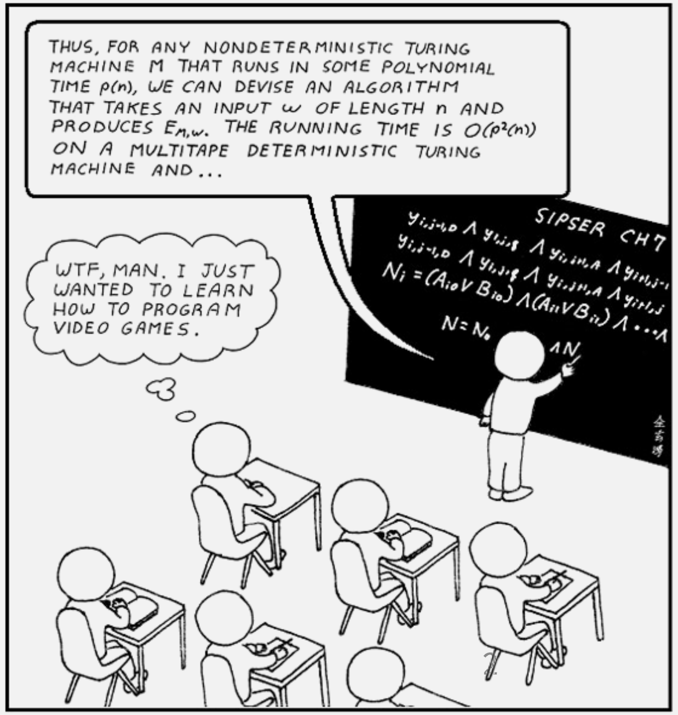
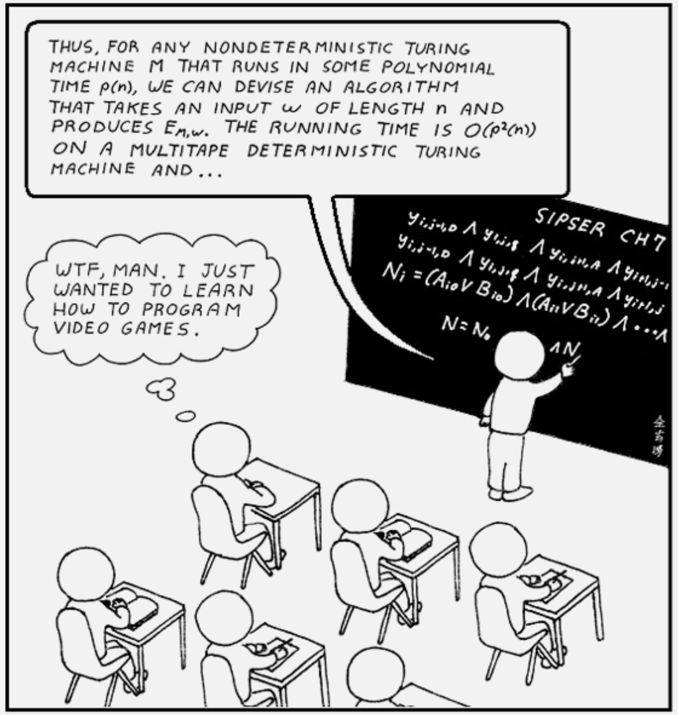
Turing Machine Variants

Monday March 22, 2021



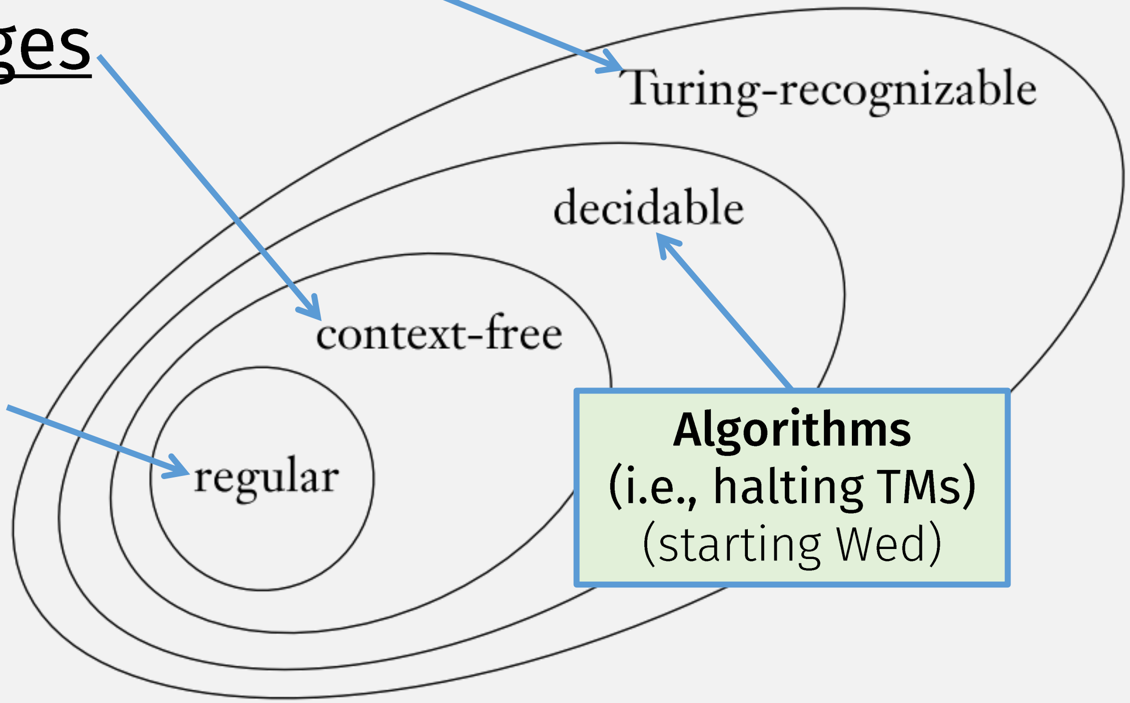
Welcome Back!

