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

Input

Tape	…		a	b	a			…

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
- Nondeterministic TMs
- Random access TMs
- Enumerators
- Finite automata with access to an unbounded queue = 2-stack PDAs
- Primitive recursive functions
- Cellular automata
- “Turing-complete” programming languages (C, Python, Java…)

…

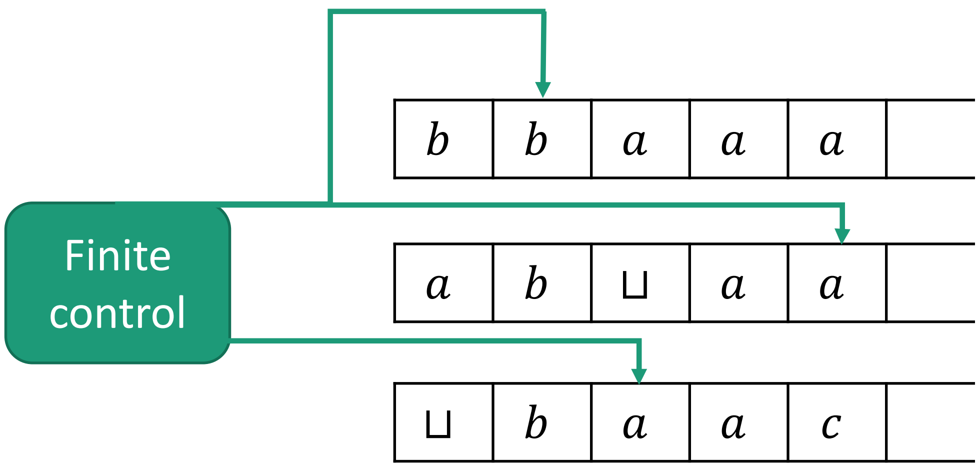
Church-Turing Thesis

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

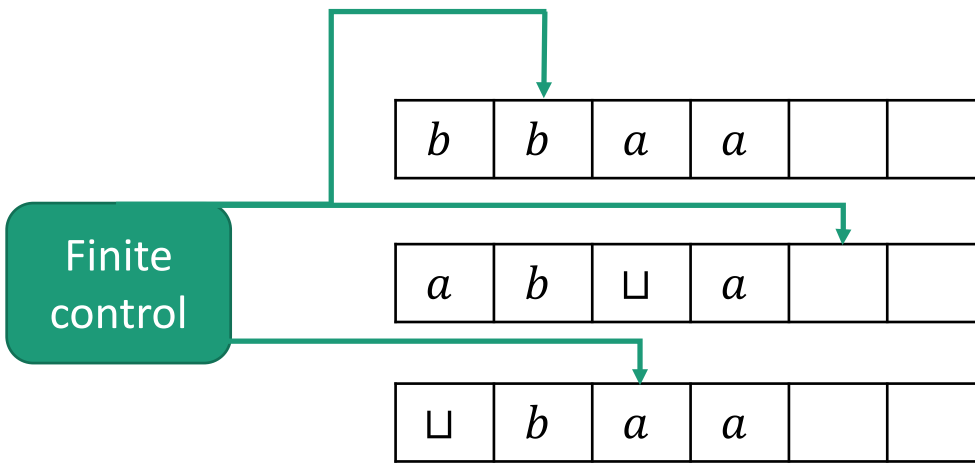


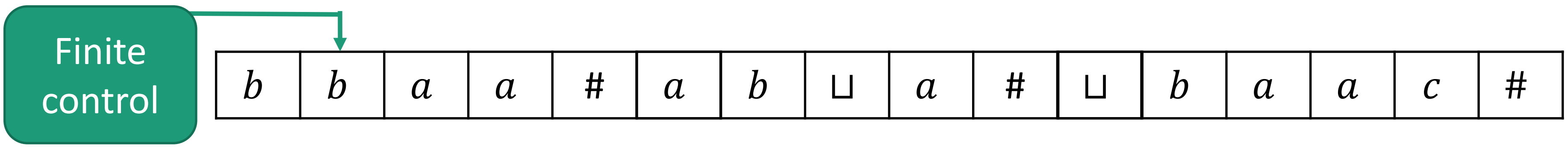
Fixed number of tapes k (can't change during computation)

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

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

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, storing current symbols in finite control

    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, 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

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 $a, b\}$

Nondeterministic TMs

Theorem: Every nondeterministic TM has an equivalent deterministic TM

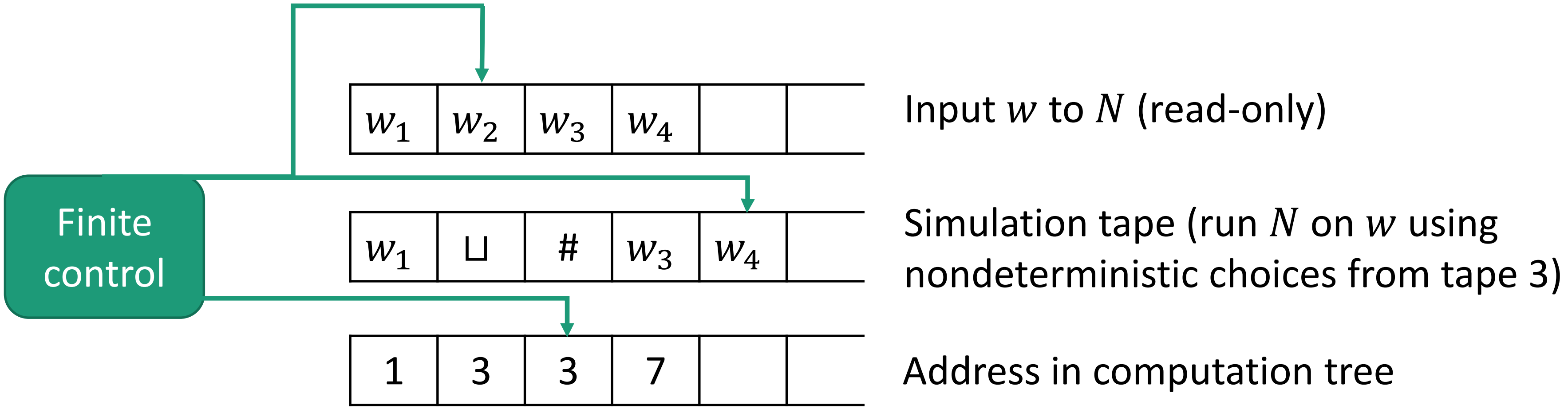
Proof idea:

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

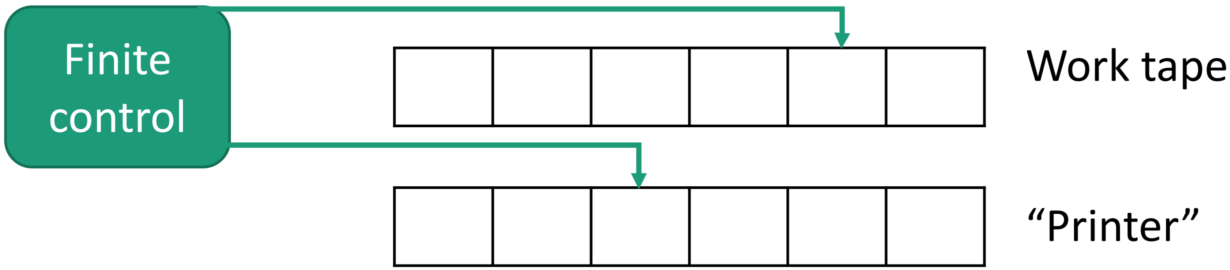
Nondeterministic TMs

Theorem: Every nondeterministic TM has an equivalent deterministic TM

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



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 $\Leftrightarrow$ some enumerator enumerates it

$\Rightarrow$ Start with a TM M for A and give an enumerator

Decidable Languages

1928 – The *Entscheidungsproblem*

The "Decision Problem"

Is there an algorithm which takes as input a formula (in first-order logic) and decides whether it's logically valid?

Questions about regular languages

Design a TM which takes as input a DFA D and a string w, and determines whether D accepts w

How should the input to this TM be represented?

Let $D = (Q, \Sigma, \delta, q_0, F)$. List each component of the tuple separated by ;

- Represent Q by ,-separated binary strings
- Represent Σ by ,-separated binary strings
- Represent $\delta : Q \times \Sigma \ \rightarrow Q$ by a ,-separated list of triples (p, a, q), …

Denote the encoding of D, w by $\langle D, w \rangle$

Representation independence

Computability (i.e., decidability and recognizability) is not affected by the choice of encoding

Why? A TM can always convert between different encodings

For now, we can take $\langle \quad \rangle$ to mean "any reasonable encoding"

A “universal” algorithm for recognizing regular languages

$A_{\text{DFA}} = \{\langle D, w \rangle \mid \text{DFA } D \text{ accepts } w\}$

Theorem: A_{DFA} is decidable

Proof sketch: Define a TM M which on input $\langle D, w \rangle$:

1. Check if $\langle D, w \rangle$ is a valid encoding (reject if not)
2. Simulate D on w, i.e.,
   - Tape 2: Maintain w and head location of D
   - Tape 3: Maintain state of D, update according to δ
3. Accept iff D ends in an accept state

Other decidable languages

$A_{\text{DFA}} = \{\langle D, w\rangle \mid \text{DFA } D \text{ accepts } w\}$

$A_{\text{NFA}} = \{\langle N, w\rangle \mid \text{NFA } N \text{ accepts } w\}$

$A_{\text{CFG}} = \{\langle G, w\rangle \mid \text{CFG } G \text{ generates } w\}$

CFG Generation

Theorem: $A_{\mathrm{CFG}} = \{\langle G, w\rangle \mid \mathrm{CFG}\ G \text{ generates } w\}$ is Turing-recognizable

Proof idea: Define a TM M recognizing A_{CFG}

On input $\langle G, w\rangle$

1. Enumerate all strings that can be generated from G

   (i.e., all length-1 derivations, all length-2 derivations, …)

2. If any of these strings equal w, accept

CFG Generation

Theorem: $A_{\mathrm{CFG}} = \{\langle G, w\rangle \mid \mathrm{CFG}\ G \text{ generates } w\}$ is decidable

Chomsky Normal Form for CFGs:

- Can have a rule $S \to \varepsilon$
- All remaining rules of the form $A \to BC$ or $A \to a$
- Cannot have S on the RHS of any rule

Lemma: Any CFG can be converted into an equivalent CFG in Chomsky Normal Form

Lemma: If G is in Chomsky Normal Form, any nonempty string w that can be derived from G has a derivation with at most $2|w| - 1$ steps

CFG Generation

Theorem: $A_{\mathrm{CFG}} = \{\langle G, w\rangle \mid \mathrm{CFG}\ G \text{ generates } w\}$ is decidable

Proof idea: Define a TM M recognizing A_{CFG}

On input $\langle G, w\rangle$

1. Convert G into Chomsky Normal Form
2. Enumerate all strings derivable in $\leq 2|w| - 1$ steps
3. If any of these strings equal w, accept

Mid-Semester Feedback Form

https://forms.gle/LTBEY1BoSZh8nupV6