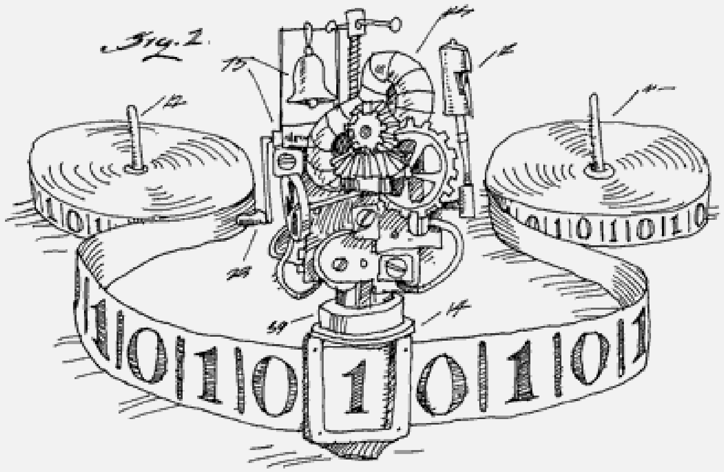
UMB CS420

Turing Machines (TMs)

Wednesday, March 2, 2022



Announcements

- HW 5 due Sun 3/6 11:59pm
- Reminder: "type check" your work!
- Example: $\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$

1st arg must be a state (from set Q)

2nd arg must be a char, or ε

Output must be set of states!

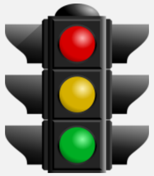
CS 420 So Far, and Looking Forward

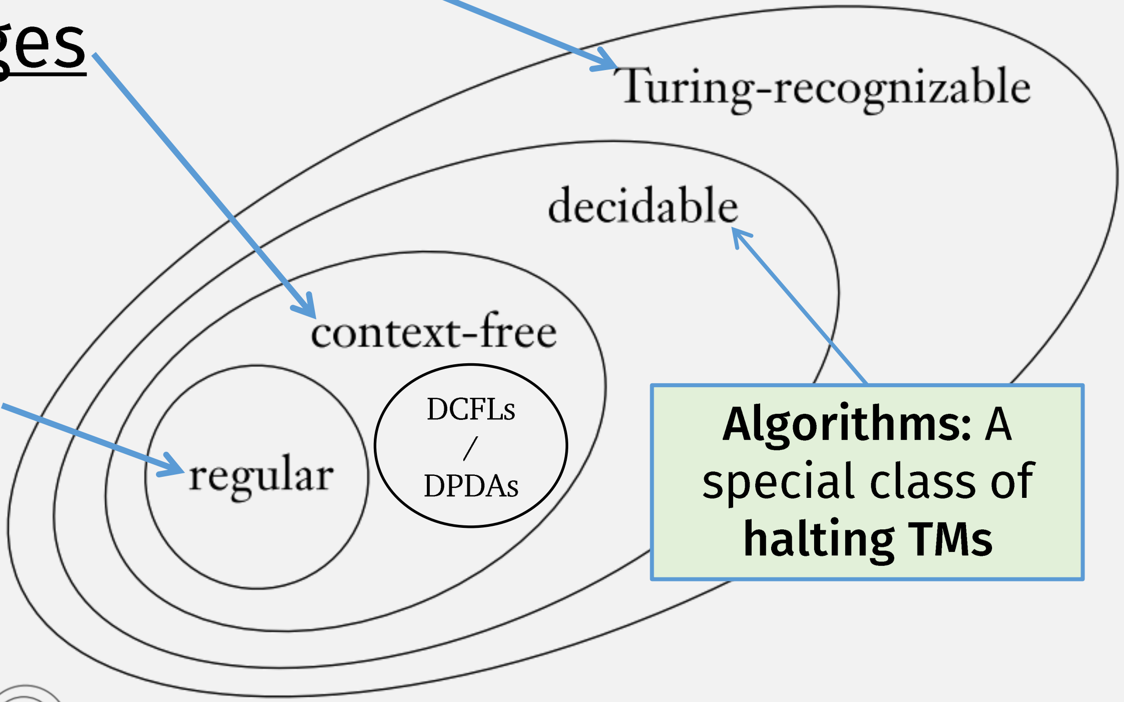
- **Turing Machines** (TMs)

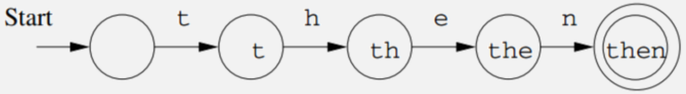
  - Infinite tape (memory), arbitrary read/write
  - Expresses any "computation"
- **PDAs:** recognize <u>context-free languages</u>
  - Infinite stack (memory), push/pop only
  - Can't express: <u>arbitrary</u> dependency,
    - e.g., $\{ww \mid w \in \{0,1\}^*\}$

$A \to 0A1$
$A \to B$
$B \to \#$

- **DFAs / NFAs:** recognize <u>regular langs</u>
  - Finite states (memory)
  - Can't express: **dependency**
    e.g., $\{0^n1^n \mid n \geq 0\}$



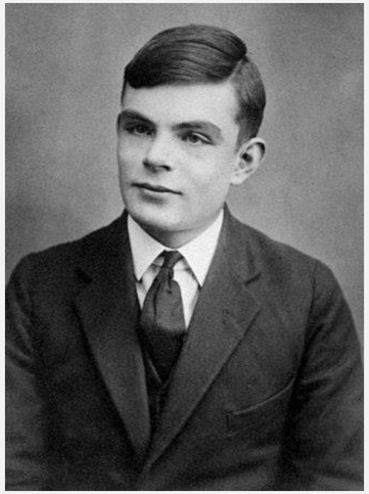




Alan Turing

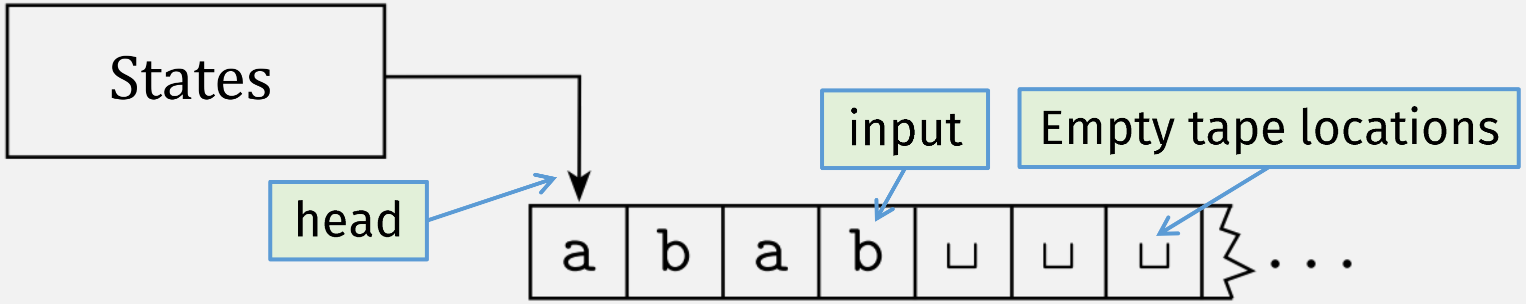
- First to formalize the models of computation we're studying
  - I.e., he invented this course
- Worked as codebreaker during WW2

- Also studied Artificial Intelligence
  - The Turing Test



Finite Automata vs Turing Machines

- Turing Machines can read and write to arbitrary "tape" cells
  - 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

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

Turing Machine Example

This is an **informal TM description**
One "step" =
multiple formal transitions

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

input

tape

head

0 1 1 0 0 0 # 0 1 1 0 0 0 ␣ . . .

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.

"Cross off" =
write "x" char

Turing Machine Example

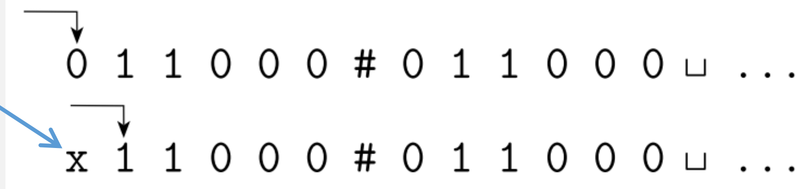
M_1 accepts inputs in language $B = \{w\#w \mid w \in \{0,1\}^*\}$

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.

