# BU CS 332 – Theory of Computation

https://forms.gle/CrFE8LxSoNBdKe3d8

## Lecture 10:

- Turing Machines
- TM Variants and Closure Properties

Reading:

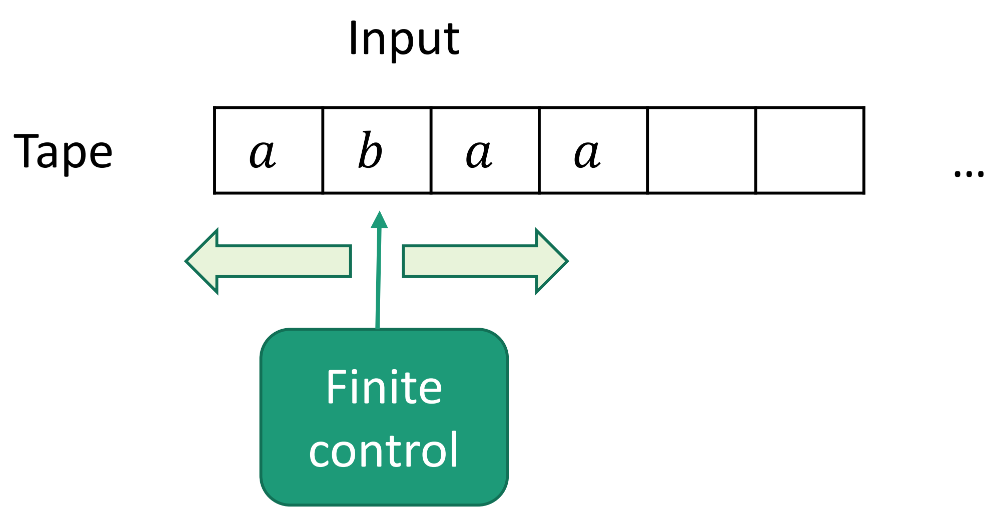
Sipser Ch 3.1-3.3

HW 4 due
Friday 3/1
11:59 PM

Mark Bun

February 28, 2024

# The Basic Turing Machine (TM)




- Input is written on an infinitely long tape
- Head can both read and write, and move in both directions
- Computation halts as soon as control reaches "accept" or "reject" state

# Three Levels of Abstraction

## High-Level Description

An algorithm (like CS 330)

## Implementation-Level Description

Describe (in English) the instructions for a TM

- How to move the head
- What to write on the tape

## Low-Level Description

State diagram or formal specification

# Example

Determine if a string $w \in \{0\}^*$ is in the language

$$A = \{0^{2^n} \mid n \geq 0\}$$

## High-Level Description

Repeat the following forever:

- If there is exactly one $0$ in $w$, accept
- If there is an odd $(> 1)$ number of $0$s in $w$, reject
- Delete half of the $0$s in $w$