- HW 6 due Sun 3/28 11:59pm EST
  - Ch 2-3 material
- HW 7 out
  - Ch 4 material (which we start on Wed)
  - Due Sun 4/4 11:59pm EST



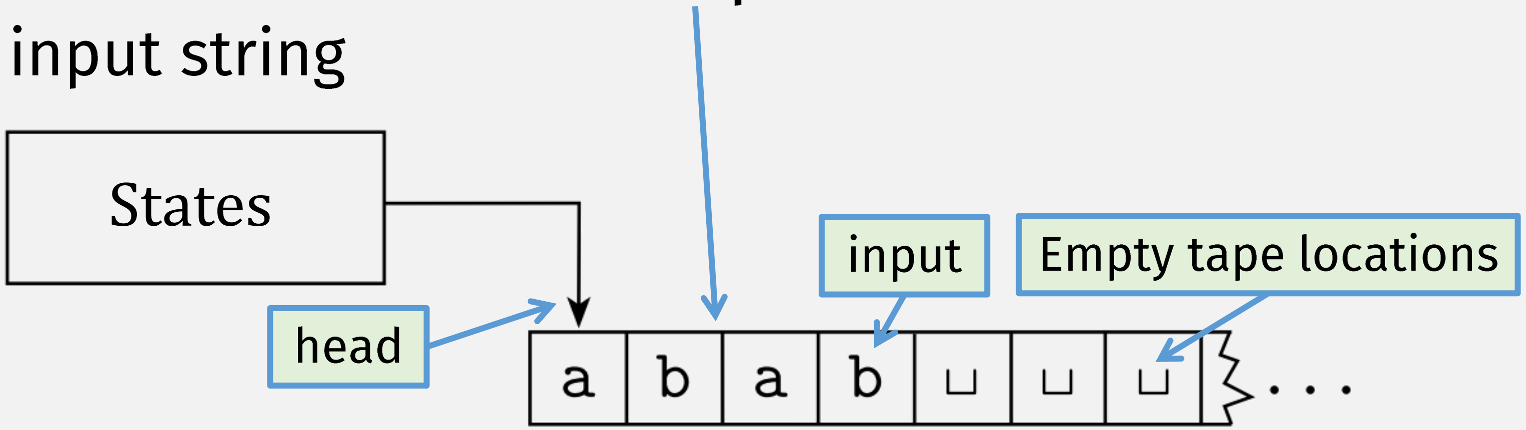
CS420, So Far

- **Turing Machines** (TMs)
  - Infinite tape (memory), arbitrary read/write
  - Expresses any "computation"
- **PDAs**: recognize context-free languages
  - Infinite stack (memory), push/pop only
  - Can't express arbitrary dependency,
    - e.g., $\{ww \mid w \in \{0,1\}^*\}$
- **DFAs / NFAs**: recognize regular langs
  - Finite states (memory)
  - Can't express dependency
    e.g., $\{0^n 1^n \mid n \geq 0\}$



Last Time: Turing Machines

- Turing Machines can read and write to a "tape"
  - Tape initially contains input string
- The tape is infinite



- Each step: "head" can move left or right
- A Turing Machine can accept/reject at any time

Last time: Informal vs Formal Description

M_1 accepts input in language: $B = \{w\#w \mid w \in \{0,1\}^*\}$

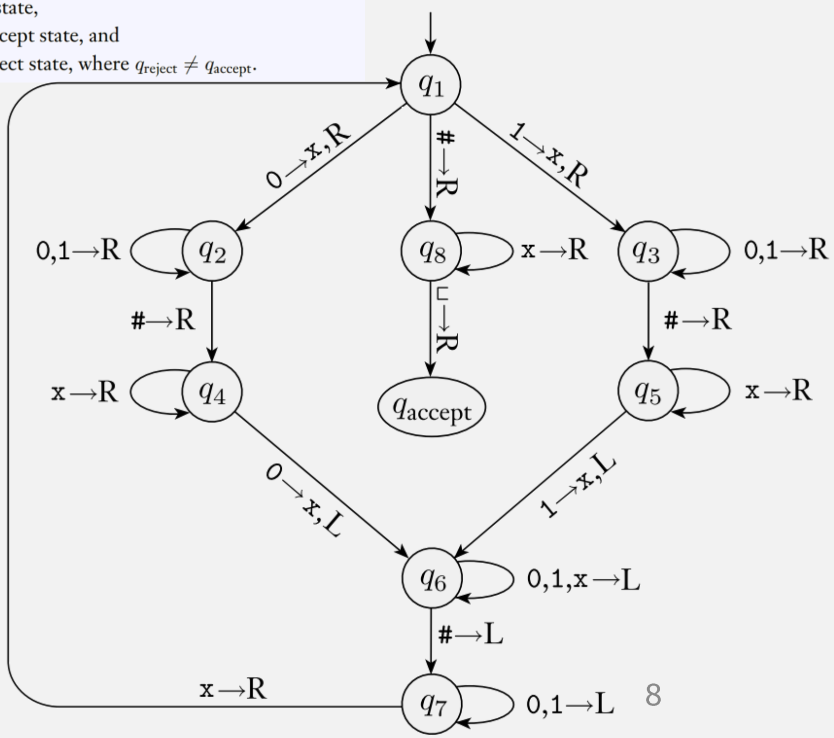
M_1 = "On input string w:

1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
2. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise, *accept*."

DEFINITION 3.3

A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.