"Cross off" = write "x" char

"Cross off" = write "x" char



Turing Machine Example

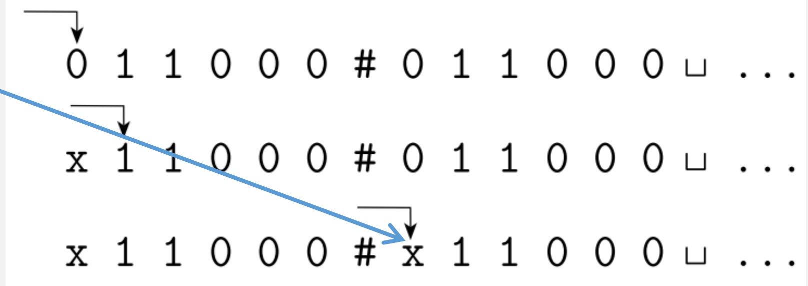
M_1 accepts inputs in language $B = \{w\#w \mid w \in \{0,1\}^*\}$

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.

"Cross off" = write "x" char

"Cross off" = write "x" char



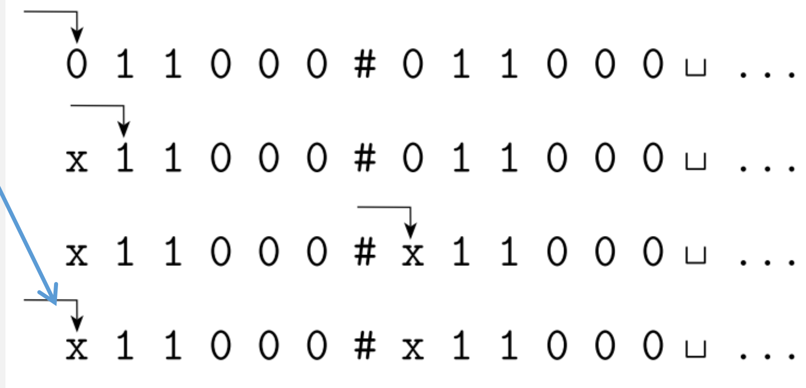
Turing Machine Example

M_1 accepts inputs 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.

"zag" to start



"Cross off" = write "x" char

Turing Machine Example

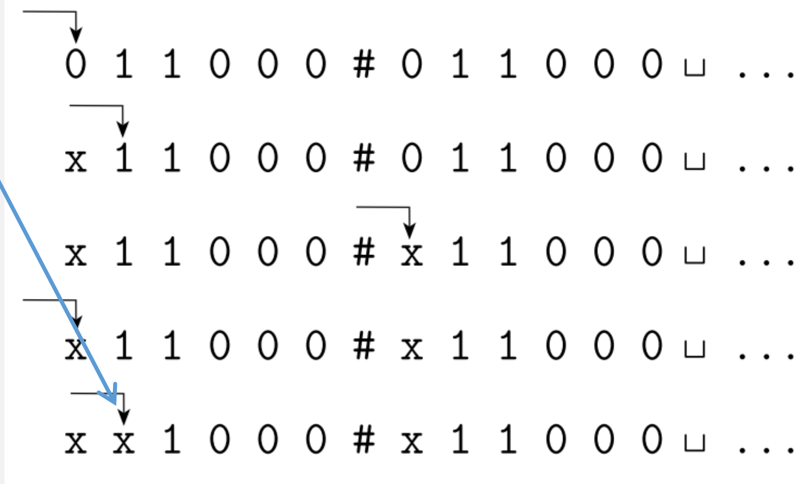
M_1 accepts inputs in language $B = \{w\#w \mid w \in \{0,1\}^*\}$

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.

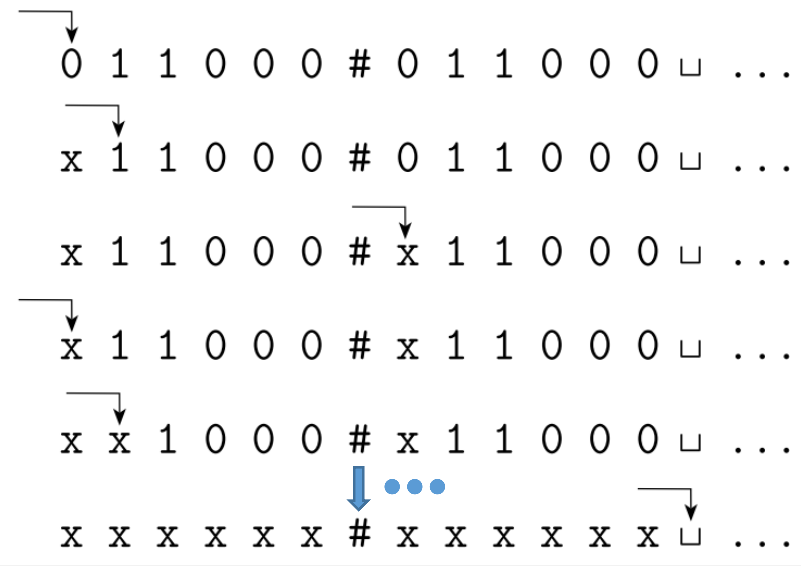
"Cross off" = write "x" char

Continue crossing off



Turing Machine Example

M_1 accepts inputs in language $B = \{w\#w |\ w \in \{0,1\}^*\}$



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*."

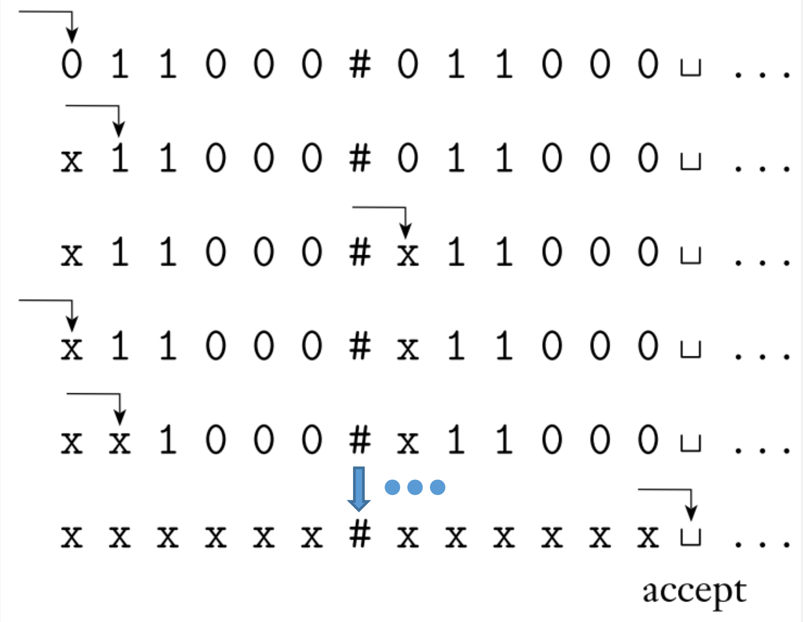
Turing Machine Example

M_1 accepts inputs 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*."



