BU CS 332 – Theory of Computation

Lecture 12:

- TM Variants
- Decidable Languages

Reading:

Sipser Ch 3.2, 4.1

Mark Bun

March 4, 2020

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

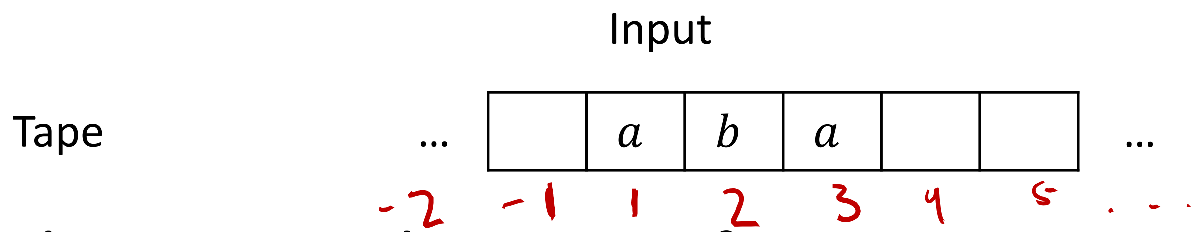
A is (Turing-)decidable if $A = L(M)$ for some TM M
which halts on every input

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

TM Variants

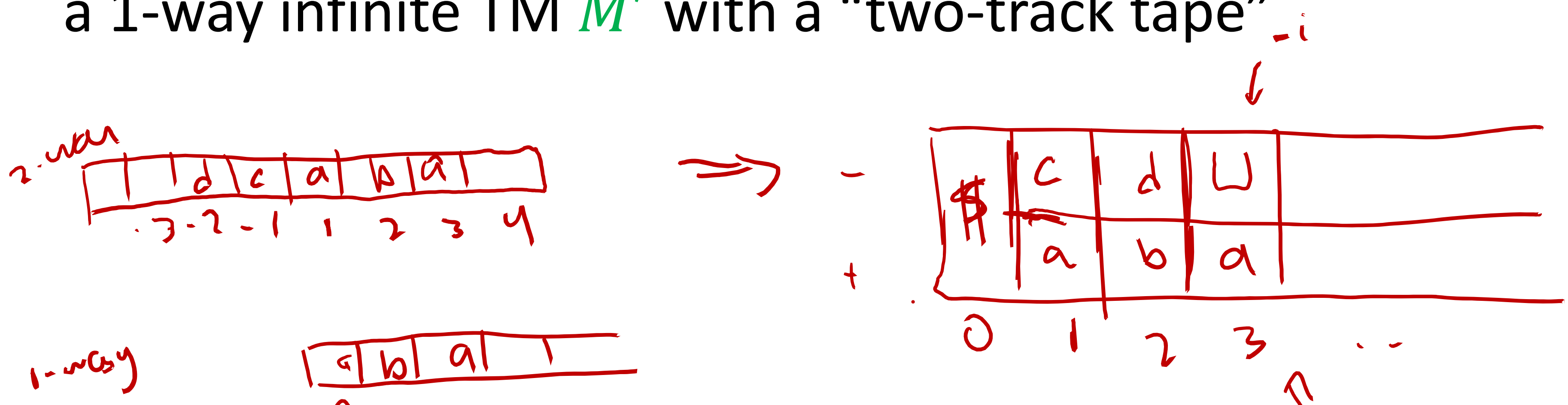
Extensions that do not increase the power of the TM model

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



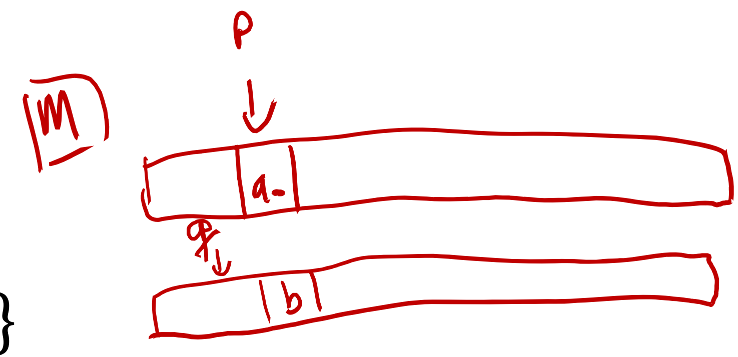
Proof that TMs with 2-way infinite tapes are no more powerful:

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



Formalizing the Simulation

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

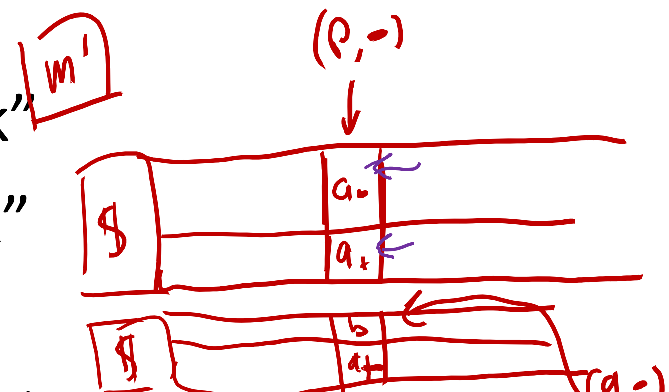


New tape alphabet: $\Gamma' = (\Gamma \times \Gamma) \cup \{\$\}$

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

TMs are equivalent to…

- TMs with "stay put"
- TMs with 2-way infinite tapes
- Multi-tape TMs — today
- Nondeterministic TMs
- Random access TMs ← "real computers"
- Enumerators — today
- Finite automata with access to an unbounded queue = 2-stack PDAs ← Homework problem
- Primitive recursive functions
- Cellular automata
- "Turing-complete" programming languages (C, Python, Java…)

…

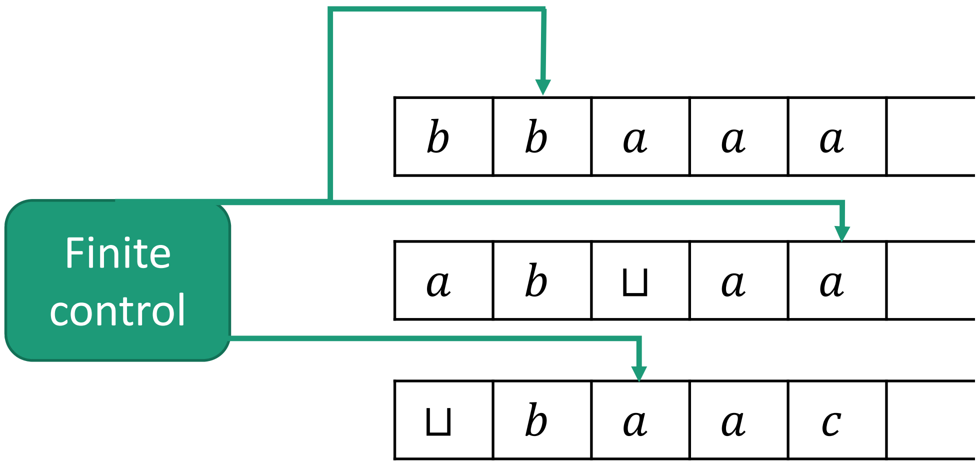
Church-Turing Thesis

The equivalence of these models is a mathematical theorem

Church-Turing *Thesis*: Each of these models captures our intuitive notion of algorithms

The Church-Turing Thesis is not a mathematical statement!

Multi-Tape TMs



Common Creation

read-only Input tape

"work" tape

Fixed number of tapes k (can't change during computation)

