

BU CS 332 – Theory of Computation

<https://forms.gle/skecSMqYiyaXE9Kq7>



Lecture 10:

- TM Variants
- TM Closure Properties

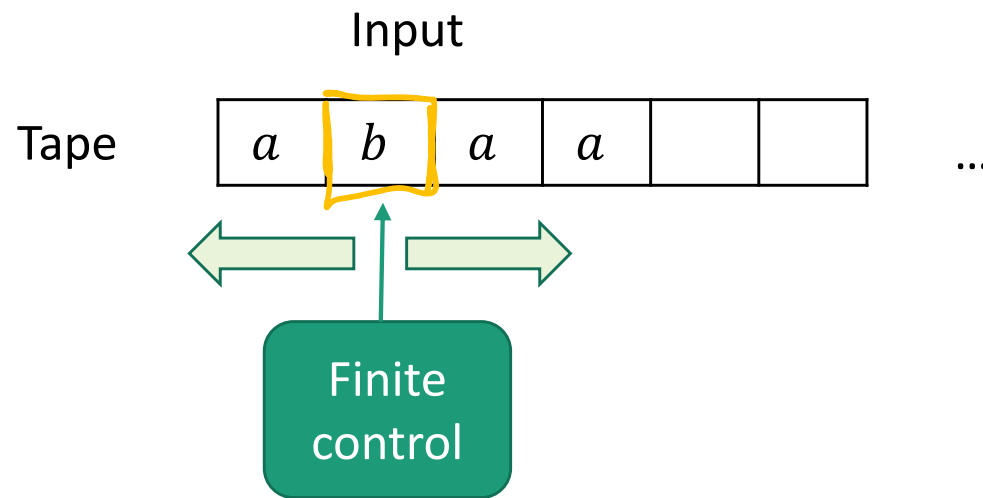
Reading:

Sipser Ch 3.1-3.3

Mark Bun & Alexander Poremba

March 3, 2025

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

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 \sqcup**) *Example: $\Sigma = \{a, b\}$*
- Γ is the tape alphabet (**contains \sqcup and Σ**) *$\Gamma = \{a, b, \sqcup, x\}$*
- δ 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

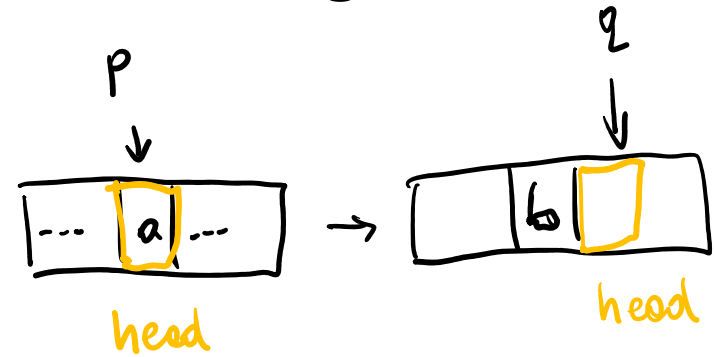
$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$

new symbol
 move right
 move left
 current state
 current symbol on tape
 next state

L means “move left” and R means “move right”

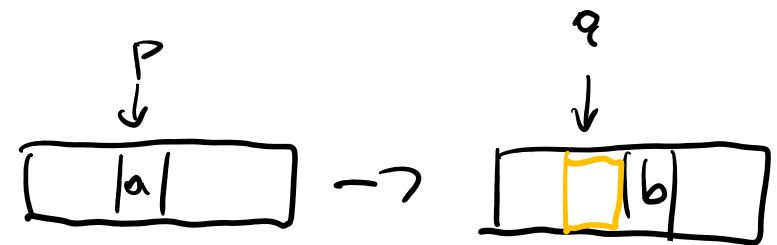
$\delta(p, a) = (q, b, R)$ means:

- 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

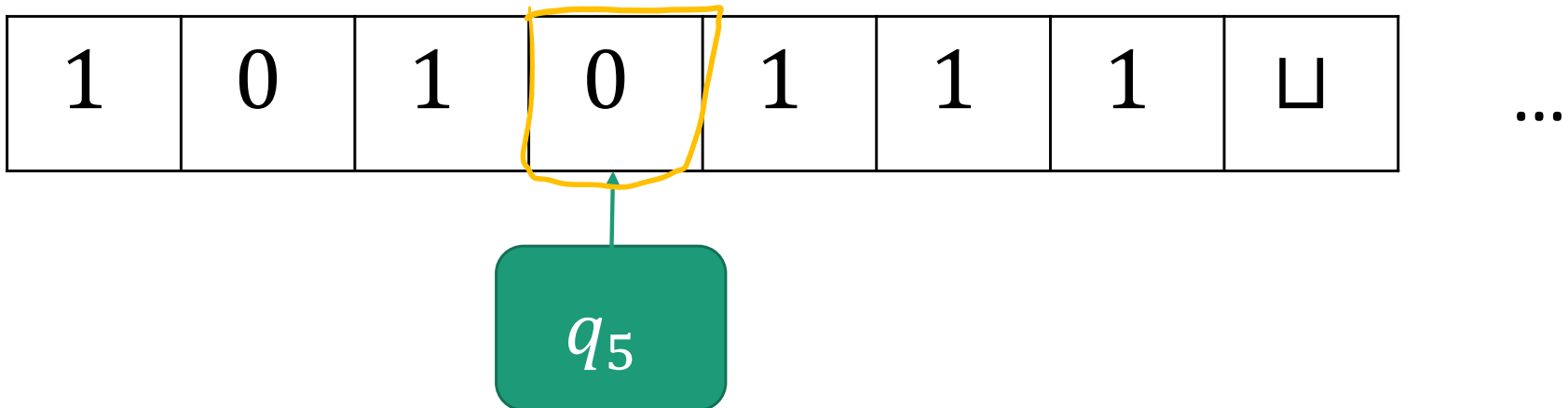


Configuration of a TM: Formally

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

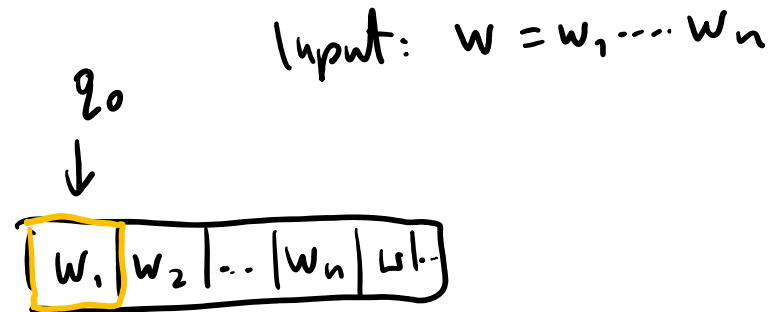
- Tape contents = uv (followed by infinitely many blanks \sqcup)
- Current state = q
- Tape head on first symbol of v

Example: $101q_50111$



How a TM Computes

Start configuration: $q_0 w$



In one step of computation:

- If $\delta(q, b) = (q', c, R)$, then $uaqbv$ yields $uacq'v$
- If $\delta(q, b) = (q', c, L)$, then $uaqbv$ yields $uq'acv$
- If $\delta(q, b) = (q', c, L)$, then qbv yields $q'cv$

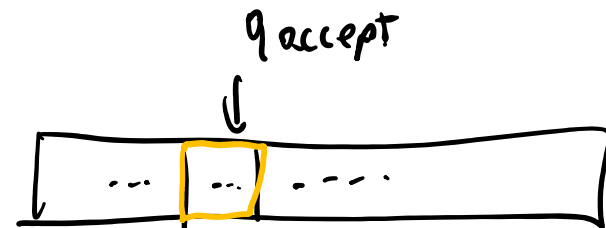
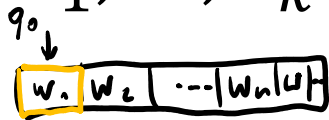
Accepting configuration: $q = q_{\text{accept}}$

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

How a TM Computes

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

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



def.

$L(M)$ = 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



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

$$\{\text{TM decidable languages}\} \subsetneq \{\text{TM recognizable languages}\}$$

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

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**.

Example: Arithmetic on a TM

$\begin{array}{cccccccc} \times & & & \times & \times & \times & \times & \\ \bar{a} & a & a & b & b & c & c & c & c & c \\ \times & & & \times & \times & \times & \times & \times & \times & \times \\ \bar{a} & a & a & b & b & c & c & c & c & c \\ \times & \times & & \times & \times & \times & \times & \times & \times & \times \\ \bar{a} & \bar{a} & a & b & b & c & c & c & c & c \end{array}$

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

On input string w :

$\begin{array}{cccccccc} \times & \times & & & & \times & \times & \times & \times & \\ \bar{a} & \bar{a} & a & b & b & c & c & c & c & c \\ \times & \times & \times & \times & \times & \times & \times & \times & \times & \times \\ \bar{a} & \bar{a} & a & b & b & c & c & c & c & c \end{array}$

1. Scan the input from left to right to determine whether it is a member of $L(a^* b^* c^*)$
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**.

