BU CS 332 – Theory of Computation

Lecture 11:

- TM Variants and Closure Properties
- Church-Turing Thesis

Reading:

Sipser Ch 3.2

Mark Bun

March 1, 2021

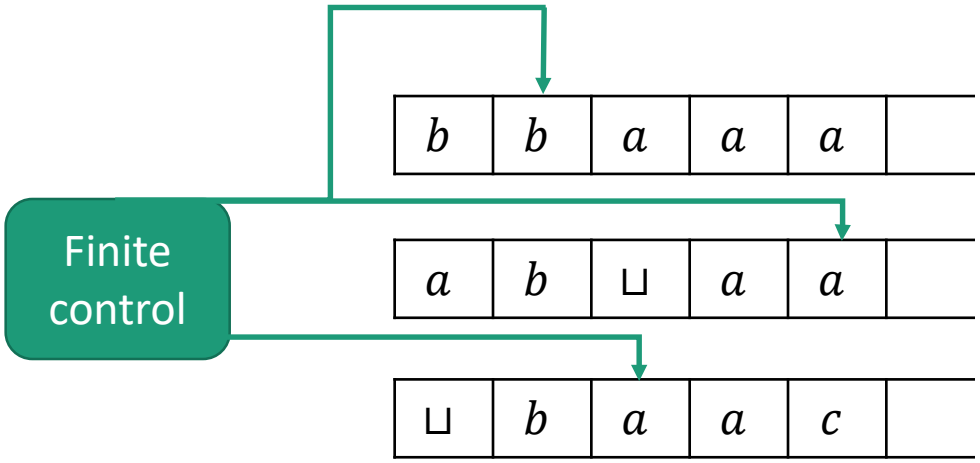
TM Variants

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 (can't change during computation)

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

How to Simulate It

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

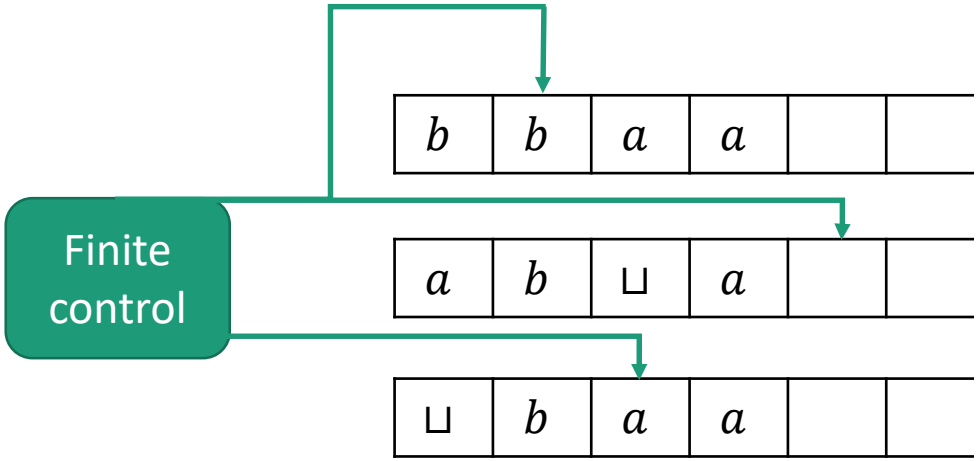
Show that if M is any variant machine, there exists a basic, single-tape TM M' that can simulate M

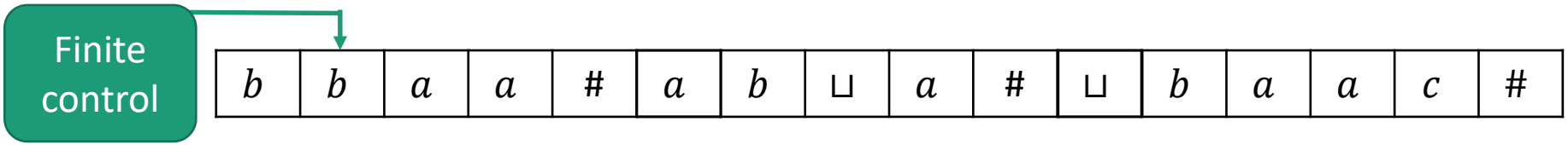
(Usual) parts of the simulation:

- 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'

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'





Simulating Multiple Tapes

Implementation-Level Description of M'

On input $w = w_1 w_2 \ldots w_n$

1. Format tape into $\# \dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$
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

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

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

What's wrong with this construction?