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

current state

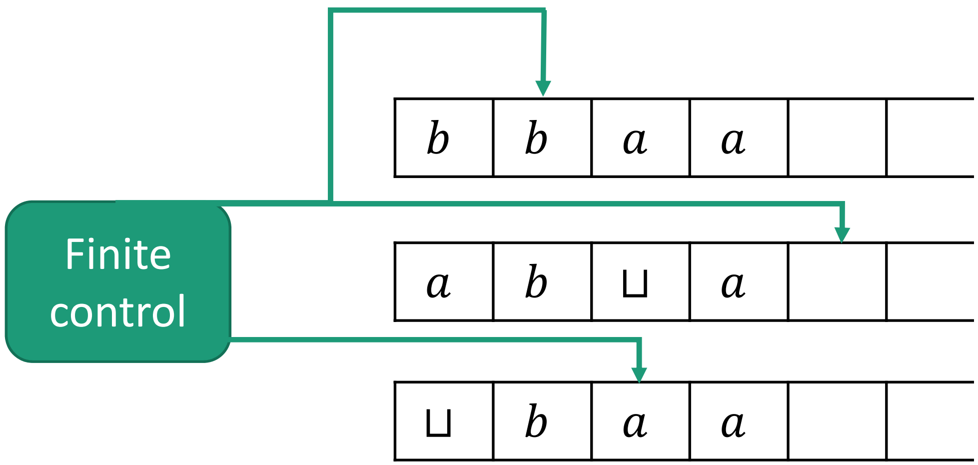
read one symbol per head

new state

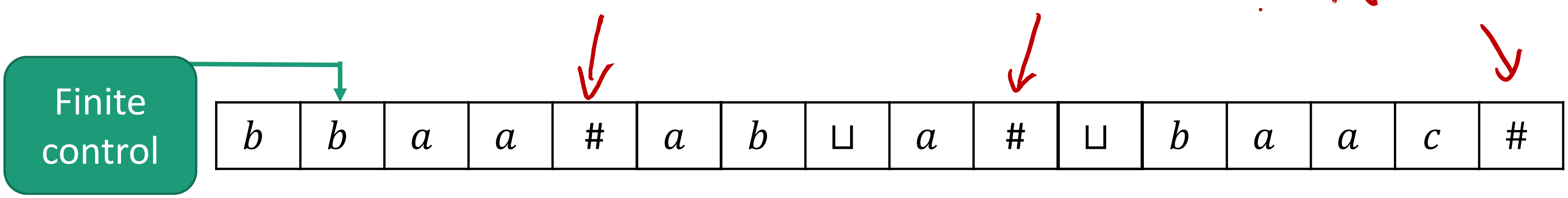
write one symbol per head

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'



#'s new symbol to mark boundaries between tapes



Simulating Multiple Tapes

$\dot{w}_1$	w_2	w_3	...	w_n	#	$\dot{\sqcup}$	#	..

Implementation-Level Description

On input $w = w_1 w_2 \ldots w_n$ (input to TM)

1. Format tape into $\# \, \dot{w}_1 w_2 \ldots w_n \# \, \dot{\sqcup} \, \# \, \dot{\sqcup} \, \# \ldots \#$
2. For each move of M:

first head points to w_1, second head to $\sqcup$ ...

Scan left-to-right, storing current symbols in finite control

Scan left-to-right, writing new symbols,

$\dot{c}$	w_2	w_3	..	w_n	#	a	#	..

Scan left-to-right, moving each tape head

c	$\dot{w}_2$	w_3	...	w_n	#	a	$\dot{\sqcup}$	#

If a tape head goes off the right end, insert blank

If a tape head goes off left end, move back right

Why are Multi-Tape TMs Helpful?

To show a language is Turing-recognizable or decidable, suffices to construct a multi-tape TM

Very helpful for proving closure properties

Ex. Closure of recognizable languages under union. Suppose M_1 is a single-tape TM recognizing L_1, M_2 is a single-tape TM recognizing L_2

Proof via Implementation-level Description:
construct 2-tape TM recognizing $L_1 \cup L_2$

| w_1 | ... | w_n |

On input w:
1. Scan Tape 1 left-to-right, copying characters onto Tape 2
2. Move both heads to left end of tapes
3. Repeatedly:
   Simulate 1 step of M_1 on tape 1
   Simulate 1 step of M_2 on tape 2
   If either machine accepts, accept

Nondeterministic TMs

At any point in computation, may nondeterministically branch. Accepts iff there exists an accepting branch.

Transition function $\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R, S\})$

Ex. NTM for $\{w \mid w$ is a binary number representing the product of two positive integers $a, b\}$

High-level ≥ 2

1. Non-deterministically "guess" $a, b \leq w$
2. compute $a \times b$, check if equal to w, accept if so.
3. Reject otherwise

Nondeterministic TMs

Theorem: Every nondeterministic TM has an equivalent deterministic TM

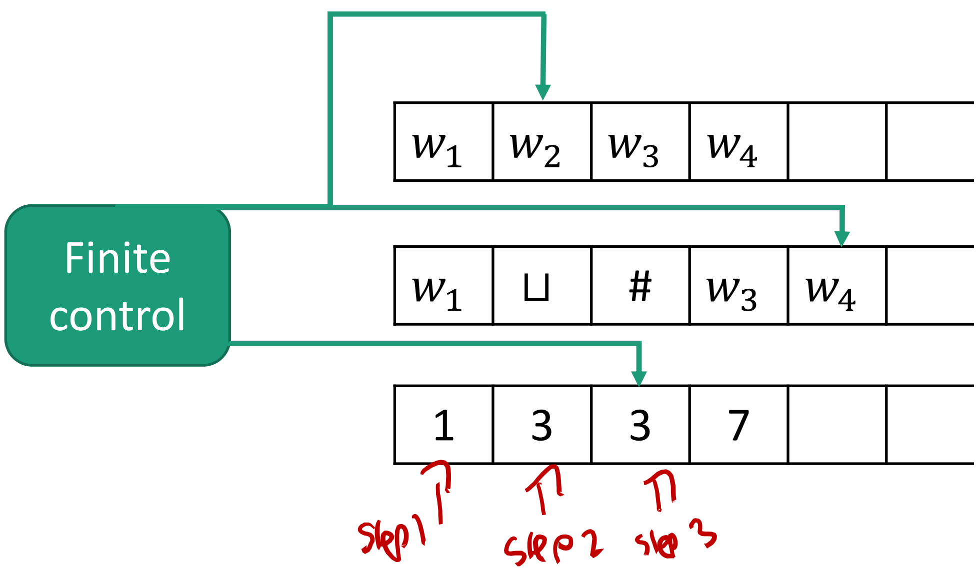
Proof idea:

Systematically try all 1-step computations, all 2-step computations, … and see if one of them accepts

Nondeterministic TMs

Theorem: Every nondeterministic TM has an equivalent deterministic TM

Proof idea: Simulate an NTM N using a 3-tape TM



Input w to N (read-only)

Simulation tape (run N on w using nondeterministic choices from tape 3)

Address in computation tree

Deterministic

Addresses

First step: 1, 2, ..., B

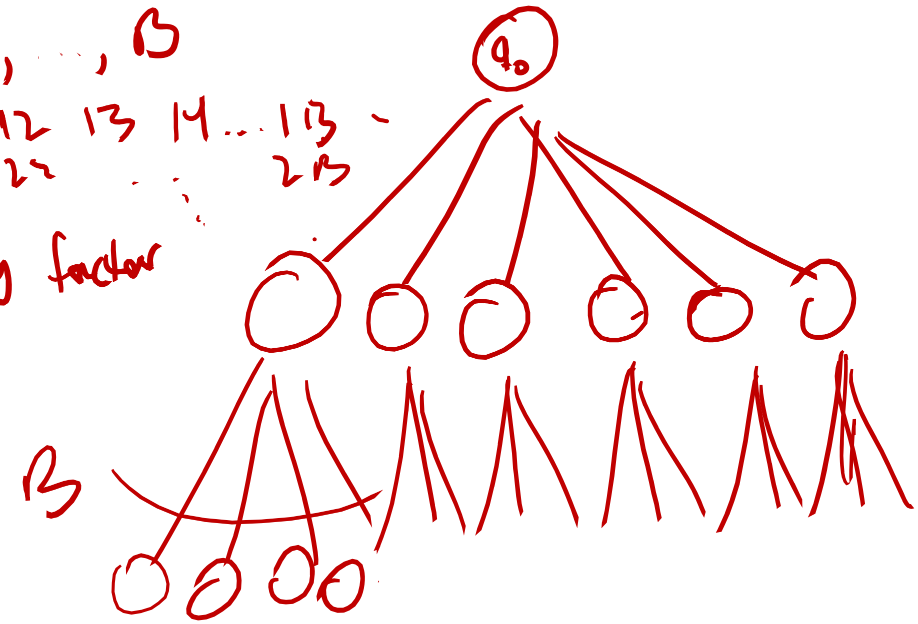
Second step: 11 12 13 14 ... 1B

21 22 ... 2B

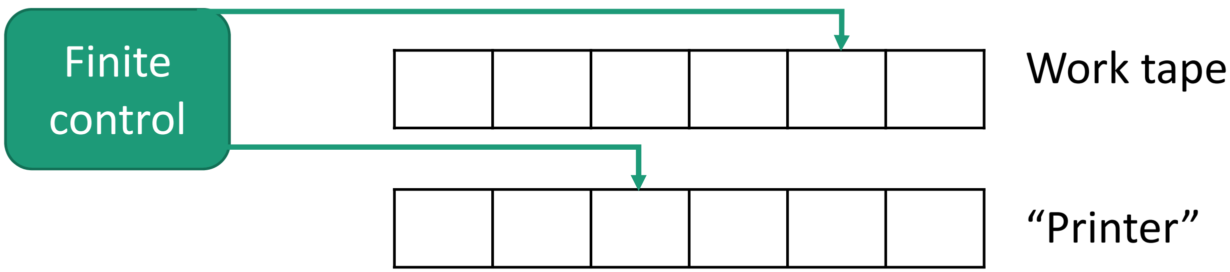
Branching factor

B

Nondeterministic



Enumerators



- Starts with two blank tapes
- Prints strings to printer

$L(E) = \{$strings eventually printed by $E\}$

- May never terminate (even if language is finite)
- May print the same string many times

Enumerable = Turing-Recognizable

Theorem: A language is Turing-recognizable $\Leftrightarrow$ some enumerator enumerates it

$\Leftarrow$ Start with an enumerator E for A and give a TM

Enumerable = Turing-Recognizable

Theorem: A language is Turing-recognizable ⇔ some enumerator enumerates it

⇒ Start with a TM M for A and give an enumerator

Mid-Semester Feedback Form

https://forms.gle/LTBEY1BoSZh8nupV6