\forall w \in \Sigma^*$
M accepts w or M rejects w

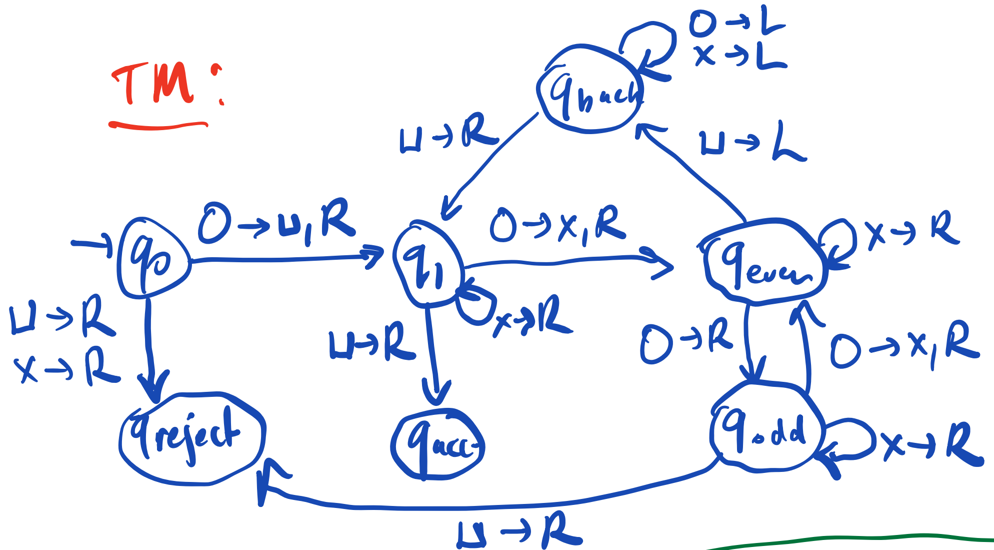
Example: $\{0^{2^n} : n \geq 0\}$ $\quad \Sigma : \{0\}$ input alphabet

Plan:
1. Check if one 0, if yes accept
2. If more than one 0, cross off every second 0 (if odd reject)
3. Repeat above with remaining 0's

eg $0\ \underset{x}{\emptyset}\ 0\ \underset{x}{\emptyset}\ 0\ \underset{x}{\emptyset}\ 0\ \sqcup$ reject

$0\ \underset{x}{\emptyset}\ 0\ \underset{x}{\emptyset}\ 0\ \underset{x}{\emptyset}\ 0\ \underset{x}{\emptyset}\ \sqcup$ need to mark start of the string to get back to mark start of string we could use a new tape symbol but to keep TM simple we use a blank ⊔ representing a 0 and the start.

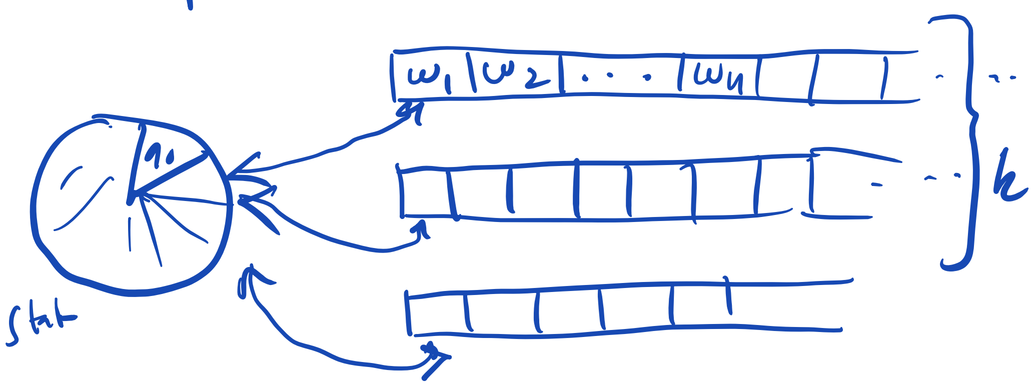
TM:



Notation: $p \xrightarrow{a \to R} q$ means $p \xrightarrow{a \to a, R} q$

Generalizations of TMs:

k-tape TM



Transitions based on all symbols scanned

Input on 1st tape, rest start blank

Head movement independent.

