BU CS 332 – Theory of Computation

https://forms.gle/HCgscGEzevjHXcAg8

Lecture 10:

- Turing Machines
- TM Variants and Closure Properties

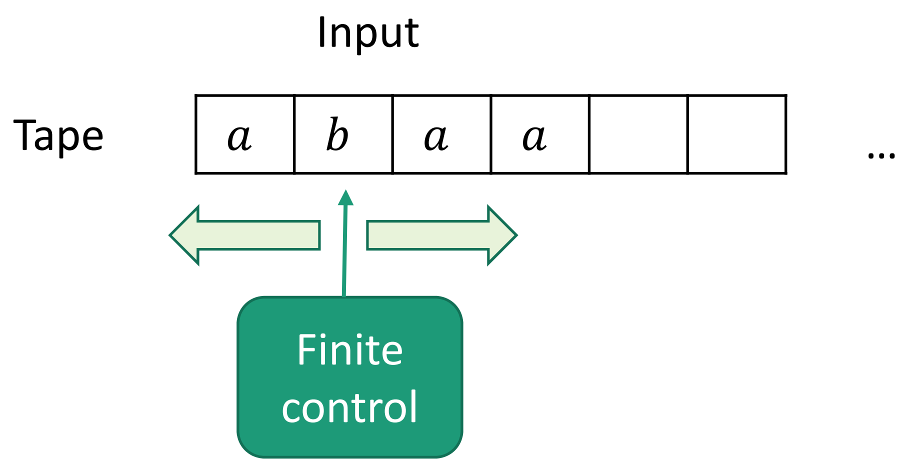
Reading:

Sipser Ch 3.1-3.3

Mark Bun

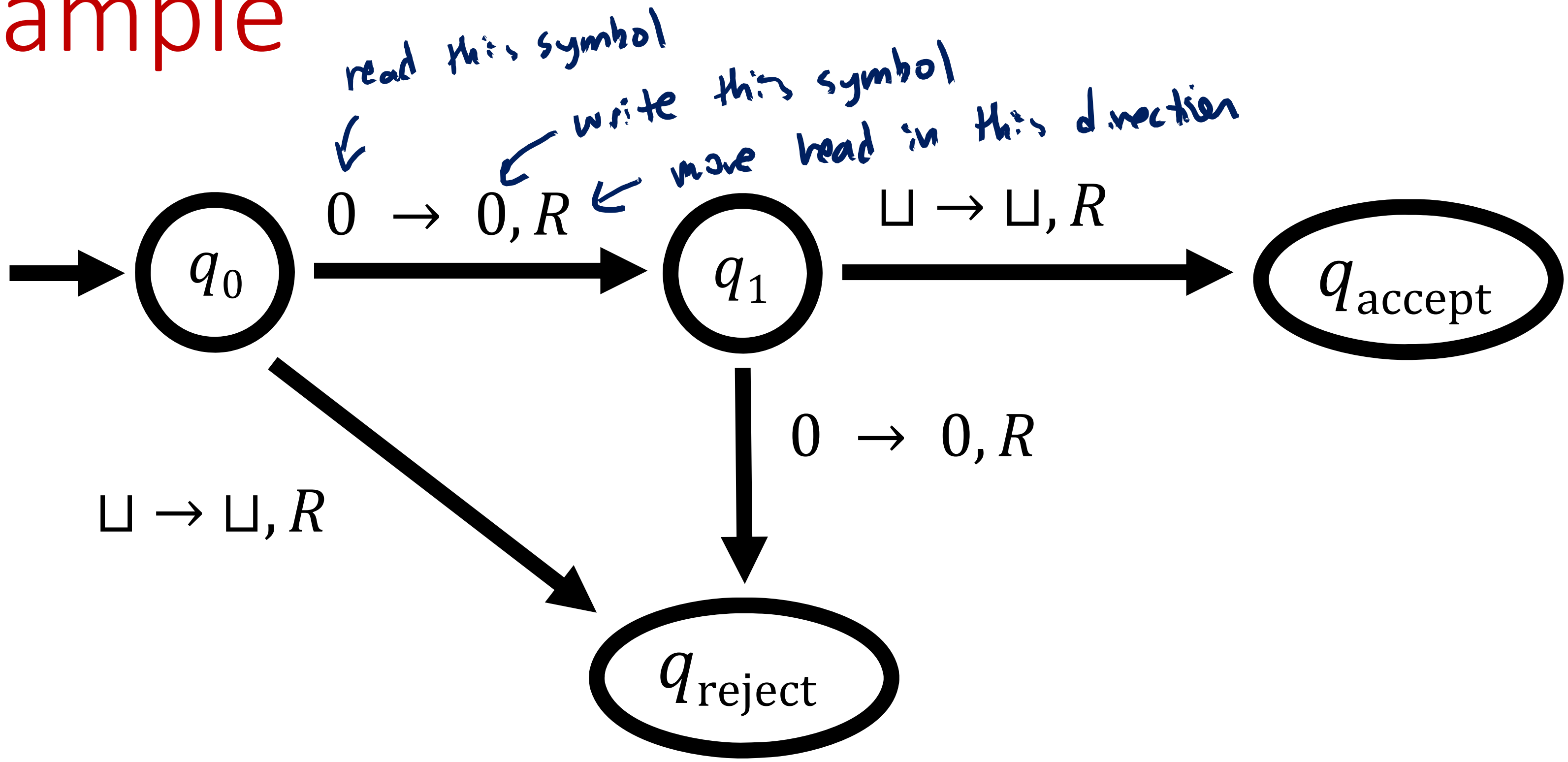
October 7, 2021

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

Example


read this symbol
write this symbol
move head in this direction
$0 \rightarrow 0, R$
q_0
q_1
$\sqcup \rightarrow \sqcup, R$
q_{accept}
$0 \rightarrow 0, R$
$\sqcup \rightarrow \sqcup, R$
q_{reject}

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
- Σ is the input alphabet (does **not** include ⊔)
- Γ is the tape alphabet (contains ⊔ and Σ)
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function

  new symbol

  $\delta(q, a) = (q', b, m)$

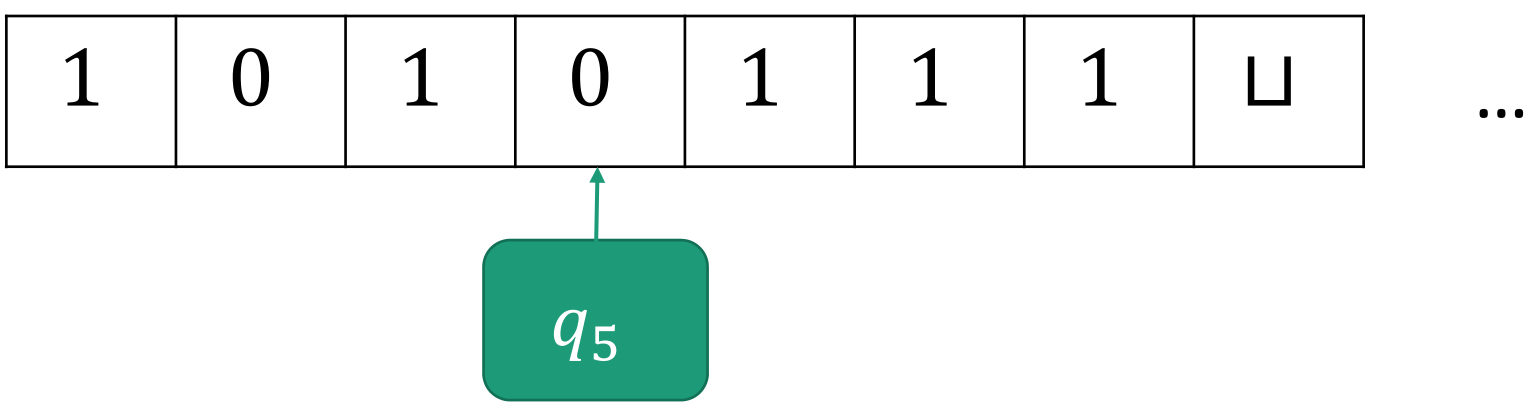
  current state   curr. symbol   new state   direction head moves (L or R)
- $q_0 \in Q$ is the start state
- $q_{\text{accept}} \in Q$ is the accept state
- $q_{\text{reject}} \in Q$ is the reject state ($q_{\text{reject}} \neq q_{\text{accept}}$)

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 blanks ⊔)
- Current state = q
- Tape head on first symbol of v

Example: $101q_50111$



How a TM Computes

Start configuration: $q_0 w$

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_{accept}

Rejecting configuration: q = q_{reject}

How a TM Computes

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

- $C_1 = q_0 w$ C_1 is start configuration
- C_i yields C_{i+1} for every i Can get from C_i to C_{i+1} by one step of computation
- C_k is an accepting configuration

$L(M) =$ 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_{accept}
- $w \notin A \implies M$ halts on w in state q_{reject} OR
  M runs forever on w

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

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

Recognizers vs. Deciders

(A TM decider for language A is also a TM recognizer for A) $\Rightarrow$ (A decidable $\Rightarrow$ A recognizable)
$\Rightarrow$ decidable langs. $\subseteq$ recognizable langs.

