BU CS 332 – Theory of Computation

https://forms.gle/YvsPJu2wGxUegL6fA

Lecture 11:

- TM Variants
- Nondeterministic TMs
- Church-Turing Thesis

Reading:

Sipser Ch 3.2

Mark Bun

October 18, 2022

Last Time

Formal definition of a TM, configurations, how a TM computes

Recognizability vs. Decidability:

A is Turing-recognizable if there exists a TM M such that

- $w \in A \implies M$ 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 there exists a TM M such that

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

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
- Cellular automata

…

Equivalent TM models

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

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

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

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

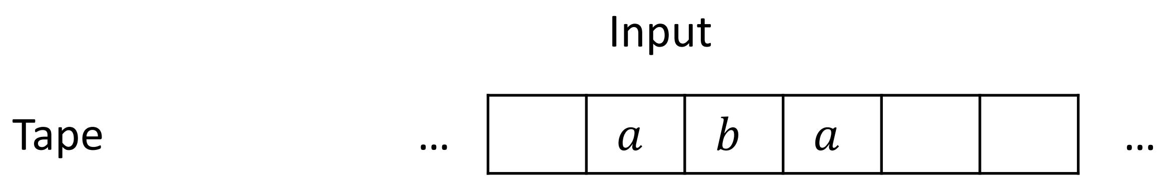
Proof that TMs with stay put are no more powerful:

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

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 $w = w_1 w_2 \dots w_n$:

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 "q, working on upper track"

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

New transitions:

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

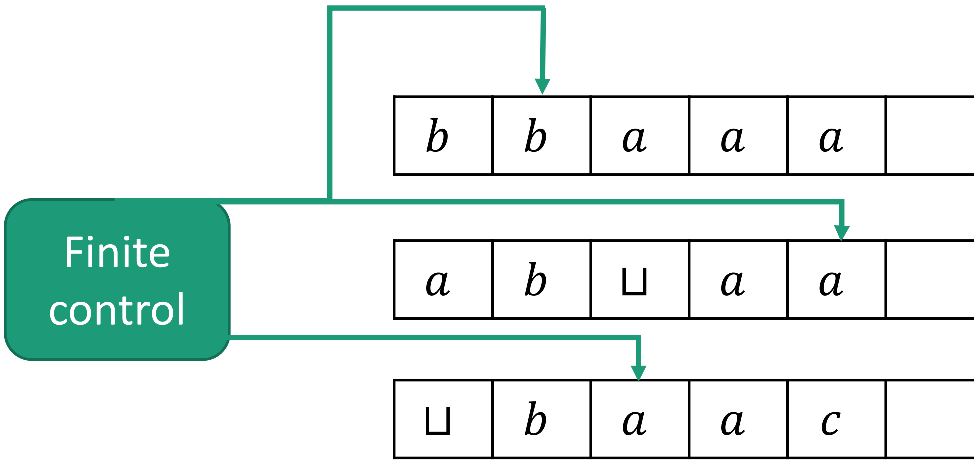
Also need new transitions for moving right, lower track, hitting $\$$, initializing input into 2-track format

TMs are equivalent to…

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

…

Multi-Tape TMs



Fixed number of tapes k

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

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

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) 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 left ends of tapes 1 and 2, scan both tapes 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