Last time: Non-Halting Machines

IMPORTANT SLIDE

- A DFA, NFA, or PDA always halts
  - Because the (finite) input is read in order, exactly once
- A Turing Machine can run forever
  - E.g., the head can move back and forth in a loop
- Thus, there are two classes of Turing Machines:
  - A recognizer is a Turing Machine that may run forever
  - A decider is a Turing Machine that always halts

DEFINITION 3.5

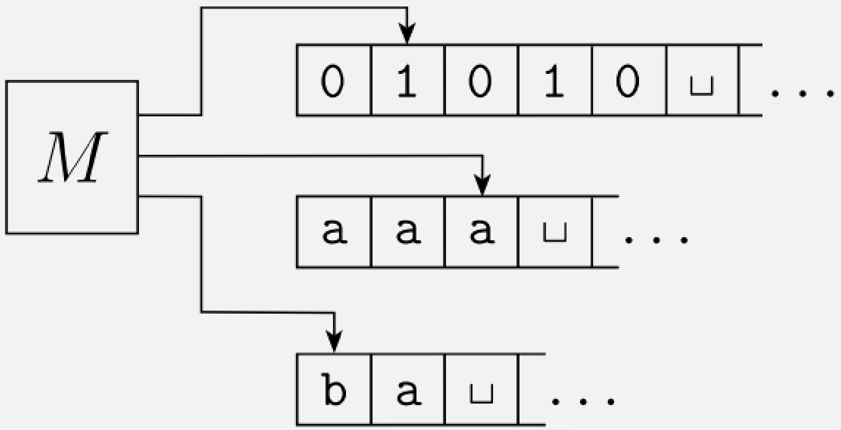
Call a language ***Turing-recognizable*** if some Turing machine recognizes it.

DEFINITION 3.6

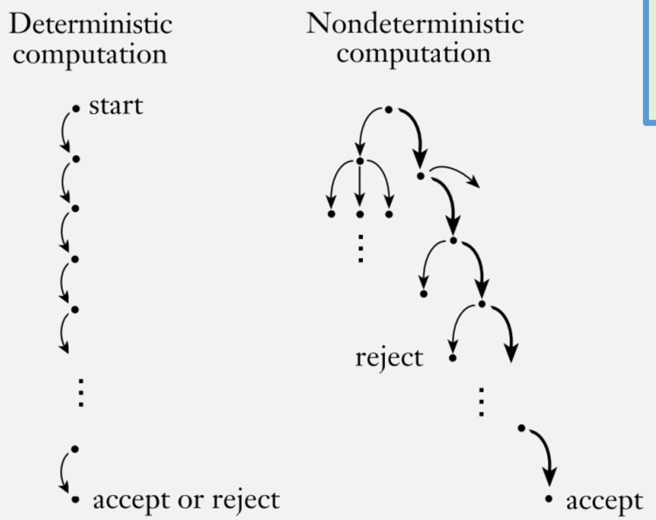
Call a language ***Turing-decidable*** or simply ***decidable*** if some Turing machine decides it.

Today:

1. **Multi-tape TMs**



2. **Non-deterministic TMs**



We will prove that all TM variations are **equivalent**

Reminder: Equivalence of Machines

- Two machines are equivalent when …
- … they recognize the same language

Theorem: Single-tape TM ⇔ Multi-tape TM

=> **If** a single-tape TM recognizes a language,
then a multi-tape TM recognizes the language

- A single-tape TM is equivalent to …
- … a multi-tape TM that only uses one of its tapes
- **DONE!**

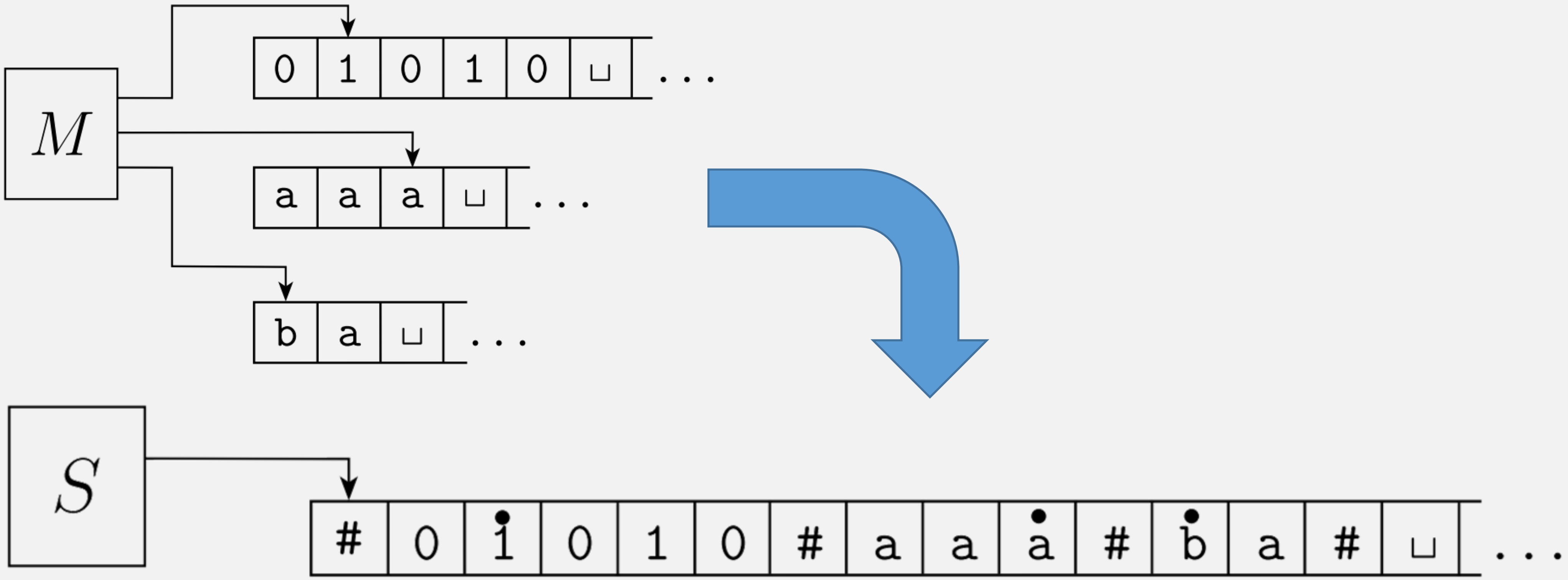
<= **If** a multi-tape TM recognizes a language,
then a single-tape TM recognizes the language