Turing Machines: Formal Definition

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

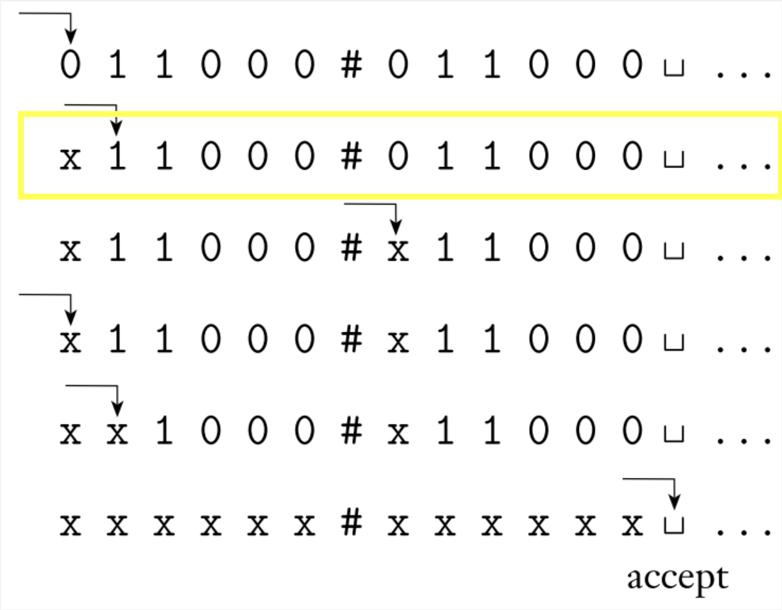
1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,

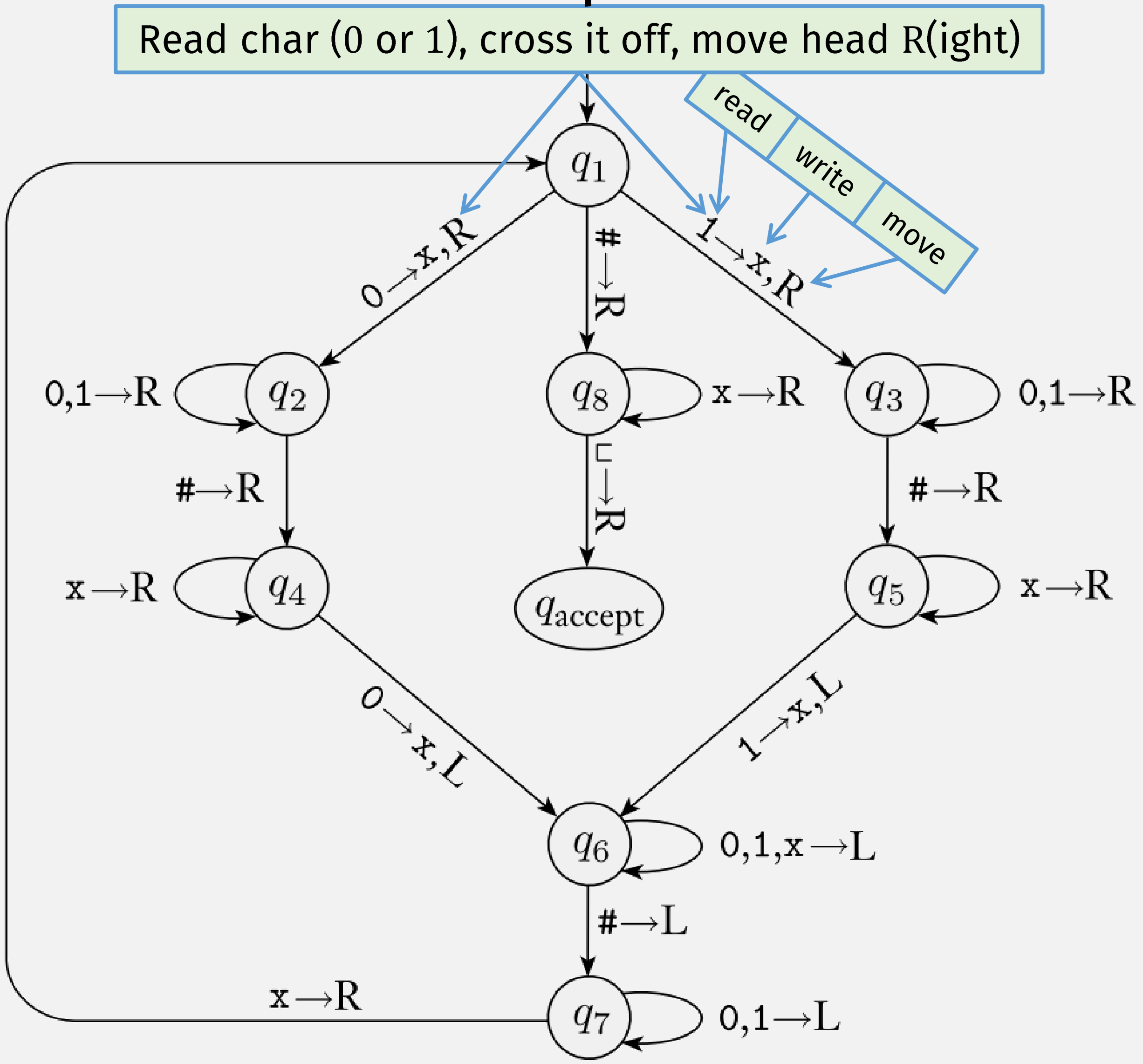
   read write move

5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

$B = \{w\#w \mid w \in \{0,1\}^*\}$

Formal Turing Machine Example



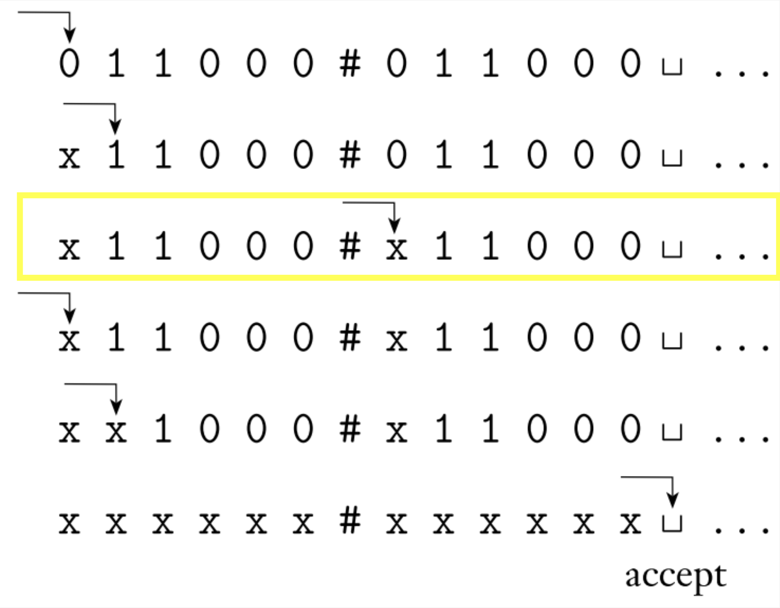


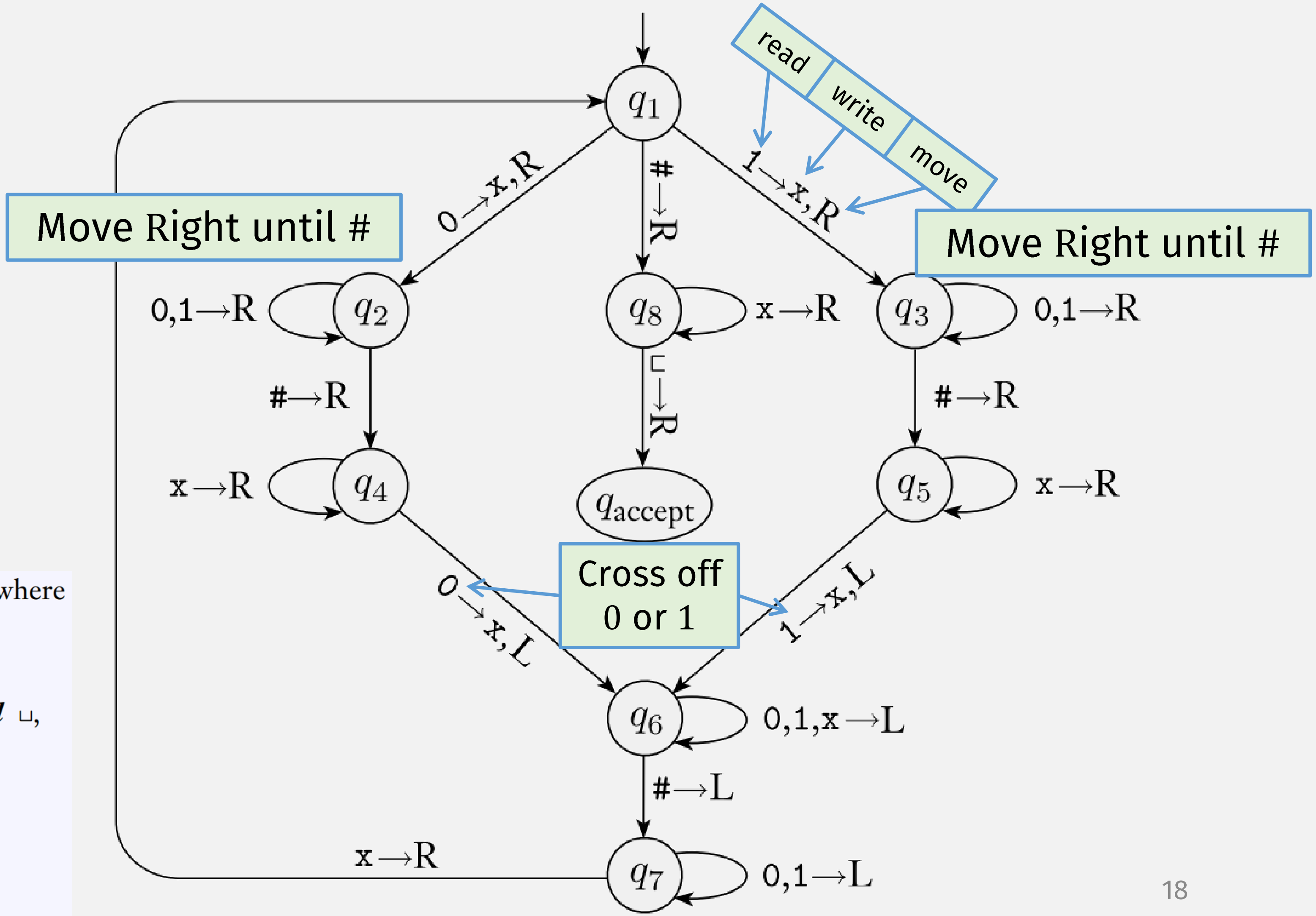
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