TM Variants

How Robust is the TM Model?

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

So far we've seen...

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

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

Turing machines have an **astounding** 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
- Cellular automata
- ...



Equivalent TM models

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

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

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?

“simulation argument”

- 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

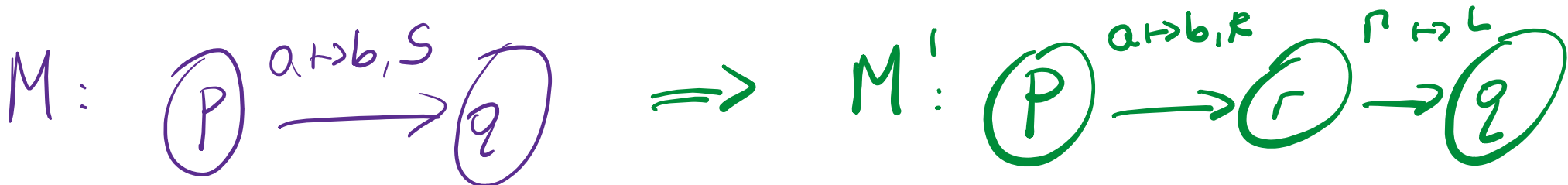
- TMs that are allowed to “stay put” instead of moving left or right

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

Proof that TMs with stay put are no more powerful:

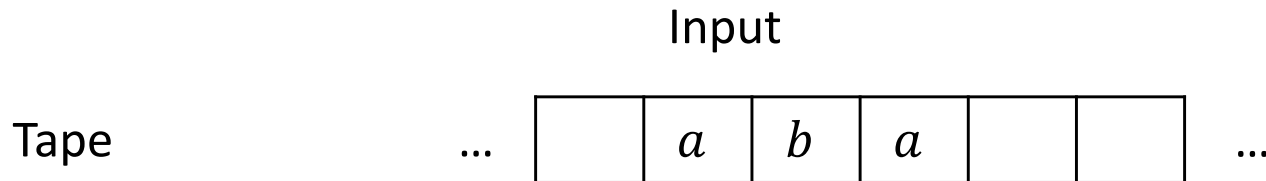
Simulation: Our goal is to convert any TM M with stay put into an equivalent basic TM M'

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



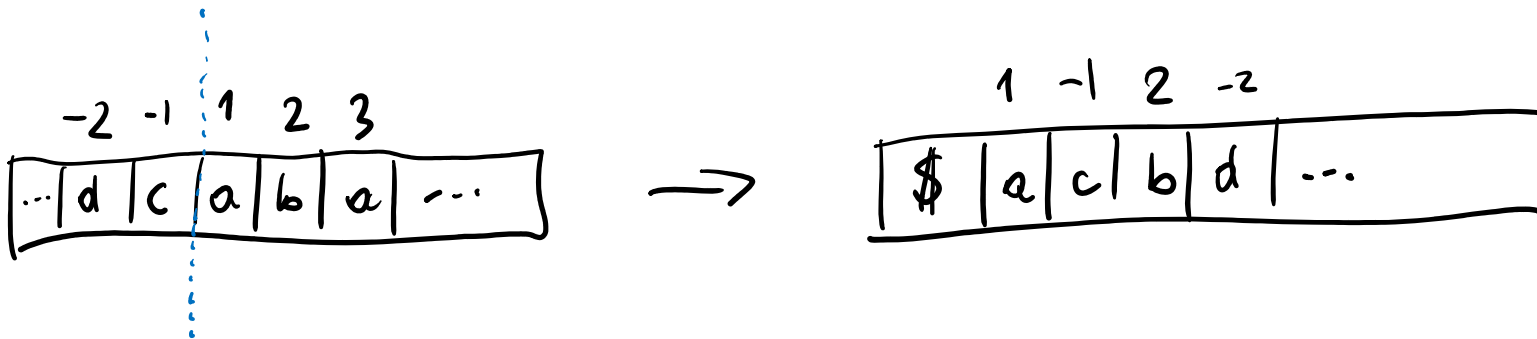
Equivalent TM models

- 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 $w = w_1w_2 \dots w_n$:

1. Format 2-track tape with contents

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



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 $M' = (Q', \Sigma, \Gamma', \delta', q'_0, q'_{\text{accept}}, q'_{\text{reject}})$