- Convert multi-tape TM to single-tape TM

Multi-tape TM ➔ Single-tape TM

Idea: Use delimiter (#) on single-tape to simulate multiple tapes

- Add "dotted" version of every char to simulate multiple heads



Theorem: Single-tape TM ⇔ Multi-tape TM

=> **If** a single-tape TM recognizes a language,
then a multi-tape TM recognizes the language

- A single-tape TM is equivalent to …
- … a multi-tape TM that only uses one of its tapes
- **DONE!**

<= **If** a multi-tape TM recognizes a language,
then a single-tape TM recognizes the language

- Convert multi-tape TM to single-tape TM
- **DONE!** ∎

Non-Deterministic Turing Machines

Flashback: DFAs vs NFAs

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta\colon Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

vs

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta\colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Last Time: Turing Machine formal def

DEFINITION 3.3

A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Non-deterministic: Turing Machine formal def

DEFINITION 3.3

A **Nondeterministic Turing Machine** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. ~~$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$~~ → $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Thm: Deterministic TM ⇔ Non-det. TM

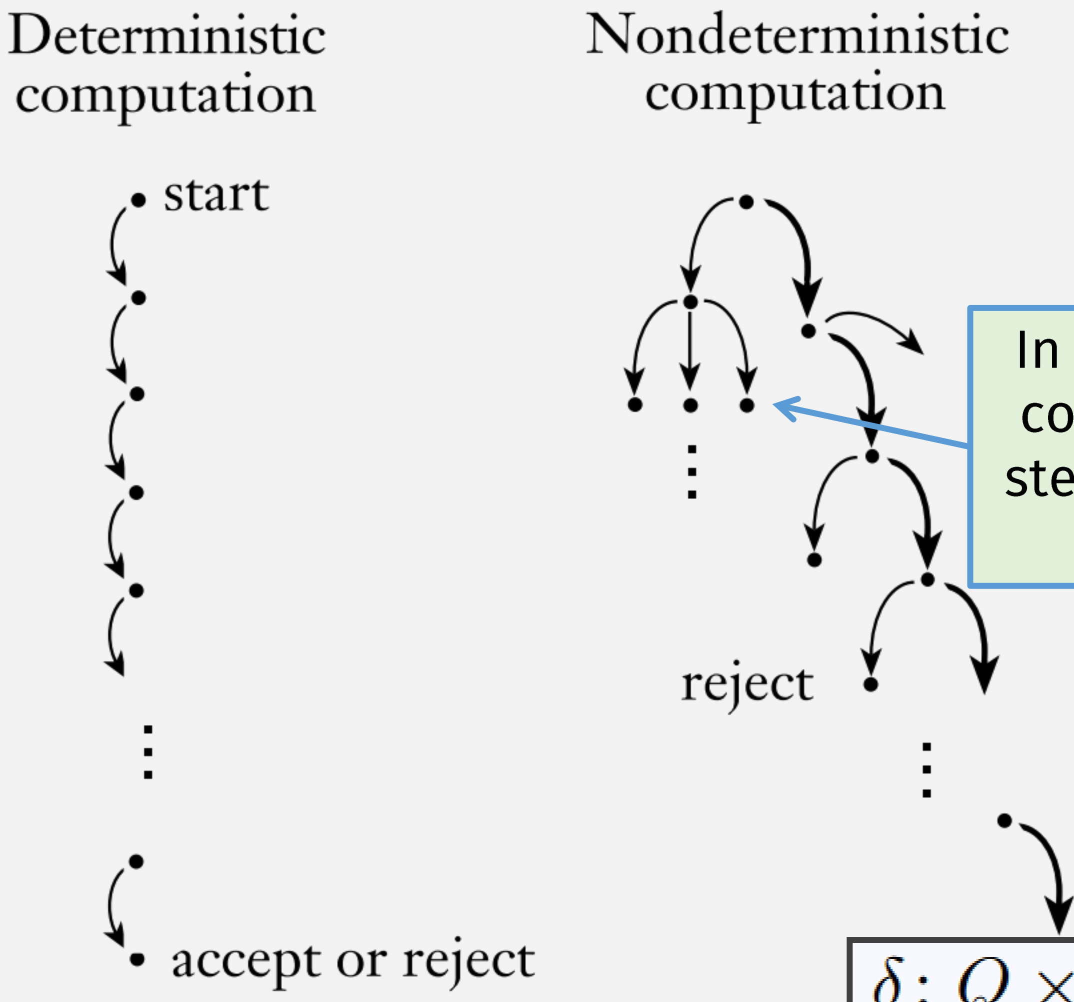
=> **If** a deterministic TM recognizes a language,
then a nondeterministic TM recognizes the language

- To convert Deterministic TM → Non-deterministic TM …
- … change Deterministic TM delta fn output to a one-element set
  - (just like conversion of DFA to NFA)
- **DONE!**

<= **If** a nondeterministic TM recognizes a language,
then a deterministic TM recognizes the language

- To convert Non-deterministic TM → Deterministic TM …
- … ???