# Example

Determine if a string $w \in \{0\}^*$ is in the language

$$A = \{0^{2^n} \mid n \geq 0\}$$

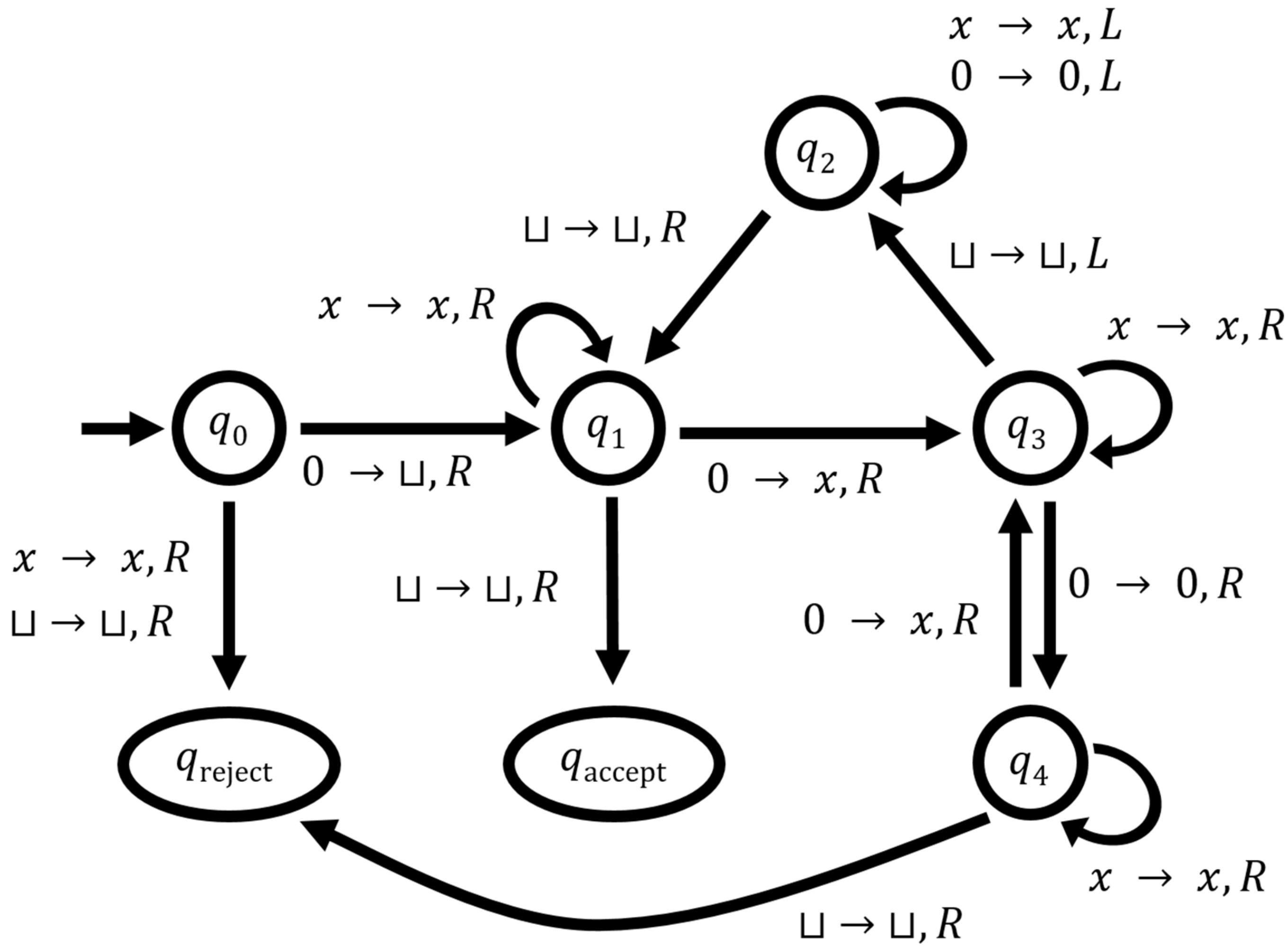
## Implementation-Level Description

1. While moving the tape head left-to-right:
   a) Cross off every other $0$ (i.e., replace it with symbol x)
   b) If there is exactly one $0$ when we reach the right end of the tape, accept
   c) If there is an odd (> 1) number of $0$s when we reach the right end of the tape, reject
2. Return the head to the left end of the tape
3. Go back to step 1

# Example

Determine if a string $w \in A = \{0^{2^n} \mid n \geq 0\}$

## Low-Level Description



# Formal Definition of a TM

A TM is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet (does **not** include ⊔)
- $\Gamma$ is the tape alphabet (contains ⊔ and $\Sigma$) Eg. $X \in \Gamma$
- $\delta$ is the transition function

…more on this later

- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state ($q_{\text{reject}} \neq q_{\text{accept}}$)

# TM Transition Function

$$\delta : \underset{\text{current state}}{Q} \times \underset{\text{current symbol}}{\Gamma} \to \underset{\text{next state}}{Q} \times \underset{\text{next symbol}}{\Gamma} \times \underset{\text{movement instruction}}{\{L, R\}}$$

$L$ means "move left" and $R$ means "move right"

$\delta(p, a) = (q, b, R)$ means:

- Replace $a$ with $b$ in current cell
- Transition from state $p$ to state $q$
- Move tape head right

$\delta(p, a) = (q, b, L)$ means:

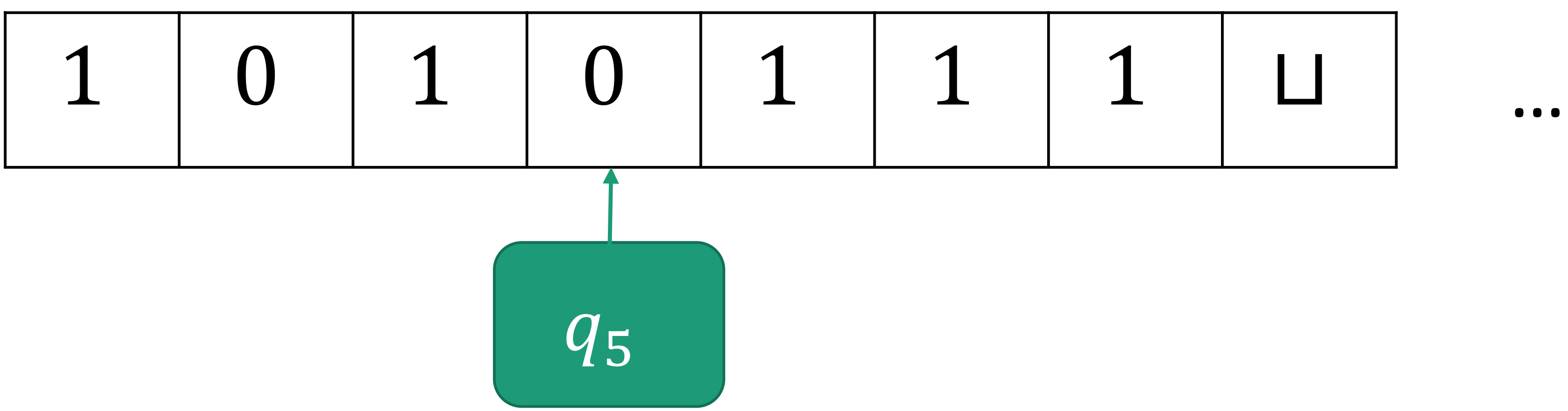
- Replace $a$ with $b$ in current cell
- Transition from state $p$ to state $q$
- Move tape head left UNLESS we are at left end of tape, in which case don't move

# Configuration of a TM: Formally

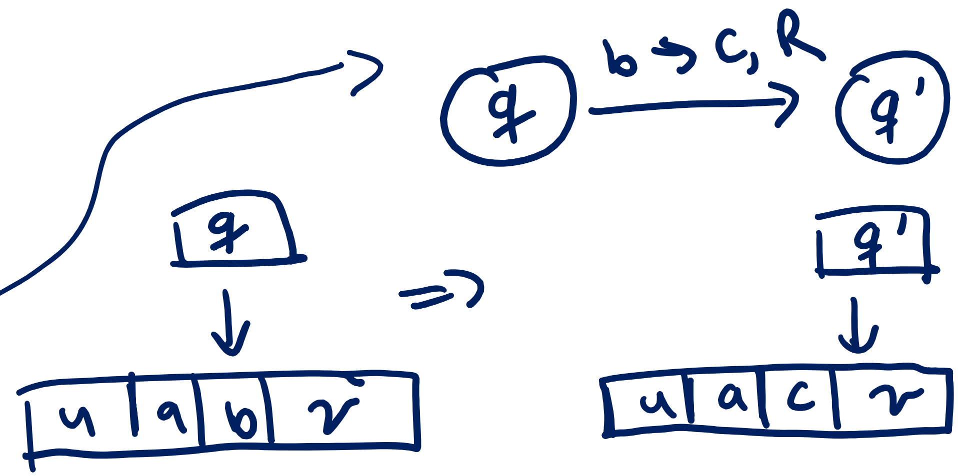
A configuration is a string $uqv$ where $q \in Q$ and $u, v \in \Gamma^*$

- Tape contents = $uv$ (followed by infinitely many blanks ⊔)
- Current state = $q$
- Tape head on first symbol of $v$

Example: $101q_5 0111$



# How a TM Computes




Start configuration: $q_0 w$ ↙ Input

One step of computation:

- If $\delta(q, b) = (q', c, R)$, then $ua\ q\ bv$ yields $uac\ q'\ v$
- If $\delta(q, b) = (q', c, L)$, then $ua\ q\ bv$ yields $u\ q'\ acv$
- If $\delta(q, b) = (q', c, L)$, then $q\ bv$ yields $q'\ cv$

Accepting configuration: $q = q_{\text{accept}}$

Rejecting configuration: $q = q_{\text{reject}}$

# How a TM Computes

$M$ accepts input $w$ if there exists a sequence of configurations $C_1, \ldots, C_k$ such that:

- $C_1 = q_0 w$
- $C_i$ yields $C_{i+1}$ for every $i$
- $C_k$ is an accepting configuration

Can get from config $C_i$ to $C_{i+1}$ in one step of computation

$L(M) =$ the set of all strings $w$ which $M$ accepts

$A$ is Turing-recognizable if $A = L(M)$ for some TM $M$:

- $w \in A \implies M$ halts on $w$ in state $q_{\text{accept}}$
- $w \notin A \implies M$ halts on $w$ in state $q_{\text{reject}}$ OR $M$ runs forever on $w$

Eg.
TM B:
On input w:
Enter an infinite loop
$L(B) = \emptyset$

# Recognizers vs. Deciders

$L(M)$ $=$ the set of all strings $w$ which $M$ accepts

$A$ is Turing-recognizable if $A = L(M)$ for some TM $M$:

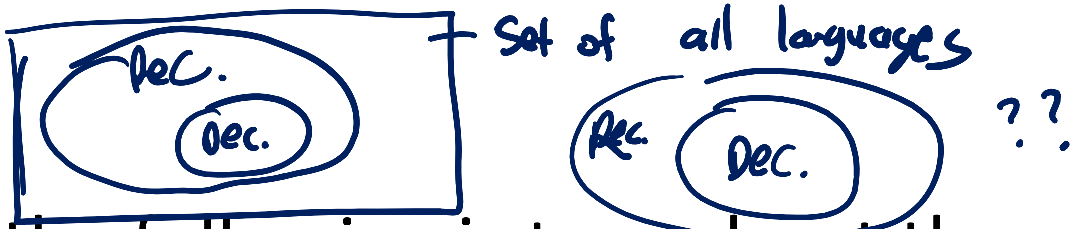
- $w \in A \implies M$ halts on $w$ in state $q_{\text{accept}}$
- $w \notin A \implies M$ halts on $w$ in state $q_{\text{reject}}$ OR
  $M$ runs forever on $w$

A is Turing decidable $\Rightarrow$ A is Turing recognizable

$A$ is (Turing-)decidable if $A = L(M)$ for some TM $M$
which halts on every input

- $w \in A \implies M$ halts on $w$ in state $q_{\text{accept}}$
- $w \notin A \implies M$ halts on $w$ in state $q_{\text{reject}}$

# Recognizers vs. Deciders




Which of the following is true about the relationship between decidable and recognizable languages?

$A$ decidable $\Rightarrow$ $A$ recognizable

$\{$decidable langs.$\} \subseteq \{$recognizable langs$\}$

a) The decidable languages are a subset of the recognizable languages

b) The recognizable languages are a subset of the decidable languages

c) They are incomparable: There might be decidable languages which are not recognizable and vice versa

# Example: Arithmetic on a TM

The following TM decides $\mathrm{MULT} = \{a^i b^j c^k \mid i \times j = k\}$:

On input string $w$: High-level

1. Check $w$ is formatted correctly
2. For each $a$ appearing in $w$:
3. For each $b$ appearing in $w$:
4. Attempt to cross off a $c$. If none exist, reject.
5. If all $c$'s are crossed off, accept. Else, reject.

# Example: Arithmetic on a TM

Ex: ~~a~~ a ~~bbb~~ ~~ccc~~ ccc
~~a~~ a bbb ~~ccc~~ ccc
~~a~~ ~~a~~ ~~bbb~~ ~~ccc~~ ~~ccc~~

The following TM decides $\text{MULT} = \{a^i b^j c^k \mid i \times j = k\}$:

On input string $w$: Implementation-level

1. Scan the input from left to right to determine whether it is a member of $L(a^* b^* c^*)$ Follows from fact that regular langs are recognized by DFAs.
2. Return head to left end of tape
3. Cross off an $a$ if one exists. Scan right until a $b$ occurs. Shuttle between $b$'s and $c$'s crossing off one of each until all $b$'s are gone. Reject if all $c$'s are gone but some $b$'s remain.
4. Restore crossed off $b$'s. If any $a$'s remain, repeat step 3.
5. If all $c$'s are crossed off, accept. Else, reject.

# Back to Hilbert's Tenth Problem

Computational Problem: Given a Diophantine equation, does it have a solution over the integers?