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

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

High-level description

On input string w:

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.

Check if $w \in L(a^* b^* c^*)$
Can do w/ DFA $\Rightarrow$ can do in one read-only pass of a TM

$i \times j > k$

$i \times j = k$

$i \times j < k$

Example: Arithmetic on a TM

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 $L(a^* b^* c^*)$

   ~~a~~a ~~bbb~~ ~~ccc~~ccc

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.

   ~~a~~ ~~a~~ ~~bbb~~ ~~ccc~~ ~~ccc~~

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.

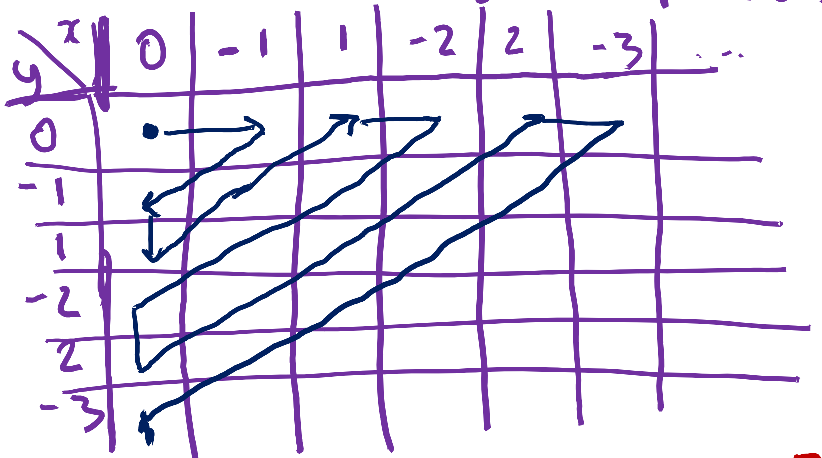
Back to Hilbert's Tenth Problem

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

$L = \{ p(z_1, \ldots, z_n) \mid p$ is a polynomial w/ integer coeffs. and $\exists (z_1, \ldots, z_n) \in \mathbb{Z}^n$ s.t. $p(z_1, \ldots, z_n) = 0 \}$

- L is Turing-recognizable

Special case: $L_2 = \{ p(x,y) \mid \exists x, y \;\; p(x,y) = 0 \}$

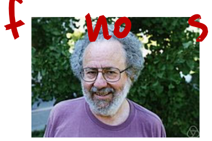
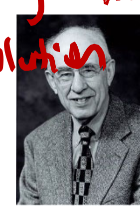
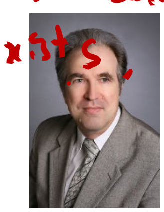


Alg: On input p:

- Test $p(0,0) \stackrel{?}{=} 0$, if so, accept
- Test $p(-1,0) \stackrel{?}{=} 0$, if so, accept
- Test $p(0,-1) \stackrel{?}{=} 0$, ...

Analysis:

- If $\exists x, y$ s.t. $p(x,y) = 0$, alg. will eventually find it and accept
- If no solution exists, alg looks forever

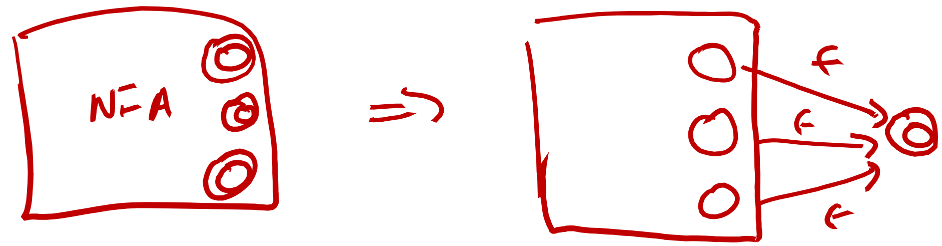





- L is not decidable (1949-70)

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 subset construction

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   λ calculus
- Cellular automata

…

Equivalent TM models

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

"stay put"

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

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 "TM with stay put can be more powerful than basic TM"

d) Construct a language that is recognizable by a basic TM, but not by any TM with stay put "Basic TM can be more powerful than TM w/ stay put"