Review: Nondeterminism

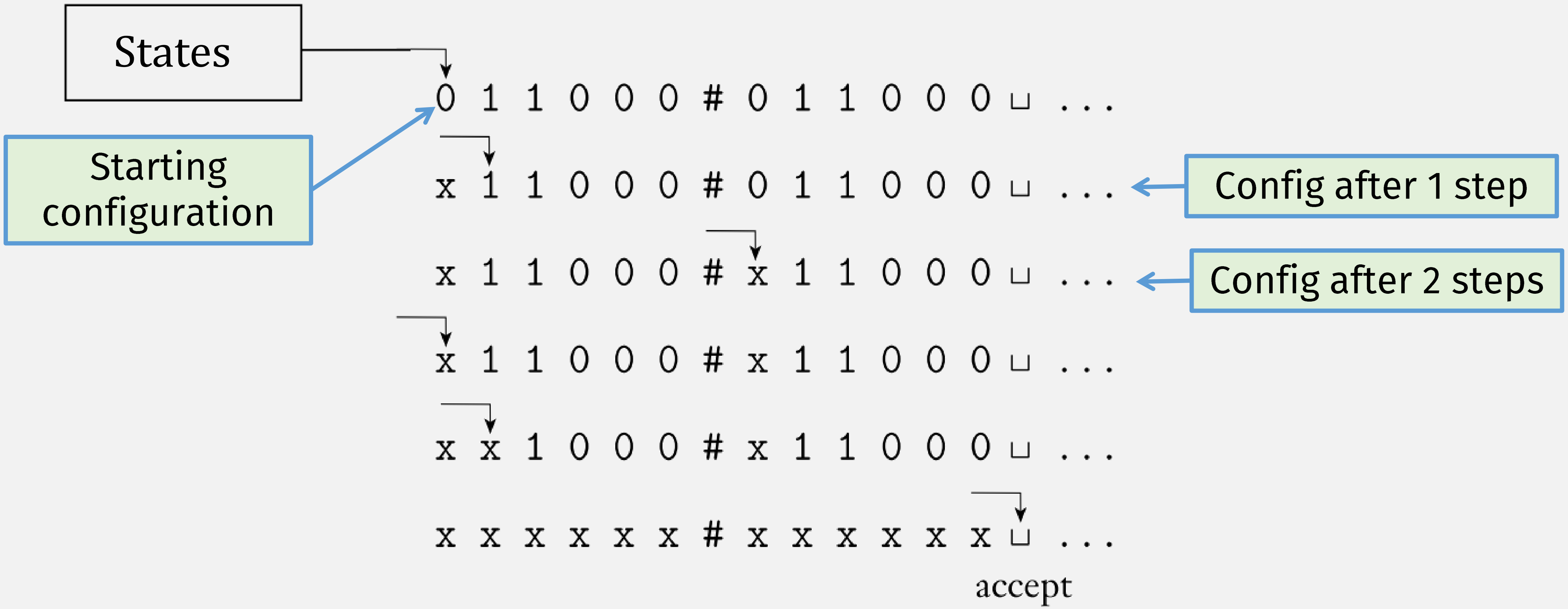


In nondeterministic computation, every step can branch into a set of states

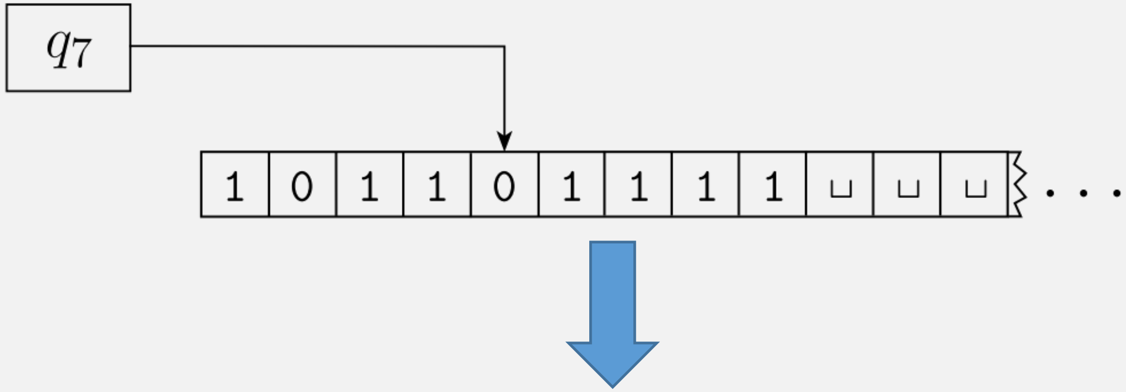
What is a "state" for a TM?