$L = \{ p(x_1,\dots,x_n) \mid p$ is an integer polynomial and $\exists z_1,\dots,z_n \in \mathbb{Z}^n$ s.t. $p(z_1,\dots,z_n) = 0 \}$

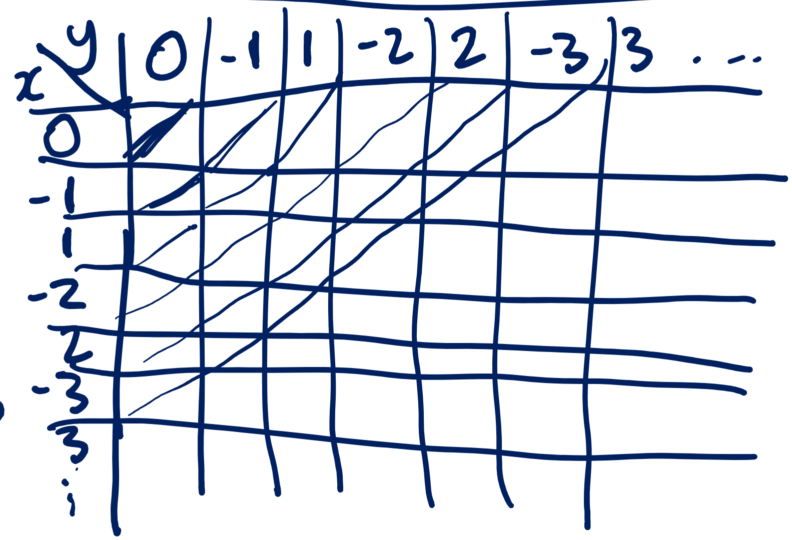
- $L$ is Turing-recognizable

Special case: $p(x,y)$

TM M:

On input $p(x,y)$:

- Try evaluating $p(x,y)$ on inputs in order specified by →
- If any evaluate to 0, accept.



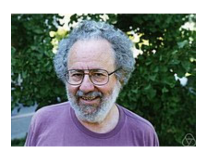
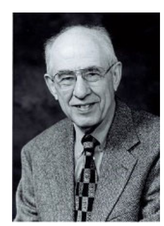

Eval.

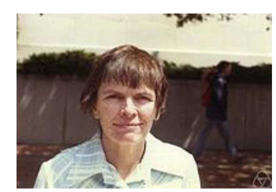
1) $p(0,0)$
2) $p(-1,0)$
3) $p(-1,-1)$
4) $p(1,1)$

⋮

M recognizes $L$. Accepts all $p \in L$, loops forever on all $p \notin L$.

- $L$ is not decidable (1949-70)



# TM Variants

# How Robust is the TM Model?

Does changing the model result in different languages being recognizable / decidable?


So far we've seen…

- We can require that NFAs have a single accept state
- Adding nondeterminism does not change the languages recognized by finite automata

Other modifications possible too: E.g., allowing DFAs to have multiple passes over their input does not increase their power

Turing machines have an astonishing level of robustness

# TMs are equivalent to…

- TMs with "stay put"
- TMs with 2-way infinite tapes
- Multi-tape TMs
- Nondeterministic TMs
- Random access TMs
- Enumerators
- Finite automata with access to an unbounded queue
- Primitive recursive functions ≃ lambda calculus
- Cellular automata

…

# Equivalent TM models

- TMs that are allowed to "stay put" instead of moving left or right

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, \underline{S}\}$$ Stay put

TMs with stay put are *at least* as powerful as basic TMs

(Every basic TM is a TM with stay put that never stays put)

How would you show that TMs with stay put are *no more* powerful than basic TMs?

a) Convert any basic TM into an equivalent TM with stay put
b) Convert any TM with stay put into an equivalent basic TM
c) Construct a language that is recognizable by a TM with stay put, but not by any basic TM
d) Construct a language that is recognizable by a basic TM, but not by any TM with stay put

# Equivalent TM models

- TMs that are allowed to “stay put” instead of moving left or right

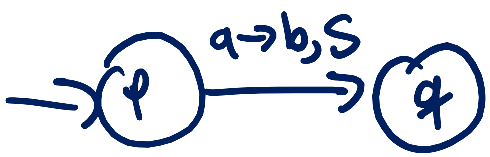
$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$$

Proof that TMs with stay put are no more powerful:

Simulation: Our goal is to convert any TM $M$ with stay put into an equivalent basic TM $M'$

How? Replace every stay put instruction in $M$ with a move right instruction, followed by a move left instruction in $M'$