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

Theorem Every k-tape TM M is equivalent to some 1-tape TM M'

equivalent:
- M' accepts w iff M accepts w
- M' rejects w iff M rejects w
- same input alphabet Σ.

Proof Basic idea: represent all k tapes contents on a single tape.

Suppose $M = (Q, \Sigma, \Gamma, \delta, \ldots)$

we create $M' = (Q', \Sigma, \Gamma', \delta', \ldots)$

Let $\# \notin \Gamma$ be a new symbol

We represent all k tapes' contents in M' by

$\# \boxed{\text{tape } 1} \# \boxed{\text{tape } 2} \# \cdots \# \boxed{\text{tape } k} \#$

Since each tape is infinite we only represent the portion that is used. The start a...

We will represent the input tape contents

$\# w_1 w_2 \cdots w_n \# \sqcup \# \sqcup \# \quad \# \sqcup \#$

but we also need to store head position for each tape

We put a $\cdot$ over each char if it also has the head on on it.

$\Gamma' = \Gamma \cup \dot{\Gamma} \cup \{\#\}$?

(*) $\# \dot{w}_1 w_2 \cdots w_n \# \dot{\sqcup} \# \quad \cdots \# \dot{\sqcup} \#$

So... first convert $w_1 \cdots w_n$ input to above string. (*)

To figure out what move to make, need to store scanned symbols in state.

- Do L to R sweep recording the symbols under the dots.
- To execute the moves for this step of M
  - sweep R to L and execute all the left moves
    (rewrite the symbol and dot the symbol just to the left
    - if that symbol was a #
    put that dot back on the first symbol of that tape)
  - sweep L to R and execute all the right moves

most right moves are simple moving the dot one to the right, except when that is # (reached end of used portion of tape) and need to insert a blank symbol and shift the rest of the tape to the right during the sweep.

(to shift by c characters keep track of queue of c most recent chars (in state) reading at one end and writing from the other.)

- Return to the left end.

Note: If original machine ran for T steps (suppose $T \geq n$) new machine may take $O(kT^2)$ steps:

Original step costs = # cells in tapes $\leq kT+k+1$

Total $O(kT^2)$

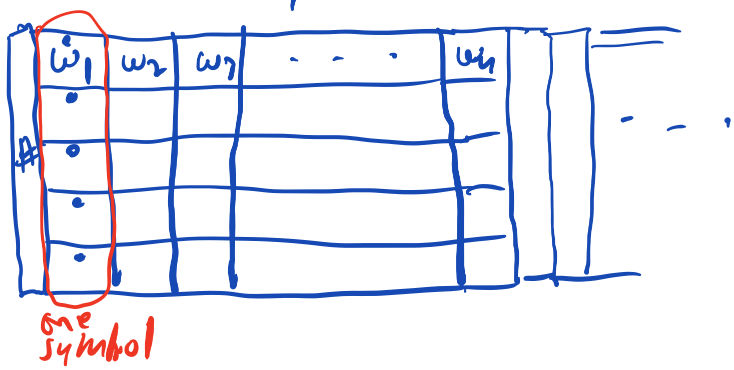
This simulation is step by step and the machine accepts

Note: k was a fixed constant independent of the input. Need to know k to define the 1-tape TM.

☐

Alternative simulation: Multitracked

New set of symbols $\Gamma' = \Gamma \cup (\Gamma \cup \dot{\Gamma})^k$



each tape represented as a "track"

Behaviour is same as before:

- Sweep L to R to collect scanned symbols
- Sweep R to L doing L moves
- Sweep L to R doing R moves
  (maybe make multitracked multi-dotted blanks to replace blanks)
- return to L end.