$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

TM Configuration = State + Head + Tape



TM Configuration = State + Head + Tape

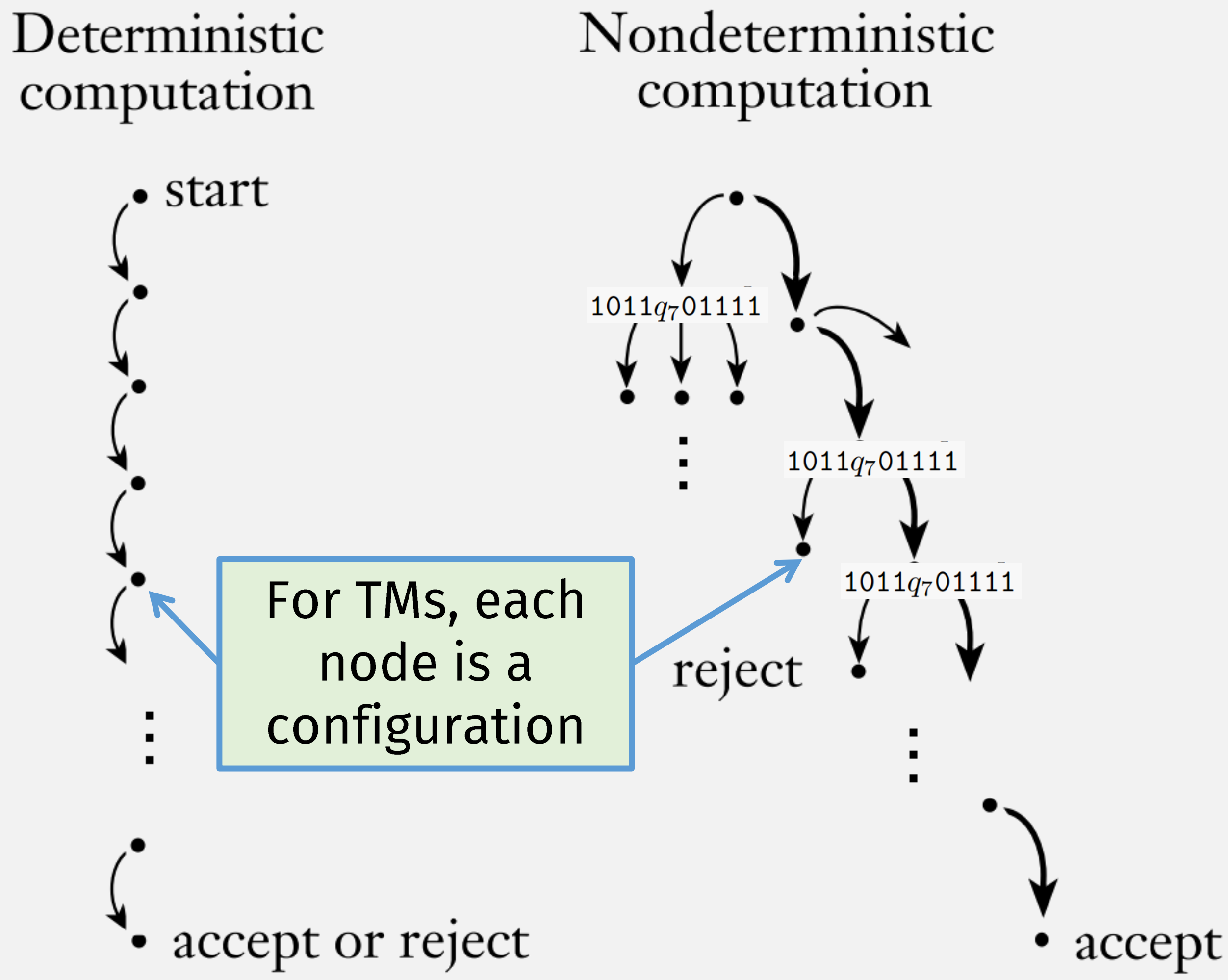


$1011q_701111$

Textual representation of "configuration" (use this in HW6)

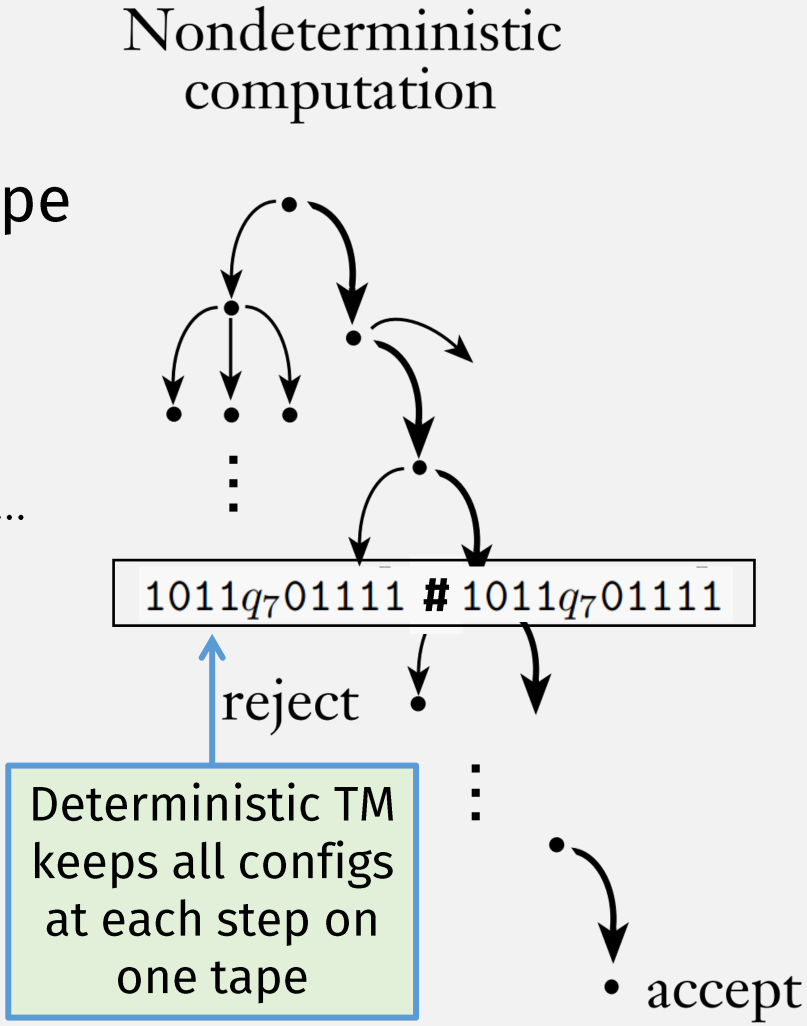
1st char after state is current head position

Nondeterminism in TMs



Nondeterministic TM → Deterministic 1st way

- Simulate NTM with Det. TM:
  - Det. TM keeps multiple configs single tape
    - Like how single-tape TM simulates multi-tape
  - Then run all configs, <u>in parallel</u>
    - I.e., 1 step on one config, 1 step on the next, …
  - Accept if any accepting config is found
  - **Important**:
    - Why must we step configs in parallel?