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

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

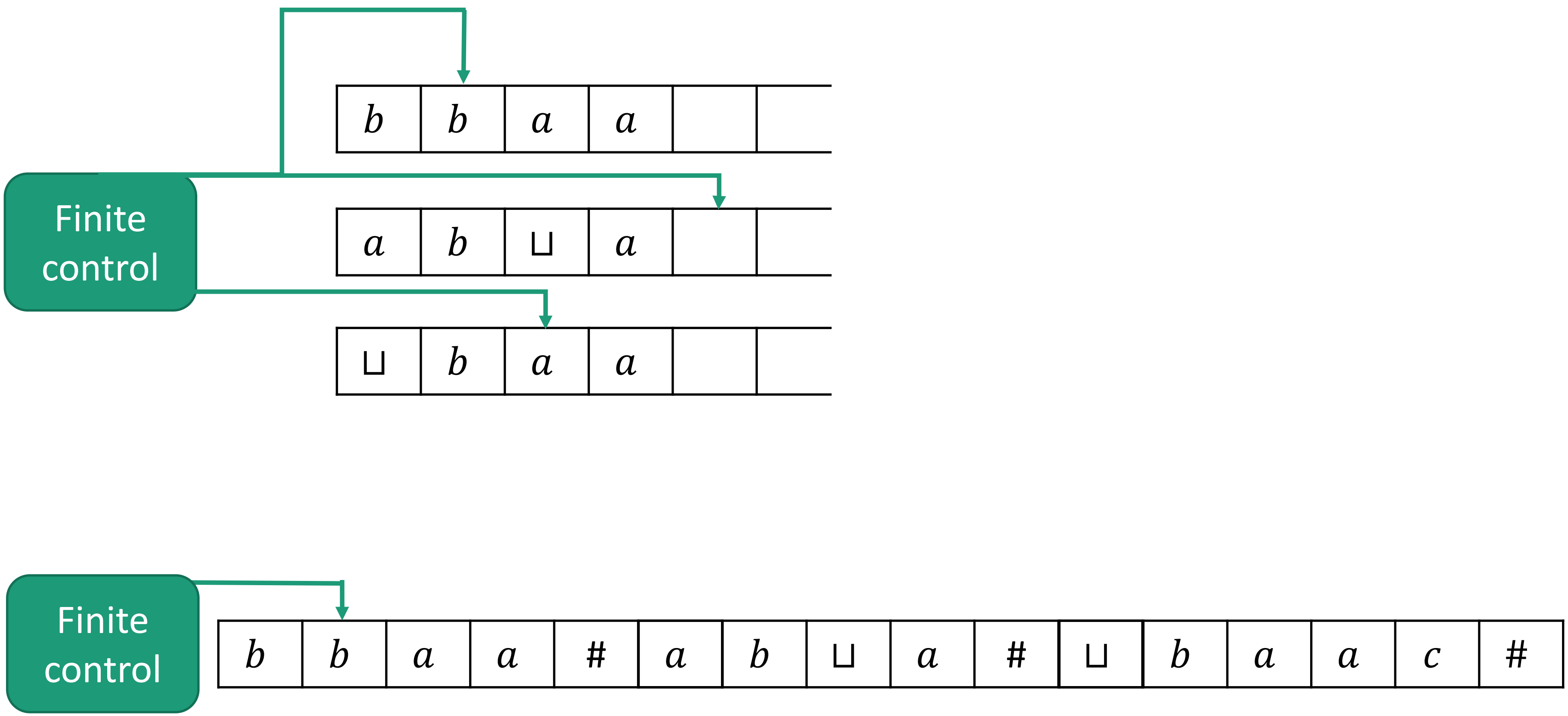
Ex. Closure of recognizable languages under union. Suppose M_1 is a single-tape TM recognizing L_1, M_2 is a single-tape TM recognizing L_2

On input w:

1) Scan tapes 1, 2, and 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

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'



How to Simulate It

To show that a TM variant is no more powerful than the basic, single-tape TM:

Show that if M is any variant machine, there exists a basic, single-tape TM M' that can simulate M

(Usual) parts of the simulation:

- Describe how to initialize the tapes of M' based on the input to M
- Describe how to simulate one step of M's computation using (possibly many steps of) M'

Simulating Multiple Tapes

Implementation-Level Description of M'

On input $w = w_1 w_2 \ldots w_n$

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

    Scan left-to-right, finding current symbols

    Scan left-to-right, writing new symbols,

    Scan left-to-right, moving each tape head

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

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

Closure Properties

The Turing-decidable languages are closed under:

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

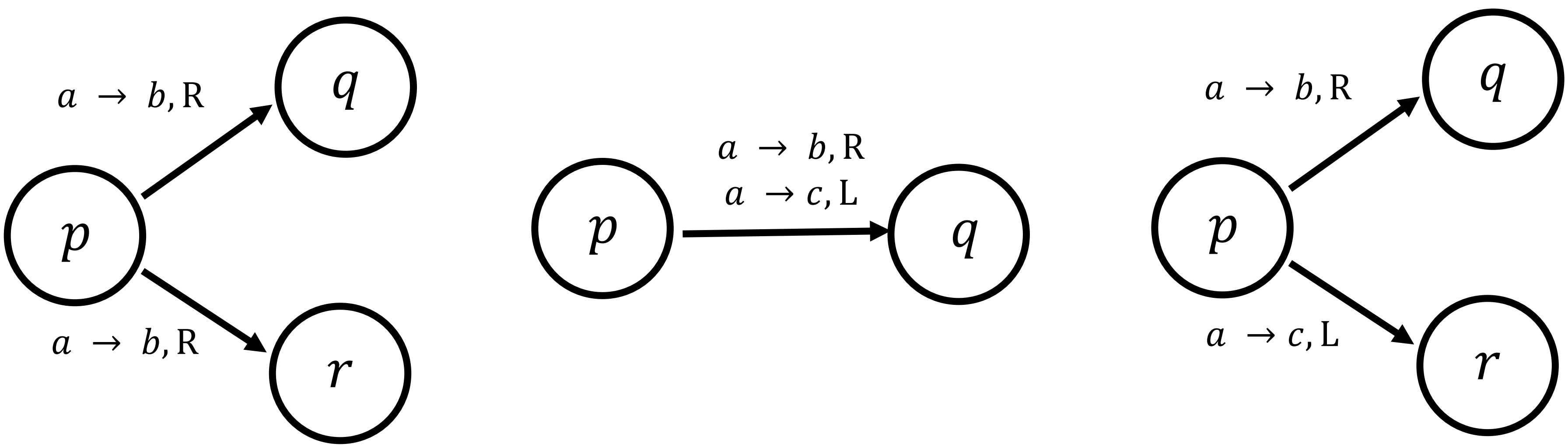
The Turing-recognizable languages are closed under:

- Union
- Concatenation
- Star
- Intersection
- Reverse

Nondeterministic TMs

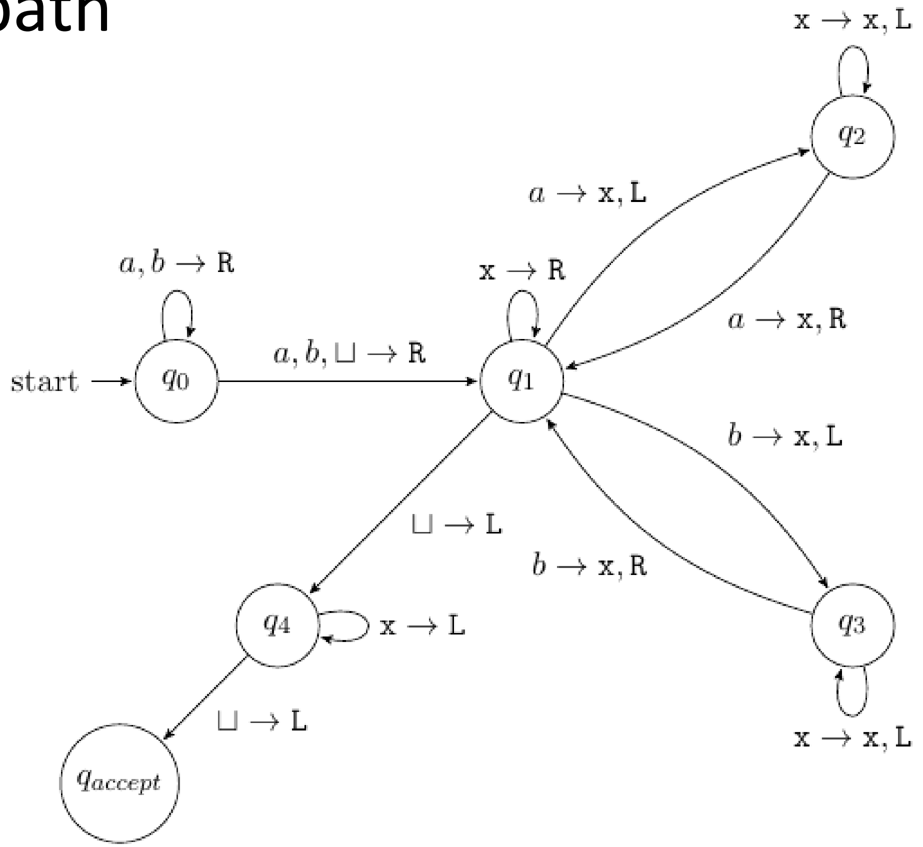
At any point in computation, may nondeterministically branch. Accepts iff there exists an accepting branch.