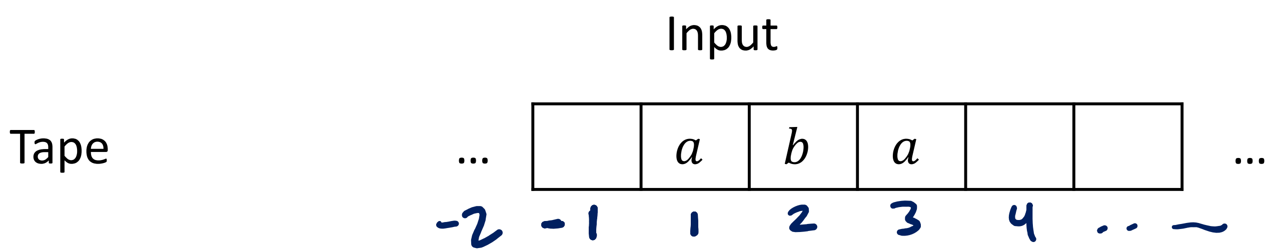
If in M:




In M'

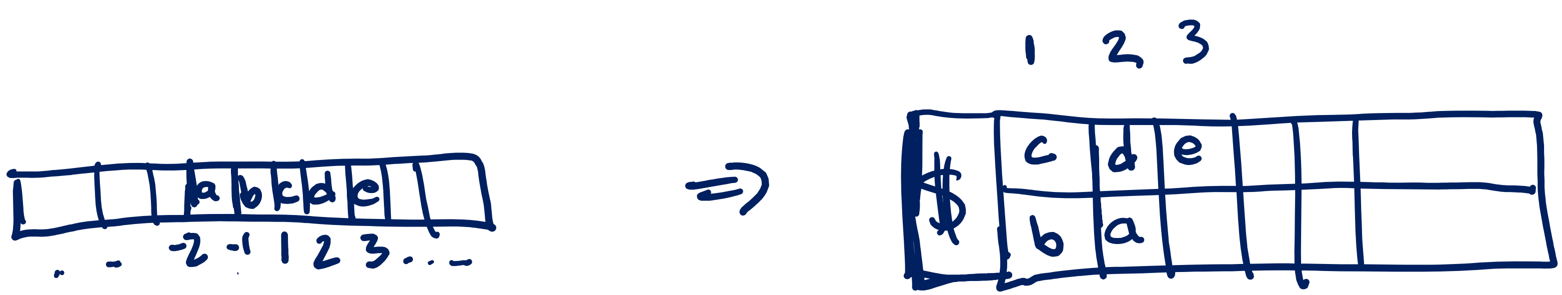
# Equivalent TM models

- TMs with a 2-way infinite tape, unbounded left to right




Proof that TMs with 2-way infinite tapes are no more powerful:

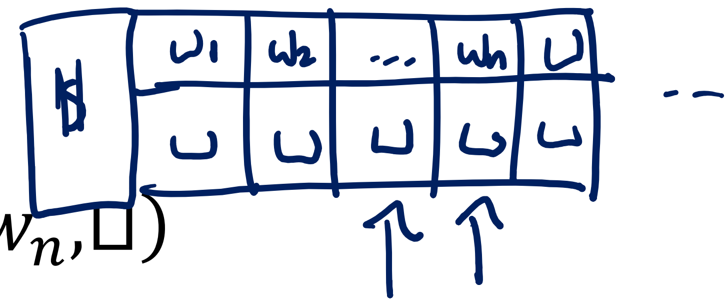
Simulation: Convert any TM $M$ with 2-way infinite tape into a 1-way infinite TM $M'$ with a "two-track tape"



# Implementation-Level Simulation

Given 2-way TM $M$ construct a basic TM $M'$ as follows.

TM $M'$ = "On input $w = w_1 w_2 \ldots w_n$:




1. Format 2-track tape with contents

$$\$, (w_1,\sqcup), (w_2,\sqcup), \ldots, (w_n,\sqcup)$$

2. To simulate one move of M:

a) If working on upper track, read/write to the first position of cell under tape head, and move in the same direction as $M$

b) If working on lower track, read/write to second position of cell under tape head, and move in the opposite direction as $M$

c) If move results in hitting \$, switch to the other track. "