Interlude: Running TMs inside other TMs

Exercise:

- Given TMs M_1 and M_2, create TM M that accepts if either M_1 or M_2 accept

Possible solution #1:

- M = on input x,
  - Run M_1 on x, accept if M_1 accepts
  - Run M_2 on x, accept if M_2 accepts

M_1	M_2	M
reject	accept	accept
accept	reject	accept ☑

✖

Note: This solution would be ok if we knew M_1 and M_2 were **deciders**

Interlude: Running TMs inside other TMs

Exercise:

- Given TMs M_1 and M_2, create TM M that accepts if either M_1 or M_2 accept

Possible solution #1:

- M = on input x,
  - Run M_1 on x, accept if M_1 accepts
  - Run M_2 on x, accept if M_2 accepts

M_1	M_2	M
reject	accept	accept
accept	reject	accept ☑
accept	loops	accept
loops	accept	loops ✗

Possible solution #2:

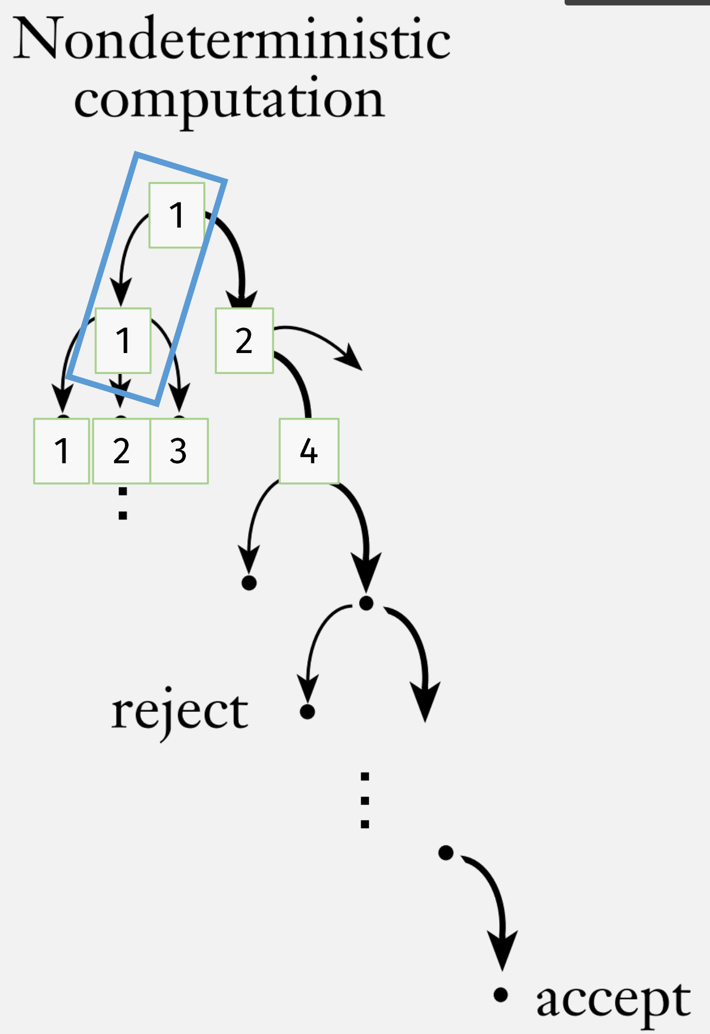
- M = on input x,
  - Run M_1 and M_2 on x in parallel, i.e.,
    - Run M_1 on x for 1 step, accept if M_1 accepts
    - Run M_2 on x for 1 step, accept if M_2 accepts
    - Repeat

M_1	M_2	M
reject	accept	accept
accept	reject	accept ☑
accept	loops	accept
loops	accept	accept ☑

Nondeterministic TM → Deterministic

2nd way
(book's way)

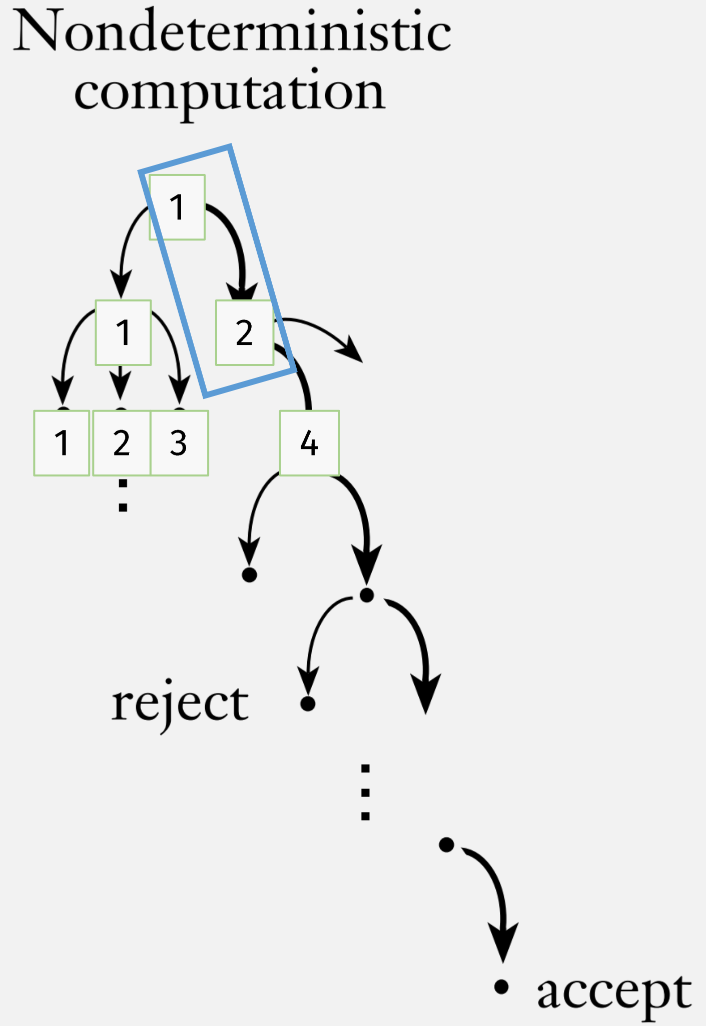
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - **1-1**



Nondeterministic TM → Deterministic

2nd way (book's way)