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



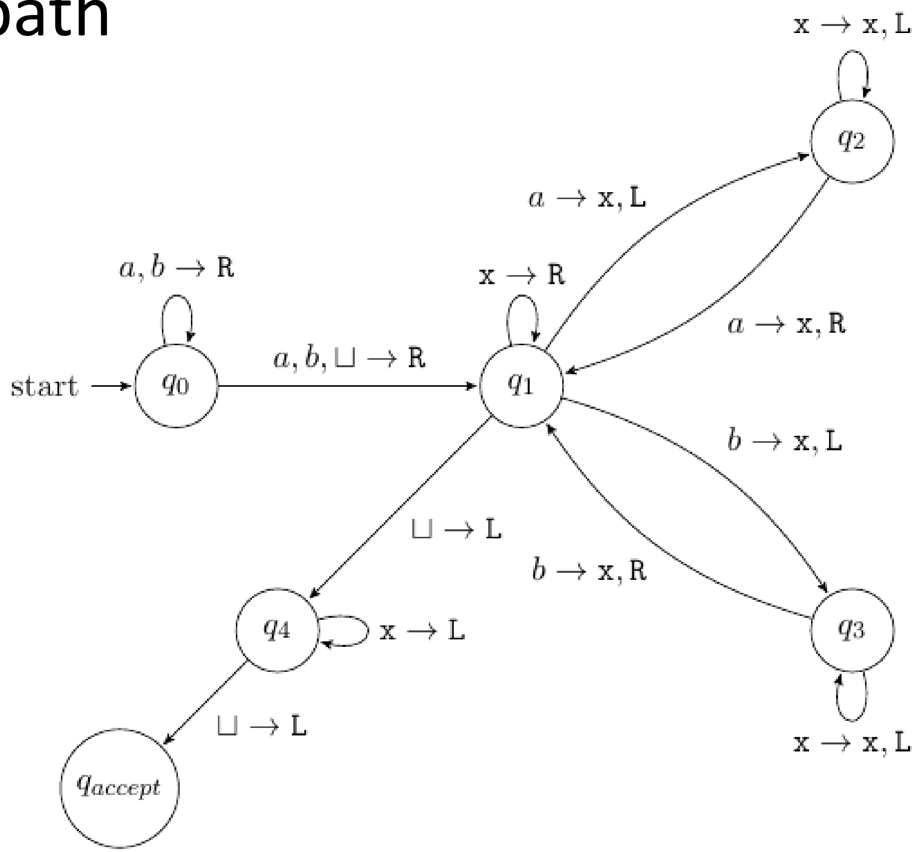
Nondeterministic TMs

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



Nondeterministic TMs

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



What is the language recognized by this NTM?

a) $\{ ww \mid w \in \{a, b\}^* \}$
b) $\{ ww^R \mid w \in \{a, b\}^* \}$
c) $\{ ww \mid w \in \{a, b, \mathrm{x}\}^* \}$
d) $\{ w\mathrm{x}^n w^R \mid w \in \{a, b\}^*, n \geq 0 \}$

Nondeterministic TMs

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

Implementation-Level Description

On input string w:

1) Scan tape left-to-right. At some point, nondeterministically go to step 2
2) a) Read the next symbol s and cross it off
   b) Move the head left repeatedly until a non-X symbols is found. If it matches s, cross it off. Else, reject.
   c) Move the head right until a non-X symbol is found. If blank is hit, go to step 3.
   d) Go back to 2a)
3) Check that the entire tape consists of X's. If so, accept. Else, reject.