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. $\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in$ read e s write move
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

$$B = \{w\#w \mid w \in \{0,1\}^*\}$$

Formal Turing Machine Example



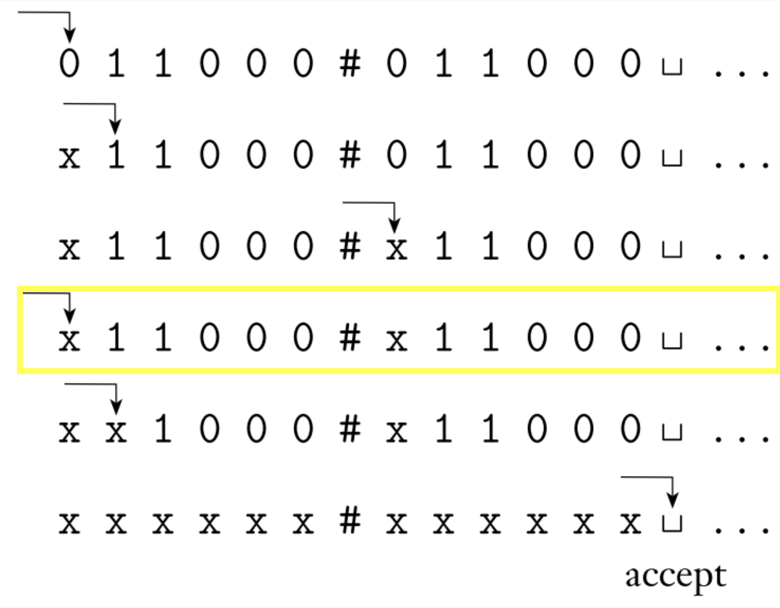


A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

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

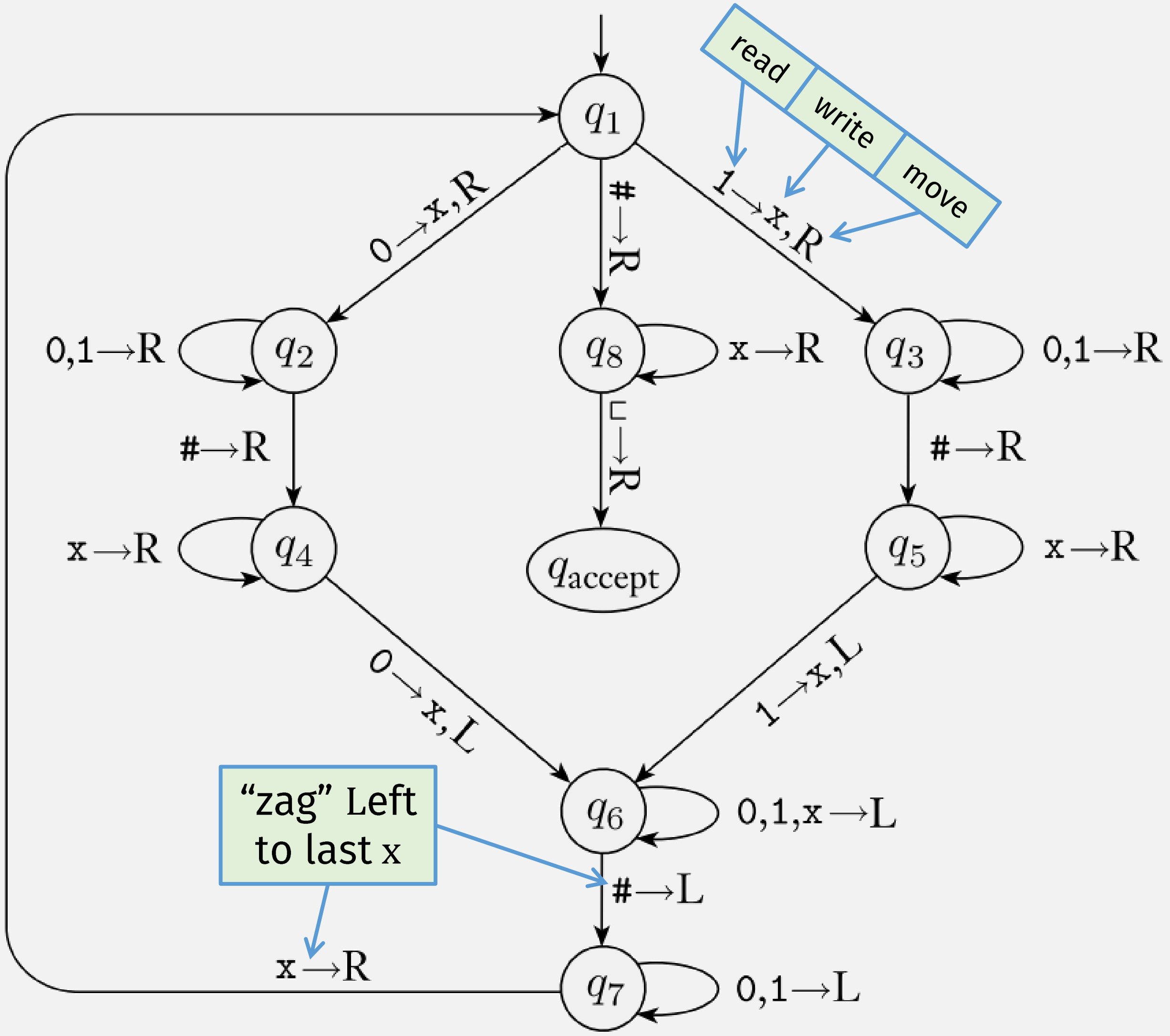
$B = \{w\#w \mid w \in \{0,1\}^*\}$

Formal Turing Machine Example



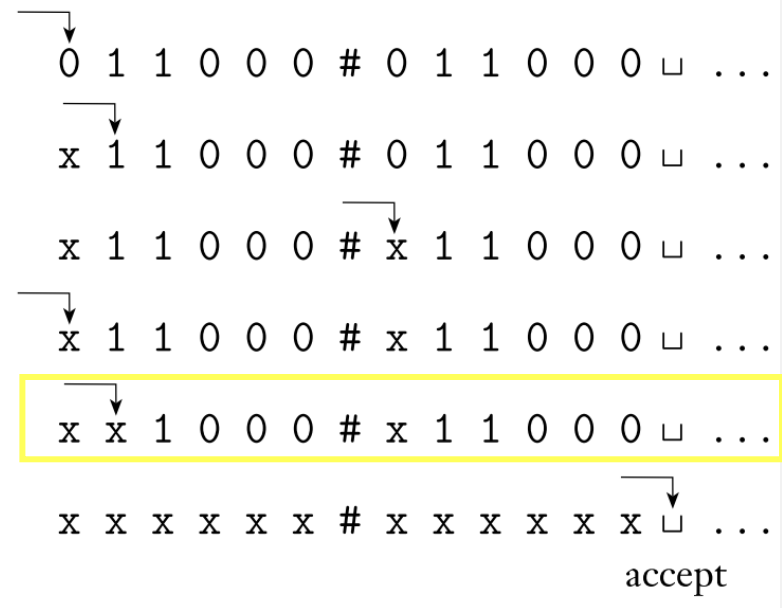
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