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

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

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

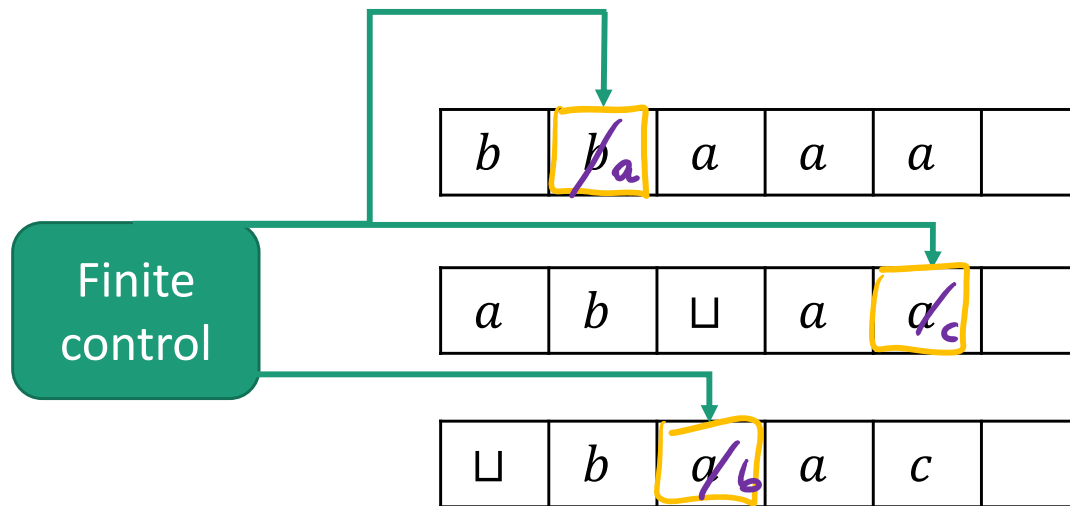
$(q, -)$ means “in state q and working on lower track”

New transitions:

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

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

Multi-Tape TMs



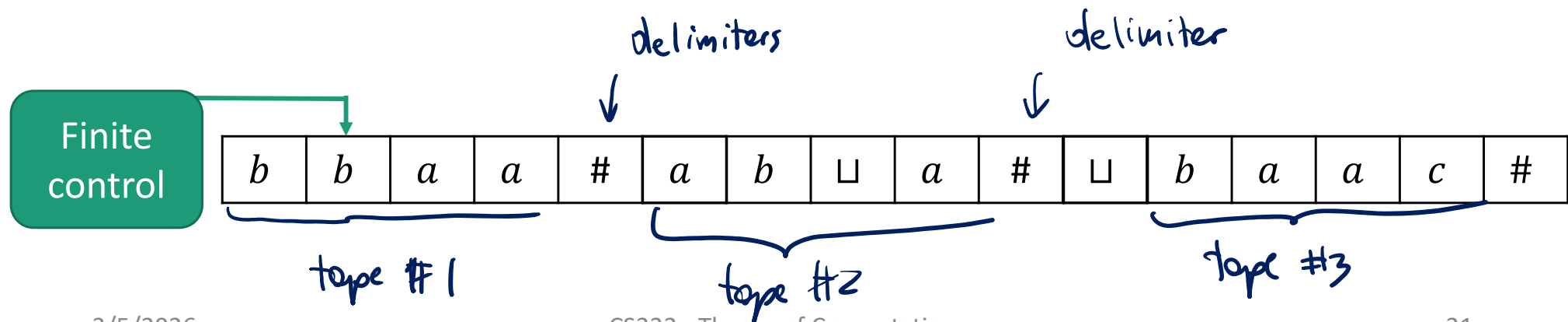
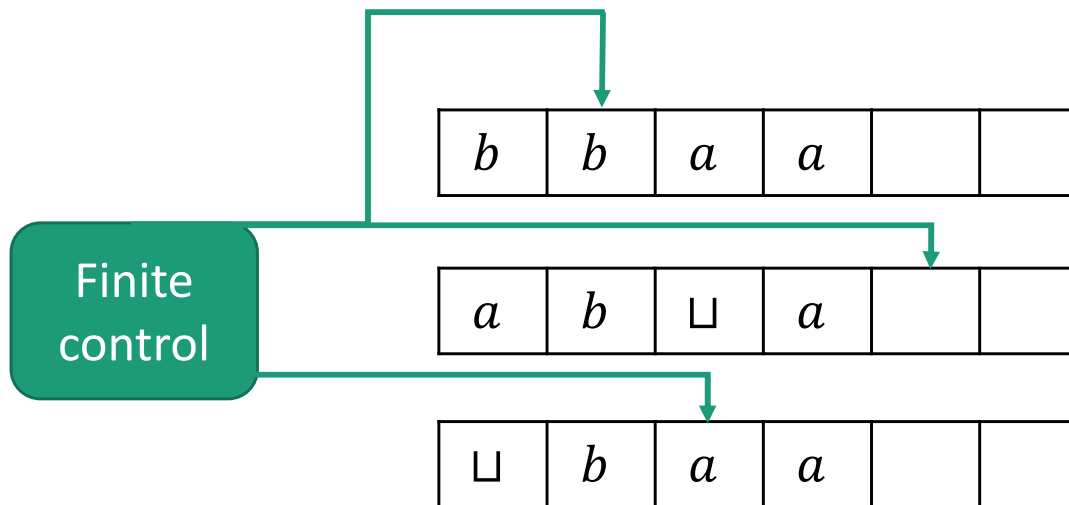
Fixed number of tapes k