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

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.

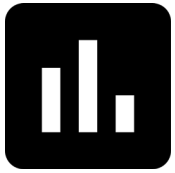
Transition function $\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R, S\})$

Nondeterministic TMs

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

Nondeterministic TMs

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



What is the language of this NTM?

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

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

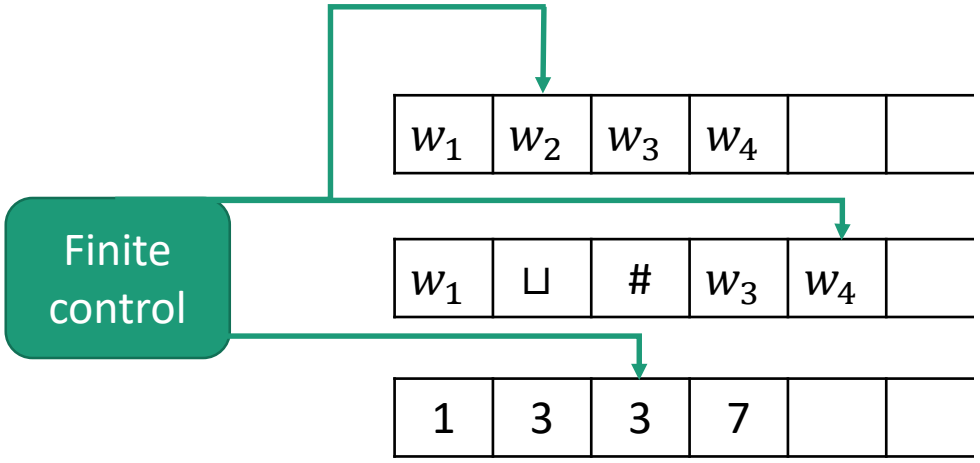
Proof idea:

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

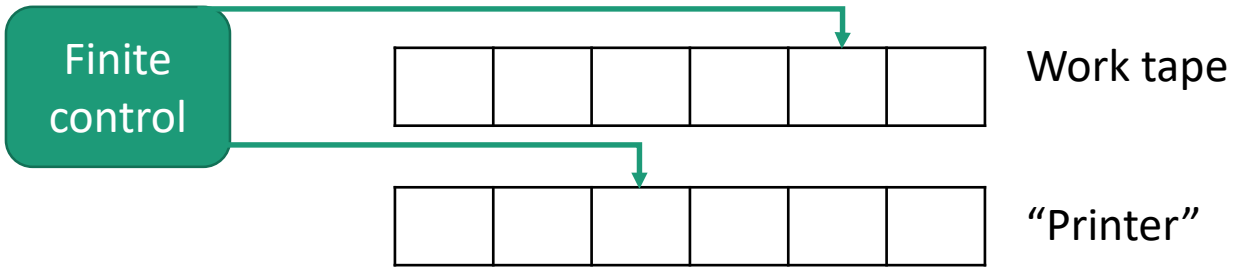


Input w to N (read-only)

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

Address in computation tree

Enumerators



- Starts with two blank tapes
- Prints strings to printer

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

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

Enumerator Example

What is the language enumerated by the following program?

"Write 1 to the work tape. Repeat the following forever: Multiply the binary number on the work tape by 2, and copy the result to the printer."

a) $\{x \mid x \text{ is a binary number } n^2 \text{ for some } n \geq 1\}$
b) $\{x \mid x \text{ is a binary number } 2^n \text{ for some } n \geq 0\}$
c) $\{x \mid x \text{ is a binary number } 2^{2^n} \text{ for some } n \geq 0\}$
d) None of the above

Enumerable = Turing-Recognizable

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

⇐ 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

Enumerable = Turing-Recognizable

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

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

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

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!