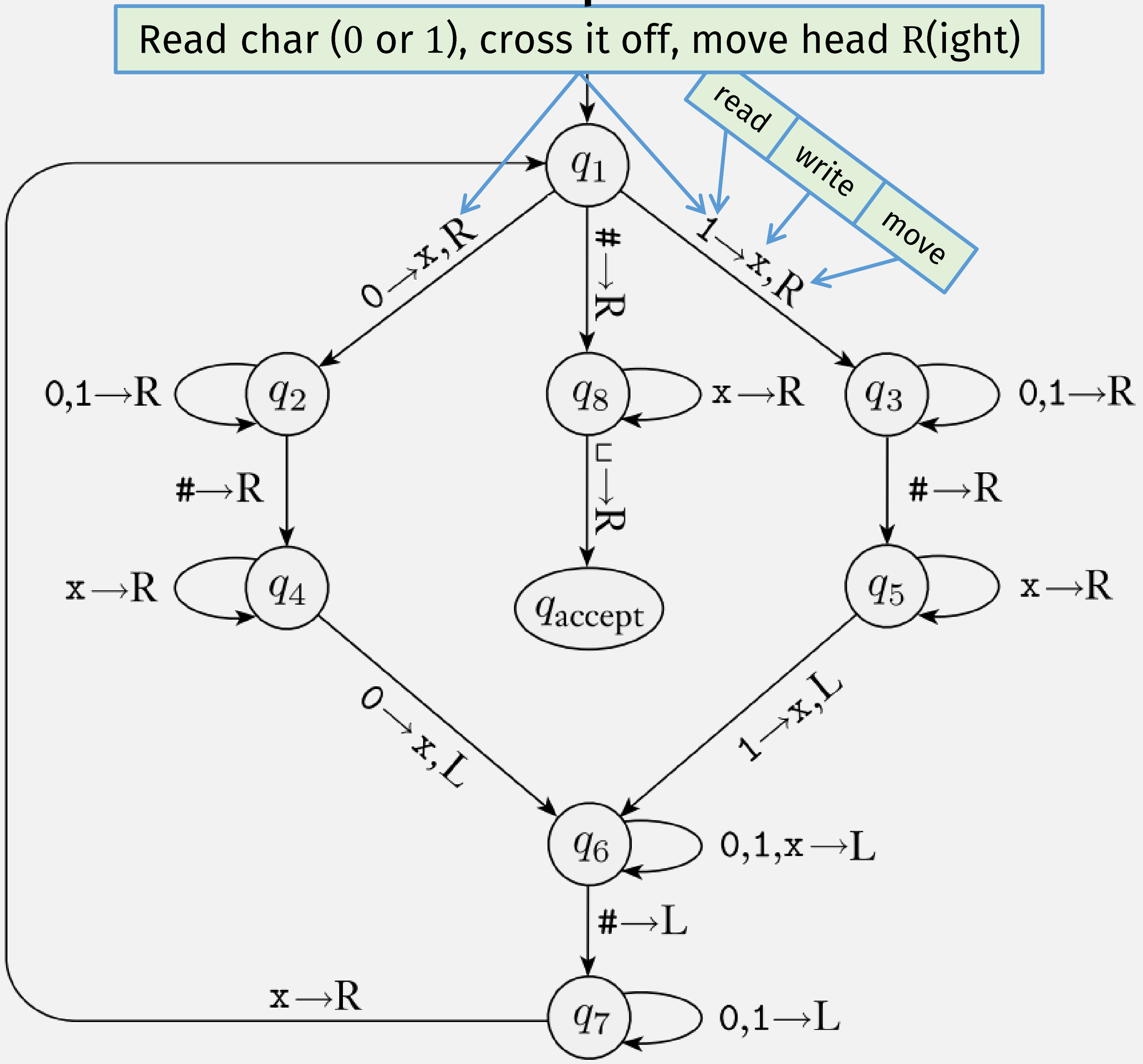
1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** ␣,
3. Γ is the tape alphabet, where ␣ $\in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in$ read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



$B = \{w\#w \mid w \in \{0,1\}^*\}$

Formal Turing Machine Example



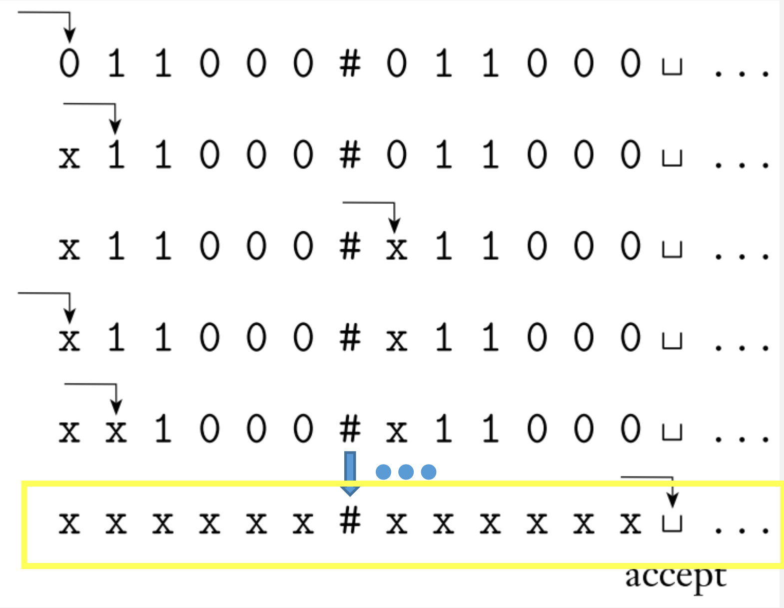


A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** ⊔,
3. Γ is the tape alphabet, where $⊔ \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in$ [read] e s [write] [move]
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