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

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

Multi-Tape TMs are Equivalent to Single-Tape TMs

Theorem: Every k -tape TM M 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 $\{a^i b^j \mid i > j\}$

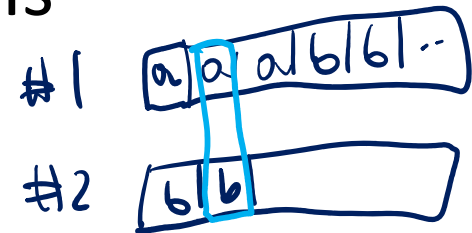
Why are Multi-Tape TMs Helpful?

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

On input w :



1. Scan tape 1 left-to-right to check that $w \in L(a^*b^*)$.
2. Scan tape 2 left-to-right to copy all b 's to tape 2.
3. Starting from the left ends of each tape, scan to check that every b on tape 2 has an accompanying a on tape 1. If not, **reject**.
4. Check that the first blank on tape 2 has an accompanying a on tape 1. If so, **accept**; otherwise, **reject**.

Why are Multi-Tape TMs Helpful?

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

Very helpful for proving **closure properties**

Example: Closure of recognizable languages under union.

If L_1 and L_2 are both Turing-recognizable, is it the case that

$$L_1 \cup L_2$$

is also Turing-recognizable?

Closure Properties

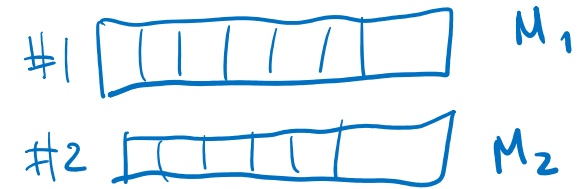
Fact: The Turing-recognizable languages are closed under the union operation:

If L_1 and L_2 are both Turing-recognizable, then $L_1 \cup L_2$ is also Turing-recognizable.

First attempt: Construct multi-tape TM N

On input $w = w_1 w_2 \dots w_n$:

1. Copy w from tape 1 to tape 2.
2. Run M_1 on tape 1. If M_1 accepts, accept.
3. Run M_2 on tape 2. If M_2 accepts, accept.
4. Else, reject.



Issue: Might never run M_2 !

(Eg. if M_1 is in an infinite loop.)

Closure Properties

Fact: The Turing-recognizable languages are closed under the union operation:

If L_1 and L_2 are both Turing-recognizable, then $L_1 \cup L_2$ is also Turing-recognizable.

On input w :

1. Scan tapes 1,2,3 left-to-right to copy w to tapes 2 and 3.

2. Repeat forever:

a) Run M_1 for one step on tape 2.

b) Run M_2 for one step on tape 3.

c) If either machine accepts, **accept**.

3. If both machines reject, **reject**.

If $w \notin L_1 \cup L_2$, then neither machine accepts. ✓

If $w \in L_1 \cup L_2$, then either $w \in L_1$ or $w \in L_2$.

• if $w \in L_1$: M_1 accepts after finite no. of steps

• if $w \in L_2$: M_2 accepts after finite no. of steps ✓

Closure Properties

The Turing-decidable languages are closed under:

- Union
- Concatenation
- Star
- Intersection
- Reverse
- Complement

The Turing-recognizable languages are closed under:

- Union
- Concatenation
- Star
- Intersection
- Reverse