Nondeterministic TMs

Ex. Given TMs M_1 and M_2, construct an NTM recognizing $L(M_1) \cup L(M_2)$

Nondeterministic TMs

Ex. NTM for $L = \{w \mid w$ is a binary number representing the product of two integers $a, b \geq 2\}$

<u>High-Level Description:</u>

Nondeterministic TMs

An NTM N accepts input w if when run on w it accepts on at least one computational branch

$$L(N) = \{w \mid N \text{ accepts input } w\}$$

An NTM N is a decider if on **every** input, it halts on **every** computational branch

Nondeterministic TMs

Theorem: Every nondeterministic TM can be simulated by an equivalent deterministic TM

Proof idea: Explore "tree of possible computations"

Simulating NTMs

Which of the following algorithms is always appropriate for searching the tree of possible computations for an accepting configuration?

a) Depth-first search: Explore as far as possible down each branch before backtracking

b) Breadth-first search: Explore all configurations at depth 1, then all configurations at depth 2, etc.

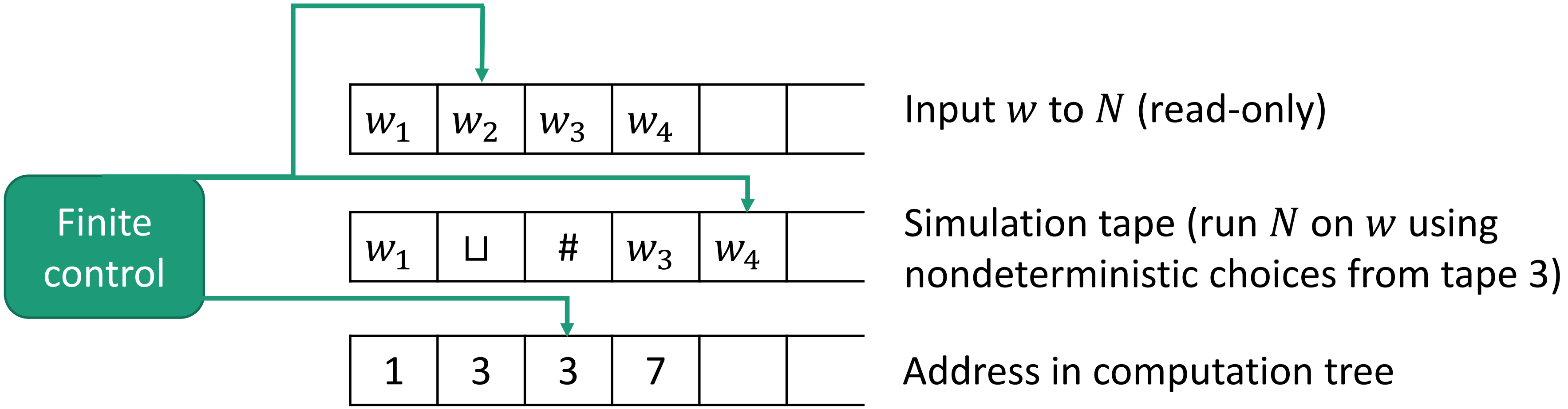
c) Both algorithms will always work

Nondeterministic TMs

Theorem: Every nondeterministic TM has an equivalent deterministic TM

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

(See Sipser for full description)



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

…

Church-Turing Thesis

The equivalence of these models is a mathematical theorem (you can prove that each can simulate another)

Church-Turing Thesis v1: The basic TM (hence all of these models) captures our intuitive notion of algorithms

Church-Turing Thesis v2: Any physically realizable model of computation can be simulated by the basic TM

The Church-Turing Thesis is not a mathematical statement! Can't be mathematically proved