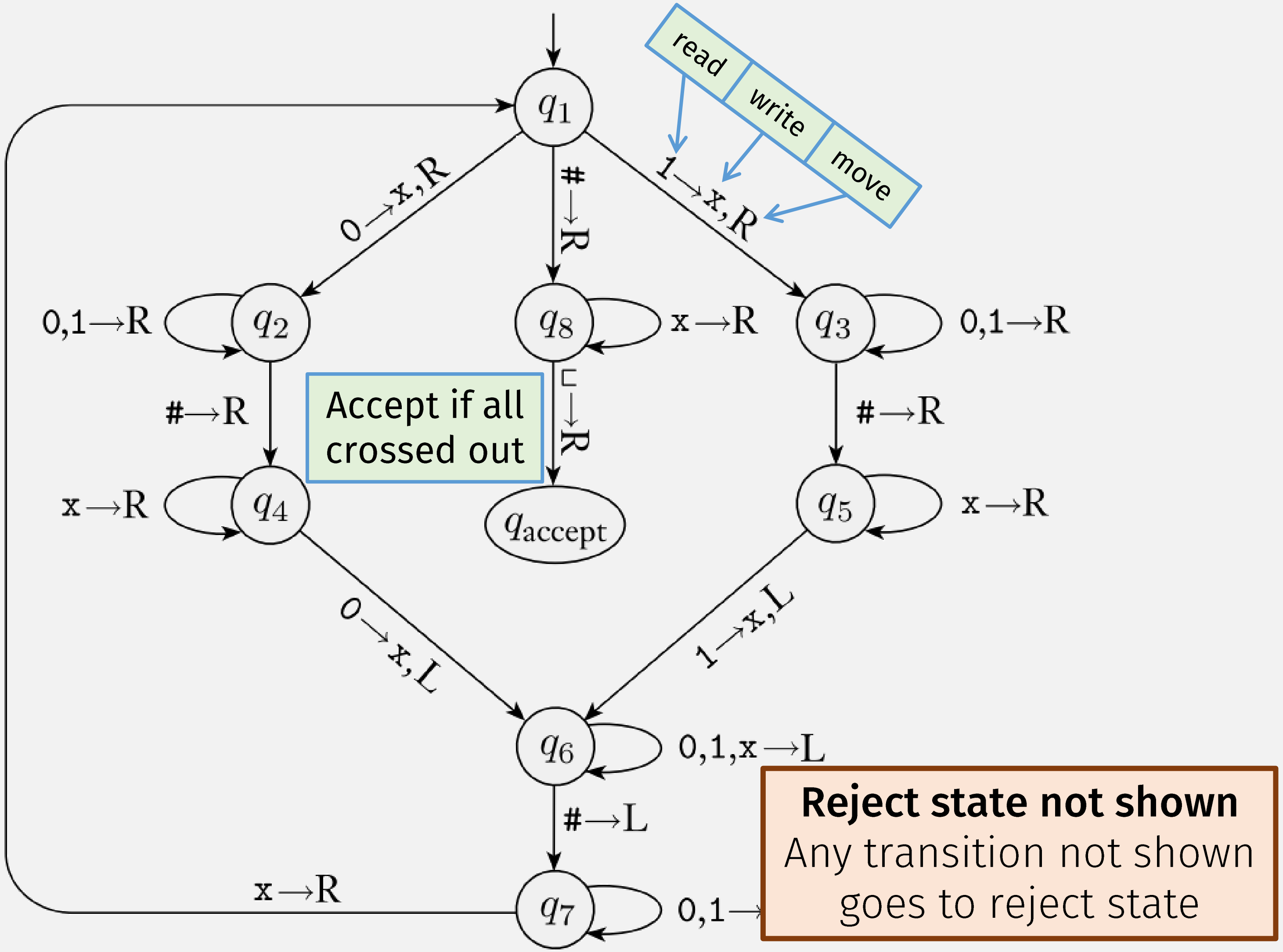
$B = \{w\#w \mid w \in \{0,1\}^*\}$

Formal Turing Machine Example



A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, 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*** ⊔,
3. Γ is the tape alphabet, where $⊔ \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in$ read e s write move
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.



Turing Machine: Informal Description

- M_1 accepts if input is 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 no … if no # is found, *reject*. Cross off symbols as they … p track of which symbols correspond.
2. When all symbols to … n crossed off, check for any remaining s … nt of the #. If any symbols remain, *reject*; otherwise, … ccept."

We will (mostly) stick to informal descriptions of Turing machines, like this one

TM Informal Description: Caveats

- TM informal descriptions are not a “do whatever” card
  - They must still represent the formal tuple
- <u>Input</u> must be a string, written with chars from finite alphabet
- An informal “<u>step</u>” represents a <u>finite</u> # of formal transitions
  - It cannot run forever
  - E.g., can’t say “try all numbers” as a “step”

Non-halting Turing Machines (TMs)

IMPORTANT SLIDE

- A DFA, NFA, or PDA always halts
  - Because the (finite) input is always read exactly once

- But a Turing Machine can run forever
  - E.g., the head can move back and forth in a loop

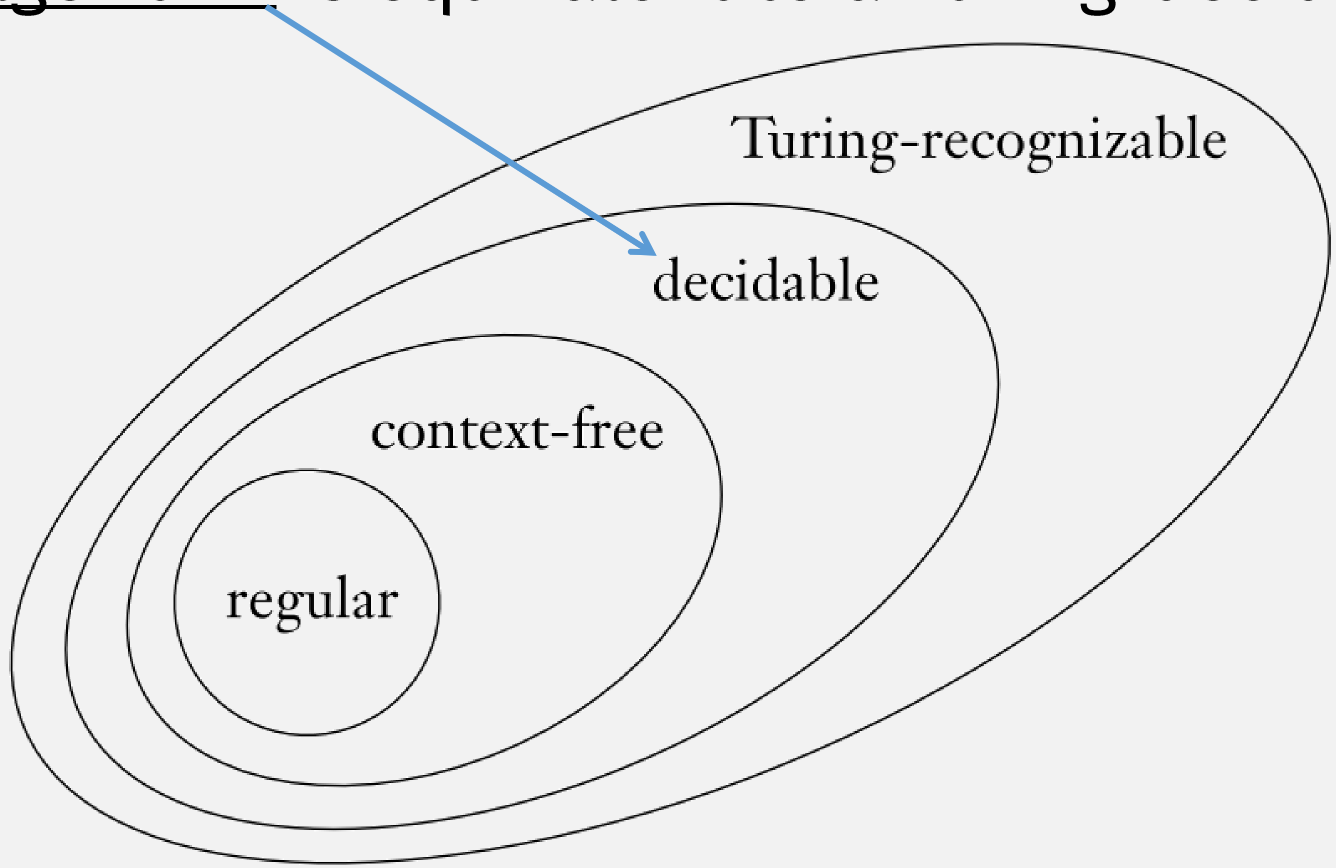
- Thus, there are two classes of Turing Machines:
  - A **recognizer** is a Turing Machine that may run forever (all possible TMs)
  - A **decider** is a Turing Machine that always halts.

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

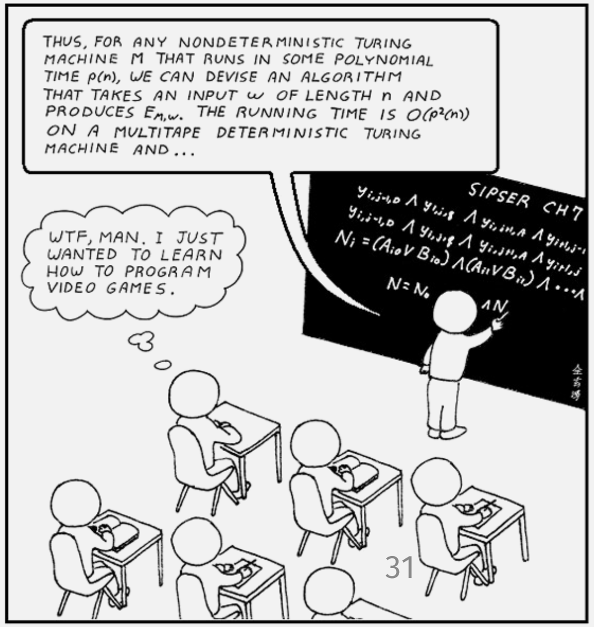
Call a language ***Turing-decidable*** or simply ***decidable*** if some Turing machine decides it.