Time: $O(T^2)$ no shifting required

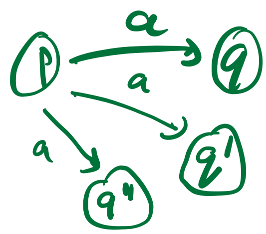
Fact: when converting 2-tapes to 1-tape best possible simulation is $O(T^2)$

Nondeterministic TMs (NTMs)

conceptual, not practical model

Recall DFAs (deterministic) vs NFAs (nondeterministic)

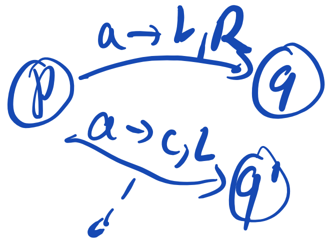
one possible move given p, a



many possible moves given p, a (or none)

We see that NFA's were convenient but for every NFA there was an equivalent DFA (though it requires exponentially more states in the worst case)

With NTMs we get the same option



of multiple possible moves given p, a.

Formally only change is that now

$$\delta : Q \times \Gamma \rightarrow \underbrace{\mathcal{P}}_{\text{power set}}(Q \times \Gamma \times \{L, R\})$$

i.e. a set of possible moves, not just one

Execution of NTM's:

start configuration $q_0 w$

$C \vdash_M D$ iff there some move in δ that takes C to D in one step.

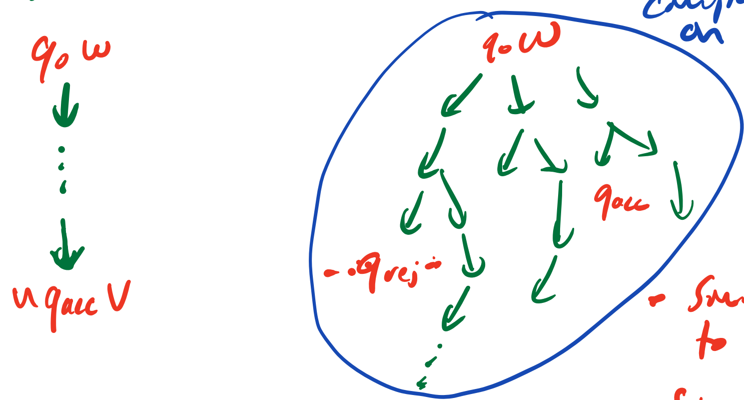
$L(M) = \{ w : q_0 w \vdash_M^* u q_{acc} v \quad u, v \in \Gamma^* \}$

is the language recognized by M.

Use $\rightarrow$ to denote $\vdash_M$

TM

NTM



1 path e.g.

computation tree on input w

many paths in a tree

- Some may lead to q_{reject}
- Some may run forever
- Some may just die off

accept iff there is some q_{acc} somewhere

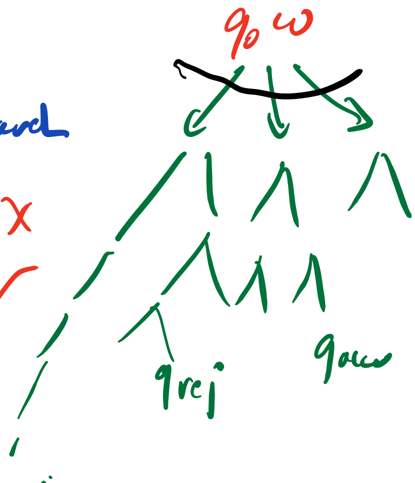
View of Nondeterminism:

- "God's eye" view — accept iff q_{acc} somewhere in infinite tree
- Perfect guesser — at each step if there is a move to lead to q_{accept}, M will take one
- Parallel exploration — M explores all branches in parallel

Theorem: For every NTM there is an equivalent TM

Proof idea: Graph search: explore every node in computation tree, stopping early if q_{accept} is found

Computation tree



Graph Search options

DFS ✗

BFS ✓

bad because might get stuck on infinite path

fan-out of every node is at most $B = |Q \times \Gamma \times \{L,R\}| = 2 \cdot |Q| \cdot |\Gamma|$ which is the max # of possible next moves for a configuration.