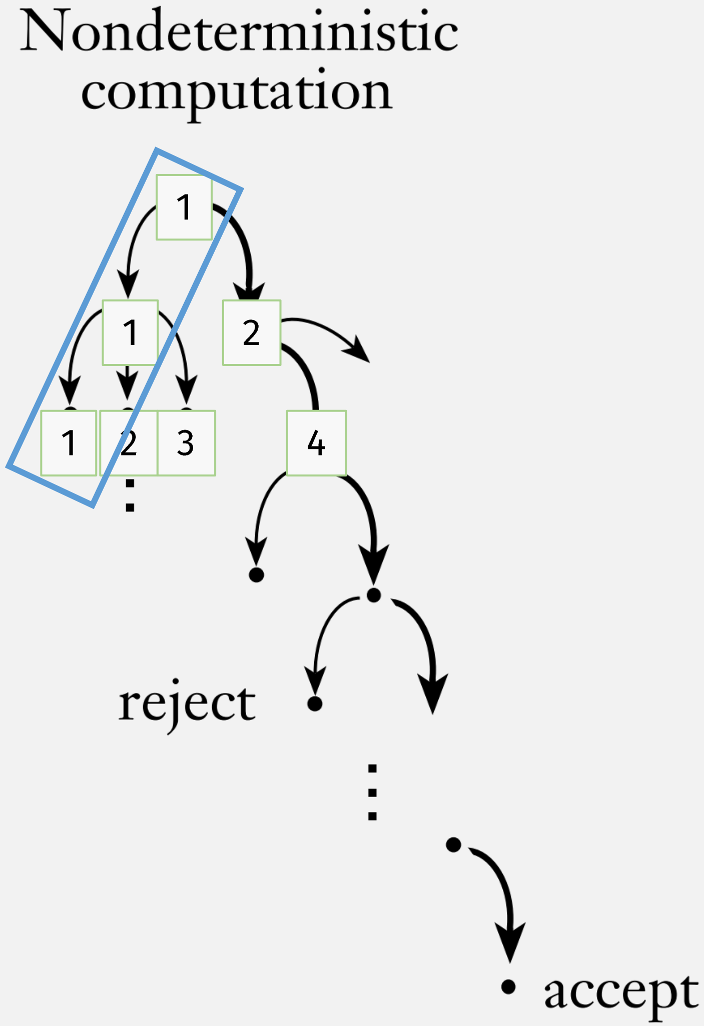
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - 1-1
    - **1-2**



Nondeterministic TM → Deterministic

2nd way
(book's way)

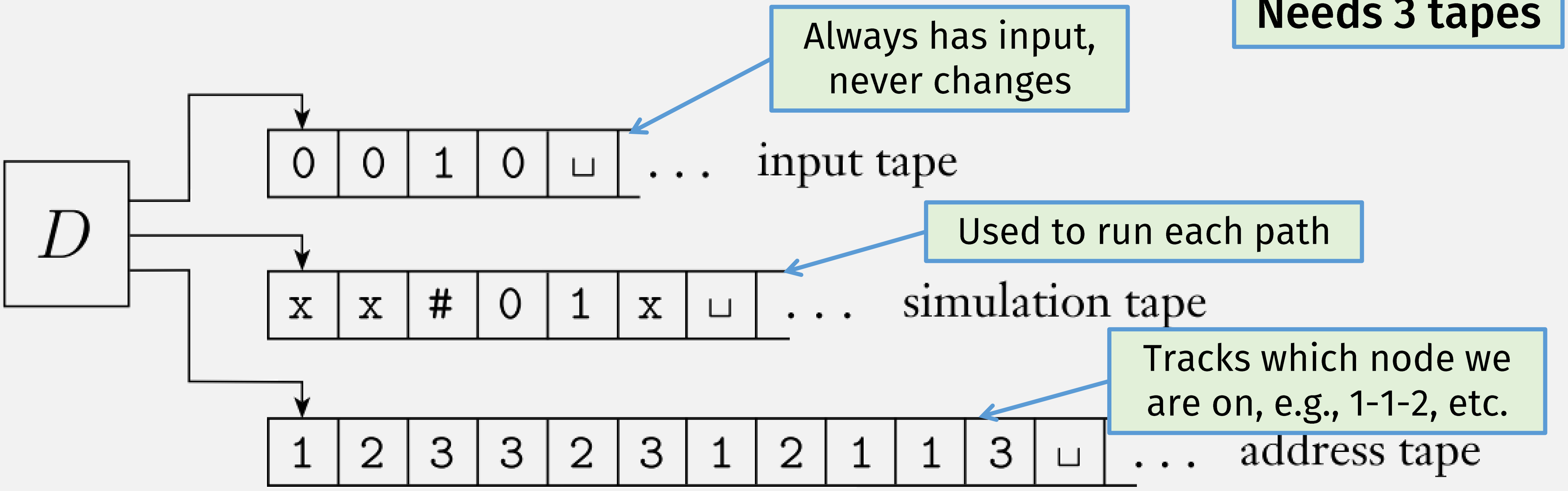
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - 1-1
    - 1-2
    - **1-1-1**



Nondeterministic TM → Deterministic

2nd way
(book's way)

Needs 3 tapes



Nondeterministic TM ⇔ Deterministic TM

=> **If** a deterministic TM recognizes a language,
then a nondeterministic TM recognizes the language

- To convert Deterministic TM → Non-deterministic TM …
- … change Deterministic TM delta fn output to a one-element set
  - (just like conversion of DFA to NFA)
- **DONE!**

<= **If** a nondeterministic TM recognizes a language,
then a deterministic TM recognizes the language

- Convert Nondeterministic TM → Deterministic TM
- **DONE!** ■

Conclusion: These are All Equivalent TMs!

- Single-tape Turing Machine
- Multi-tape Turing Machine
- Non-deterministic Turing Machine

Check-in Quiz 3/22

On gradescope