Formal Definition of an "Algorithm"

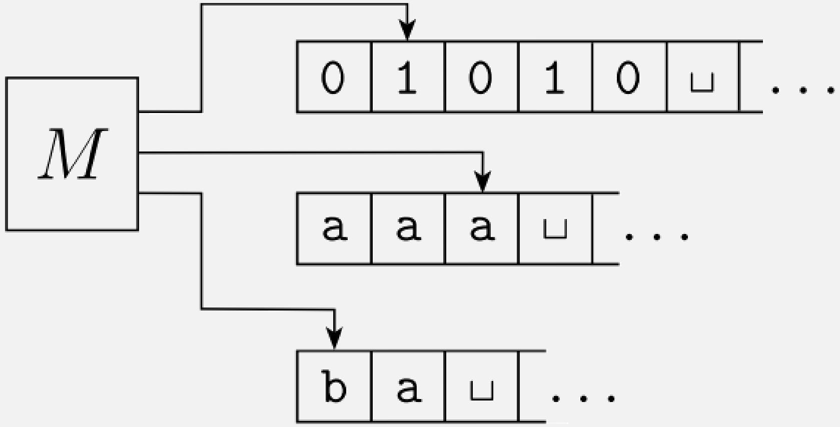
- An algorithm is equivalent to a Turing-decidable Language



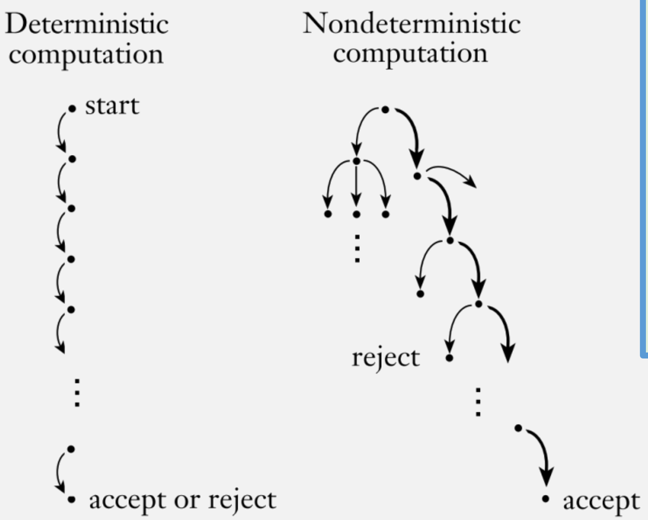
Turing Machine Variations



1. **Multi-tape TMs**



2. **Non-deterministic TMs**



We will prove that these TM variations are **equivalent to** deterministic, single-tape machines

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!**

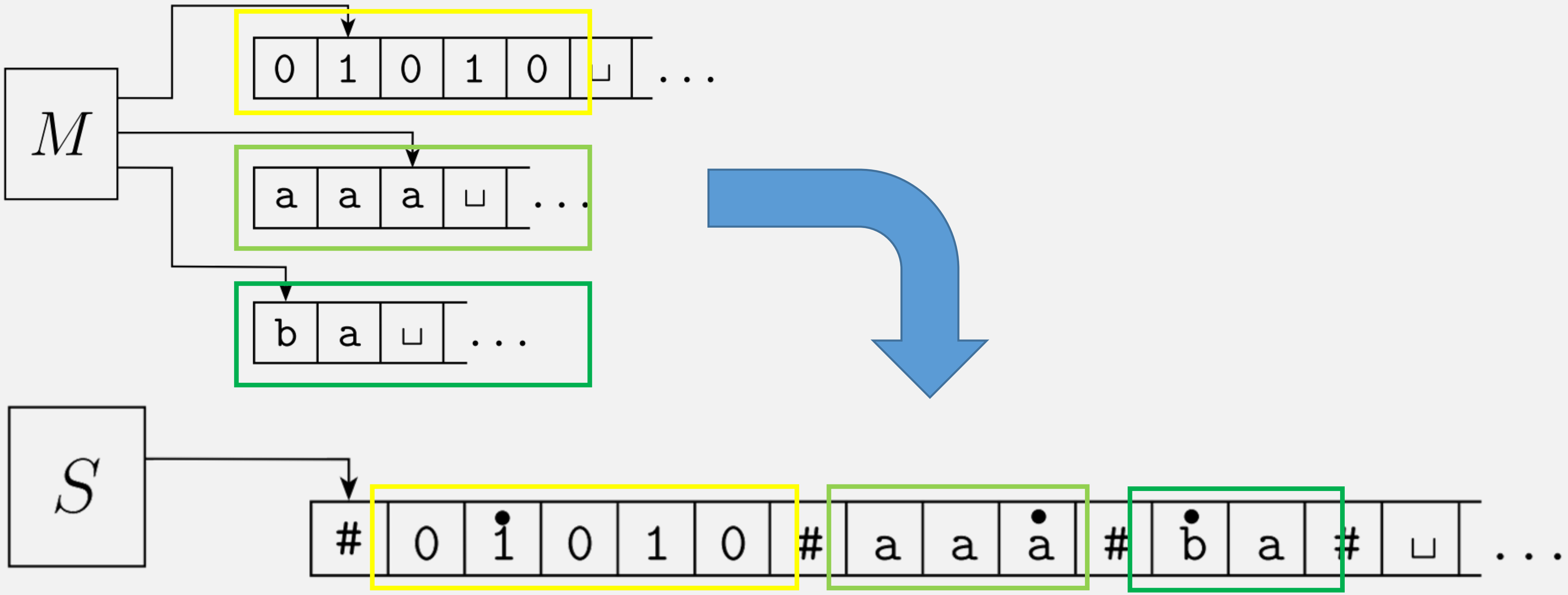
⇐ **If** a multi-tape TM recognizes a language,
then a single-tape TM recognizes the language

- 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

☑ ⇐ **If** a multi-tape TM recognizes a language,
then a single-tape TM recognizes the language

- Convert multi-tape TM to single-tape TM

■

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. $\delta : Q \times \Sigma \longrightarrow Q$ 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. $\delta : Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Nondeterministic transition produces set of possible next states

Remember: Turing Machine Formal Definition

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 $q_{\text{reject}} \neq q_{\text{accept}}$.

Nondeterministic Remember: Turing Machine Formal Definition

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

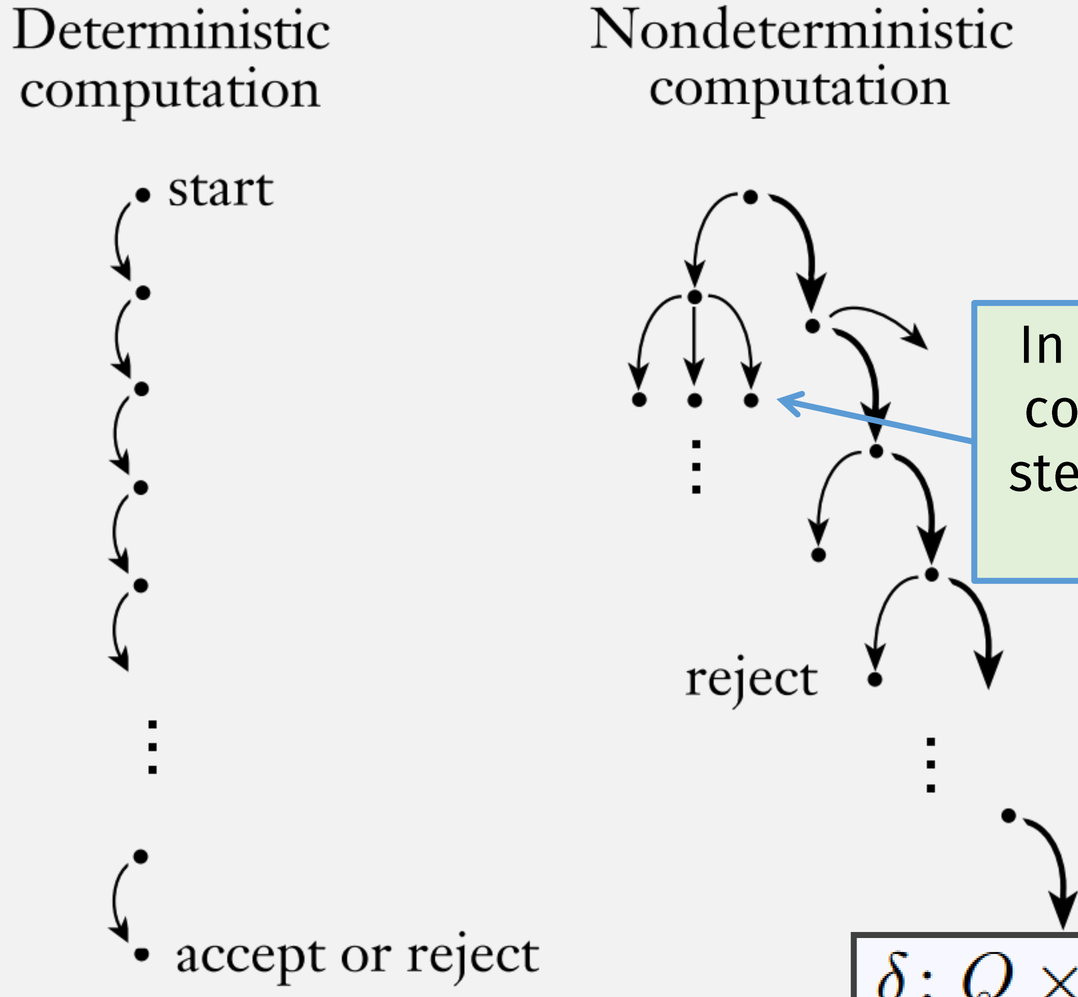
⇒ **If** a deterministic TM recognizes a language,
then a nondeterministic TM recognizes the language