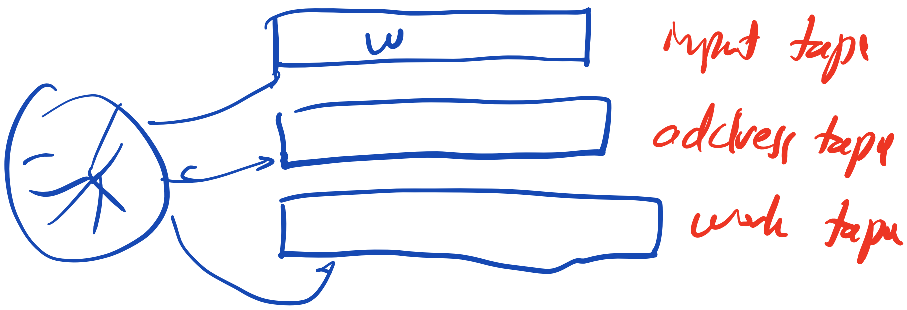
Associate each move with a number from $0, \dots, B-1$

- Each node v (target node) in tree at level t associated with a base B string of length t: address of v.

- Some addresses might not be reachable nodes

Overall idea: loop through all addresses (strings base B) figuring out what the configuration would be at that node and stop & accept iff that accepts

Implementation: 3 tapes



Repeat forever

1. Copy input from input tape to work tape

2. Use address on tape 2 to see which sequence of moves to try
   - Execute each move on work tape if legal for M
   - if not legal abort this address
   - if q_{accept} reached then halt & accept

3. Erase work tape

4. Run machine to convert address on tape 2 to next address (just as with HW1 problem 3 but for bigger B)
   $\varepsilon, 0, 1, \ldots, B-1, 01, 02, \ldots, 0(B-1) \ldots$

This will find a q_{acc} iff there is one

Note: If at some address length t all addresses abort or reject, can reject.

This means that if NTM always halts then TM is a decider □

Suppose original NTM accepted w in T steps

New TM will explore $\sim b^T$ addresses (actually $\sum_{t=0}^{T} b^T$)

This is $2^{O(T)}$ since b is constant. ■

P vs NP question: can we reduce $2^{O(T)}$ to $poly(T)$?

Enumerator TMs

- no input
- read/write work tape initially blank
- write-only 1-way output tape alphabet $\Sigma \cup \{\$\}$ $\$ \notin \Sigma$ (print)

$\$ = $ "\n"

This is a 2-tape TM but all the work is on the first tape

String is printed iff it appears between $ on tape 2.

Language enumerated by M is the set of strings $w \in \Sigma^*$ that M eventually prints (between two \$ on output tape)

$E(M)$

<u>Def^n</u> L is <u>recursively enumerable (r.e.)</u> iff there is some enumerator TM M s.t. $L = E(M)$.

there is an enumerator M s.t. $L = E(M)$

<u>Thm</u> L is recursively enumerable $\iff$ L is Turing-recognizable

i.e. $\exists$ TM M s.t. $L = E(M) \iff \exists$ TM M' s.t. $L = L(M')$

<u>Proof</u> ($\Rightarrow$) Suppose there is an enumerator M s.t. $L = E(M)$

3-tape TM M' recognizing L:

we build M into M' but modify it a bit

- On input w, run M (using 2 other tapes)
- Every time M prints a \$ compare the string it just produced to w. If they are equal accept. If not just continue

Clearly M' will accept w iff
M prints w.