Equivalent TM models

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

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

Proof that TMs with stay put are no more powerful:

Simulation: Convert any TM M with stay put into an equivalent basic TM M'

"Implementation level"

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

If I have

$$p \xrightarrow{a \to b,\, S} q$$

in M

$$\Longrightarrow$$

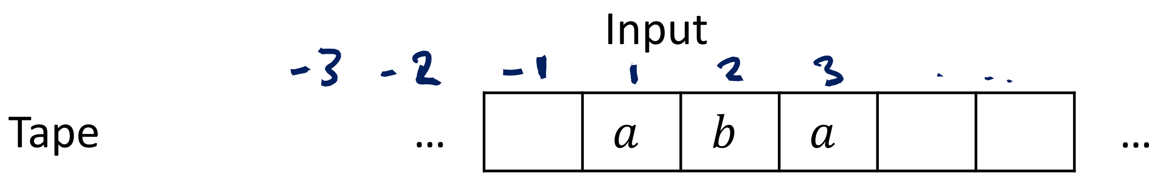
$$p \xrightarrow{a \to b,\, R} q' \xrightarrow{\sigma \to \sigma,\, L} q$$

(for all $\sigma \in \Gamma$)

in M'

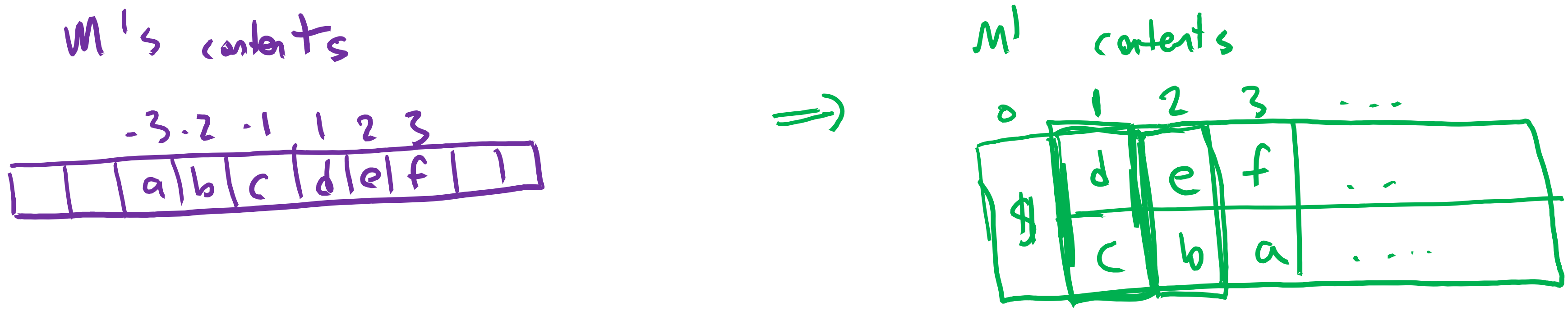
Equivalent TM models

- 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 $w = w_1 w_2 \dots w_n$:

1. Format 2-track tape with contents

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

means

\$	w_1	w_2	...	w_n	...
	$\sqcup$	$\sqcup$		$\sqcup$	...

How to format input

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 $M' = (Q', \Sigma, \Gamma', \delta', q_0', q_{\text{accept}}', q_{\text{reject}}')$

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

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

$(q, -)$ means "q, working on upper track"

$(q, +)$ means "q, working on lower track"

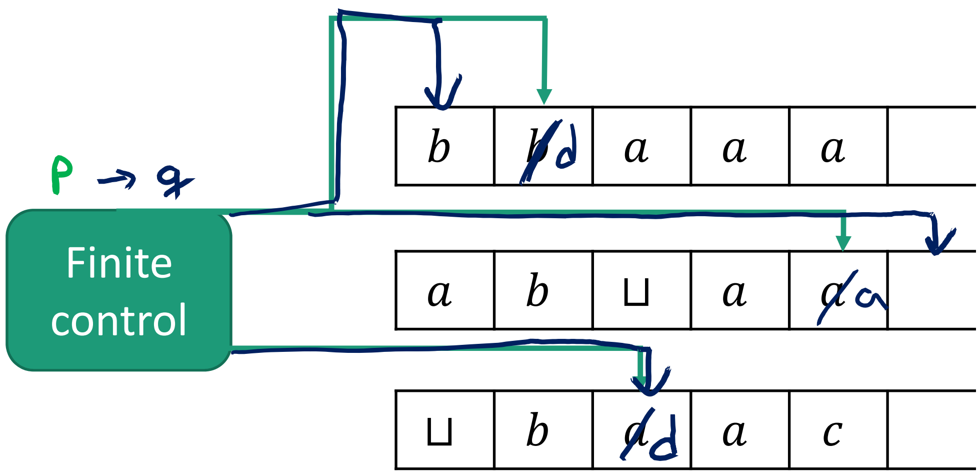
New transitions:

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

$\delta(p,(b,a,a)) = (q,(d,a,d),(L,R,S))$



Convention:

Input written on tape 1, read-only

"work tapes"

Fixed number of tapes k

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

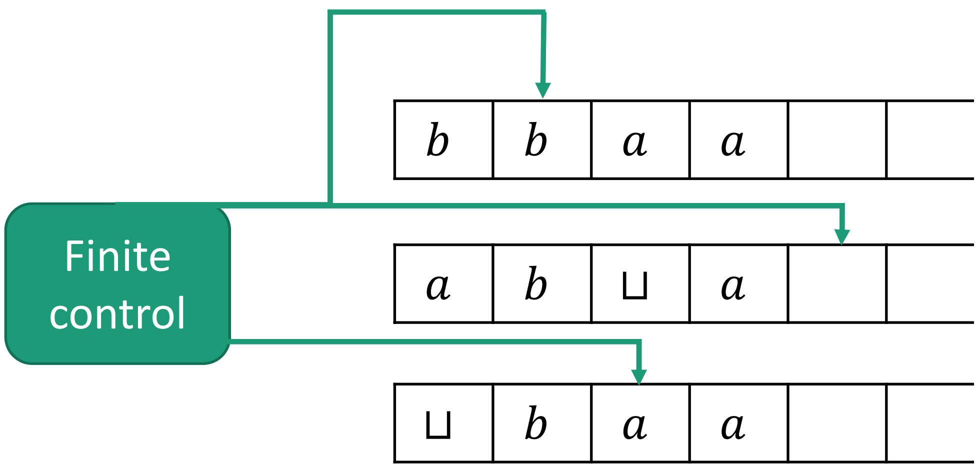
new state; write k symbols $(b_1, \dots, b_k)$

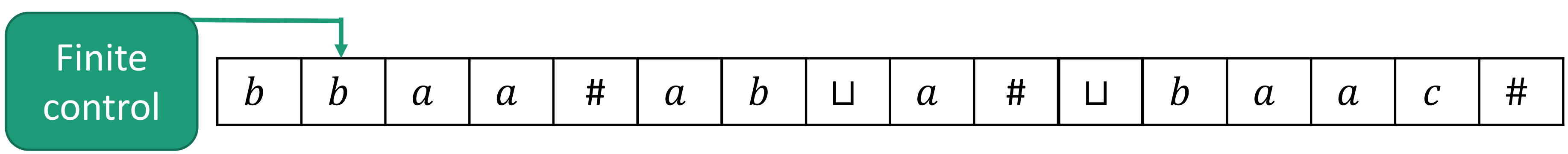
Transition function $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \underbrace{\{L, R, S\}^k}_{\text{k left/right/stay instructions}}$

current state; k symbols to read $(a_1, \dots, a_k)$

Multi-Tape TMs are Equivalent to Single-Tape TMs

Theorem: Every k-tape TM M with can be simulated by an equivalent single-tape TM M'

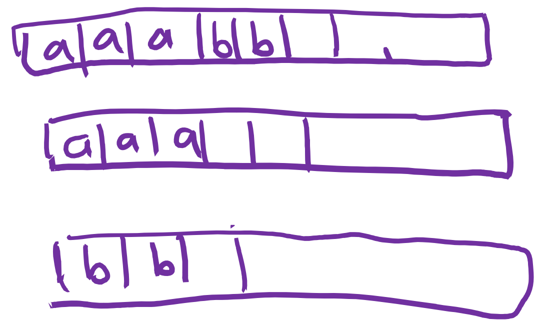




Why are Multi-Tape TMs Helpful?

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 $\{a^i b^j | i > j\}$

Three tape TM decider

On input w: (w written on tape 1)

1) Format check: Reject if $w \notin L(a^* b^+)$
2) Copy all a's from w to tape 2
3) Copy all b's from w to tape 3
4) Return heads to left ends of tapes 2 and 3
   Scan tapes 2 & 3 left-to-right. If hit a blank on tape 3, accept iff still any a's left on tape 2.