# Formalizing the Simulation

Given 2-way TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, construct $M' = (Q', \Sigma, \Gamma', \delta', q_0', q_{\text{accept}}', q_{\text{reject}}')$

upper track  lower track

New tape alphabet: $\Gamma' = (\Gamma \times \Gamma) \cup \{\$\}$

New state set: $Q' = Q \times \{+, -\}$

$(q, +)$ means "in state $q$ and working on upper track"

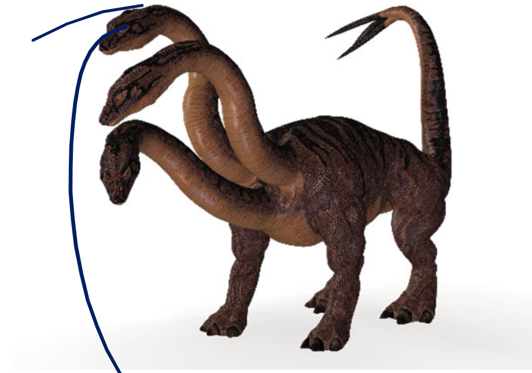
$(q, -)$ means "in state $q$ and working on lower track"

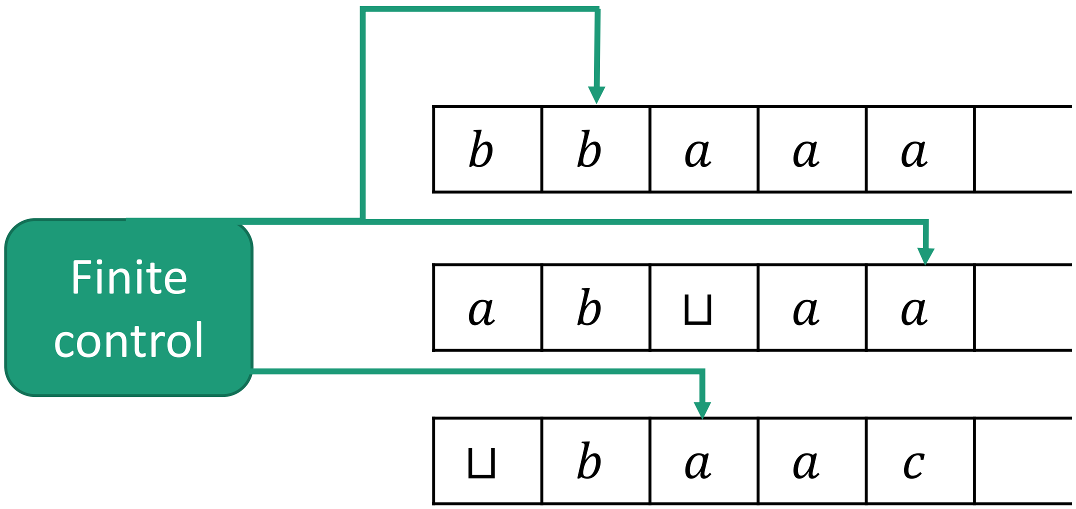
New transitions:

If $\delta(p, a_-) = (q, b, L)$, let $\delta'\big((p, -), (a_-, a_+)\big) = ((q, -), (b, a_+), R)$

Also need new transitions for moving right, lower track, hitting $\$$, initializing input into 2-track format

# Multi-Tape TMs






Fixed number of tapes $k$

**(*$k$* can't depend on input or change during computation)**

Transition function $\delta : Q \times \Gamma^{k} \rightarrow Q \times \Gamma^{k} \times \{L, R, S\}^{k}$

current state

k symbols read

new state

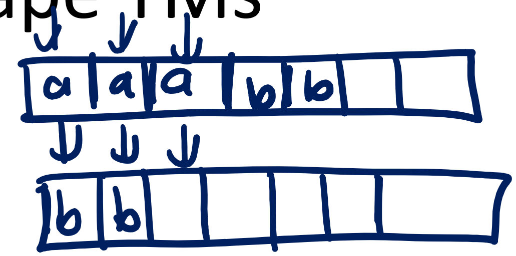
k symbols written

k movements

# Why are Multi-Tape TMs Helpful?

To show a language is Turing-recognizable or decidable, it's enough to construct a multi-tape TM

Often easier to construct multi-tape TMs




Ex. Decider for $\{a^i b^j | i > j\}$

On input $w$:

1) Check in one left-to-right pass that $w \in L(a^* b^*)$
2) Copy all b's to tape 2
3) Check one character at a time that every b has a matching a
4) Check at least one a left over.