($\Leftarrow$) this is the harder direction

Suppose we have a TM M' with $L = L(M')$

- We need to build an enumerator for L that runs M' on all possible strings in Σ^*

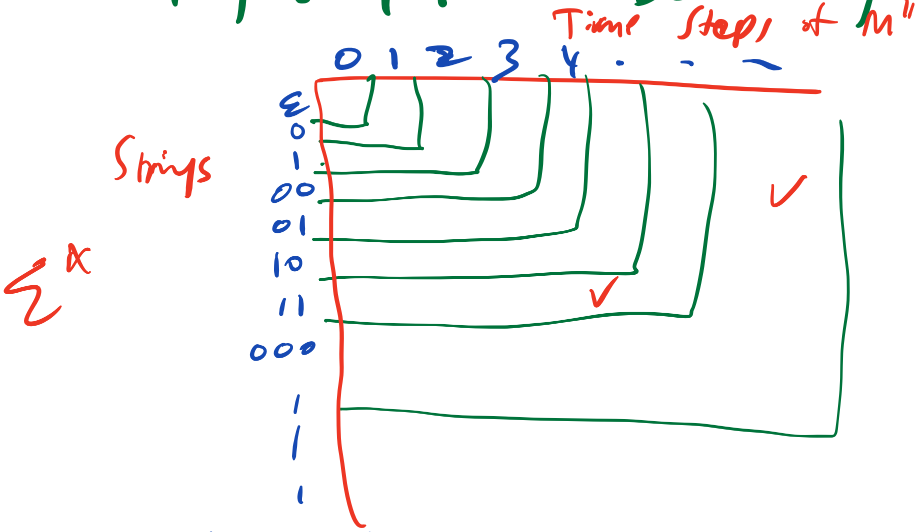
- We can generate all the strings in Σ^* one after another using the counter method we used for addresses

Obvious idea: Every time we produce $w \in \Sigma^*$ run M' on it.

Problem: That's an infinite # of strings but just one computation may run forever!

To fix this we use an idea from Cantor

Key Saving idea: "Dovetailing"



We need to try every string for all possible # of time steps (find all the accept)

Enumerator M''':

For $t = 0 \dots \infty$ do
- For each of the first t strings $w \in \Sigma^*$
  - Run M'' on input w for t steps
  - If M'' accepts print w

Has a modified version of M'' inside it

Eventually, will explore every point in above infinite table so will find and print all accepted strings

$\therefore E(M''') = L$. ∎

Another use of 2-tapes...

<u>Thm</u> If A and $\overline{A}$ are T-rec then A is decidable

<u>Proof</u> Suppose A and $\overline{A}$ are Turing-recognized by TMs M_A and $M_{\overline{A}}$:

Decider M for A:
On input w:
Copy w to a second tape
Run M_A and $M_{\overline{A}}$ one step at a time on each tape "in parallel"
One of M_A or $M_{\overline{A}}$ will halt and accept.
- if M_A accepts then accept
- if $M_{\overline{A}}$ accepts then reject ☐

So far...

k-tape TM ≡ 1-tape TM

NTM ≡ TM

2-dim TM ≡ TM (Homework)

Also TM ≡ Random Access Machine

Random Access Machine

Idealized computer

Just like model for ordinary assembly language except that

- each register holds an arbitrary integer (initially 0)
- there is one register for each natural number 0, 1, 2, ...

Indirect addressing still allowed.

- register indexed uses absolute value

Simulation: Configuration - store value of each register touched between # ~ # on tape

e.g. # i # R_i # (rest are implicitly 0)

i: binary; R_i: value

⇒ C, Java, any programming language + unbounded memory ≡ TM

Also TM ≡ λ-calculus ≡ μ-recursive functions

Turing 1936 Kleene 1937

Church-Turing Thesis (1936)

$\boxed{\text{Any reasonable model that captures all of computation is equivalent in power to Turing machines}}$

Not a statement that can be proved or disproved since it uses a notion "reasonable model" which is not formal

The text phrases this as

$$\boxed{\text{Algorithm} \equiv \text{TM Algorithm}}$$

but this only makes sense if the left side is meant to say "our intuitive notion of algorithm"

Since it was put forth, many other models of computation have been developed and all models are either

- equivalent to TM in power (or weaker) e.g. quantum computers
- allow unreasonable operations with infinite action in a single step