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

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

Flashback: PDA Configurations (IDs)

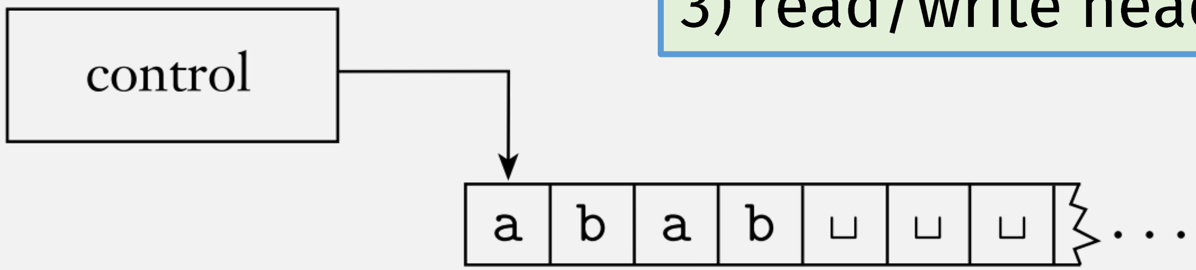
- A **configuration** (or **ID**) is a snapshot of a PDA's computation

- A configuration (or **ID**) (q, w, γ) has three components:
  q = the current state
  w = the remaining input string
  γ = the stack contents

TM Configuration (ID) = ???

1) states

3) read/write head

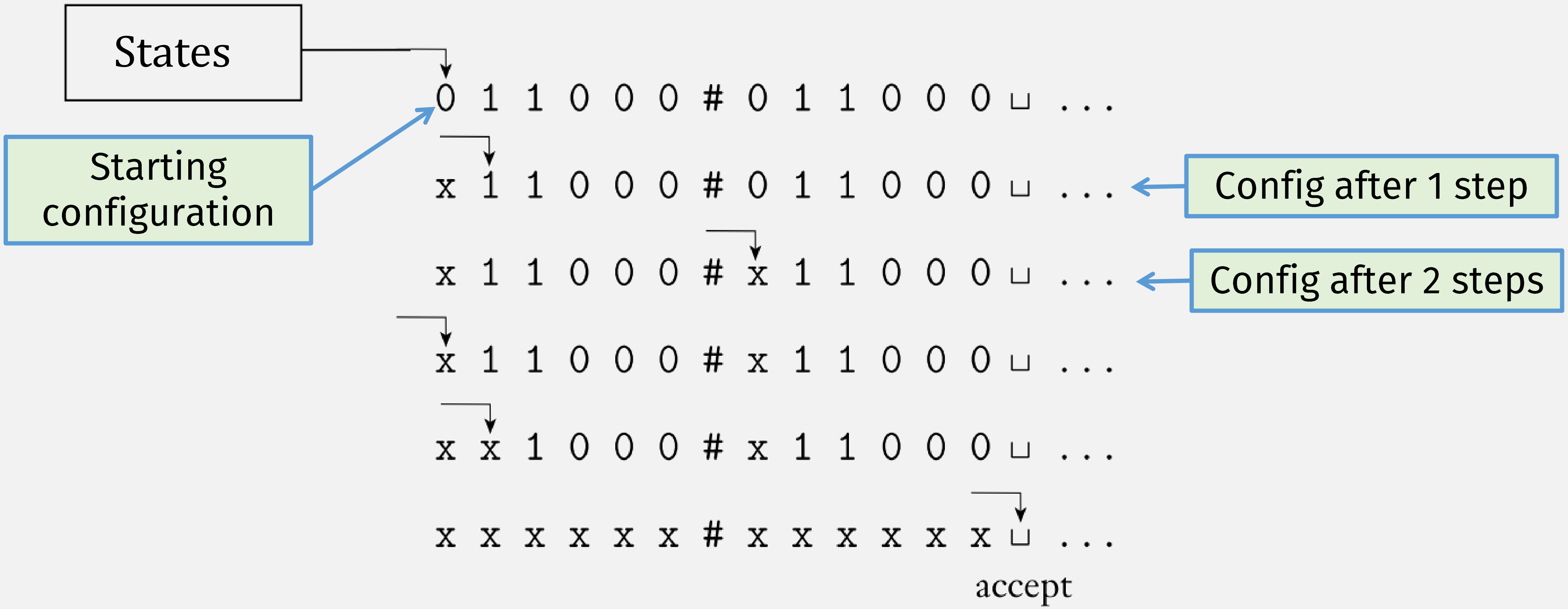


2) Tape contents

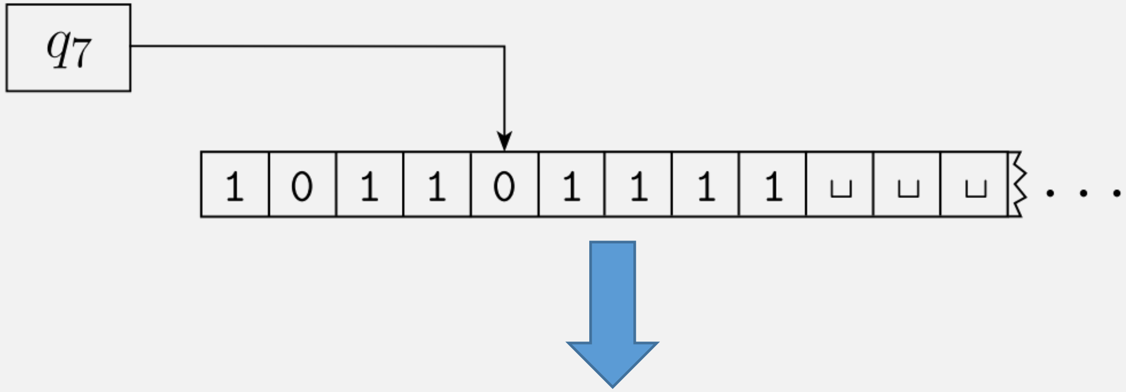
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

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. $\delta\colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

TM Configuration = State + Head + Tape



TM Configuration = State + Head + Tape



$$1011q_701111$$

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

1st char after state is current head position

TM Computation, Formally

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

Single-step

head — Next config

(Right) $\alpha q_1 \mathbf{a} \beta \vdash \alpha \mathbf{x} q_2 \beta$

write

if $q_1, q_2 \in Q$

$\delta(q_1, \mathbf{a}) = (q_2, \mathbf{x}, \mathrm{R})$

read — $\mathbf{a}, \mathbf{x} \in \Gamma \quad \alpha, \beta \in \Gamma^*$

(Left) $\alpha b q_1 \mathbf{a} \beta \vdash \alpha q_2 b \mathbf{x} \beta$

if $\delta(q_1, \mathbf{a}) = (q_2, \mathbf{x}, \mathrm{L})$

Edge cases: $q_1 \mathbf{a} \beta \vdash q_2 \mathbf{x} \beta$ if $\delta(q_1, \mathbf{a}) = (q_2, \mathbf{x}, \mathrm{L})$

$\alpha q_1 \vdash \alpha \sqcup q_2$ if $\delta(q_1, \sqcup) = (q_2, \sqcup, \mathrm{R})$

Extended

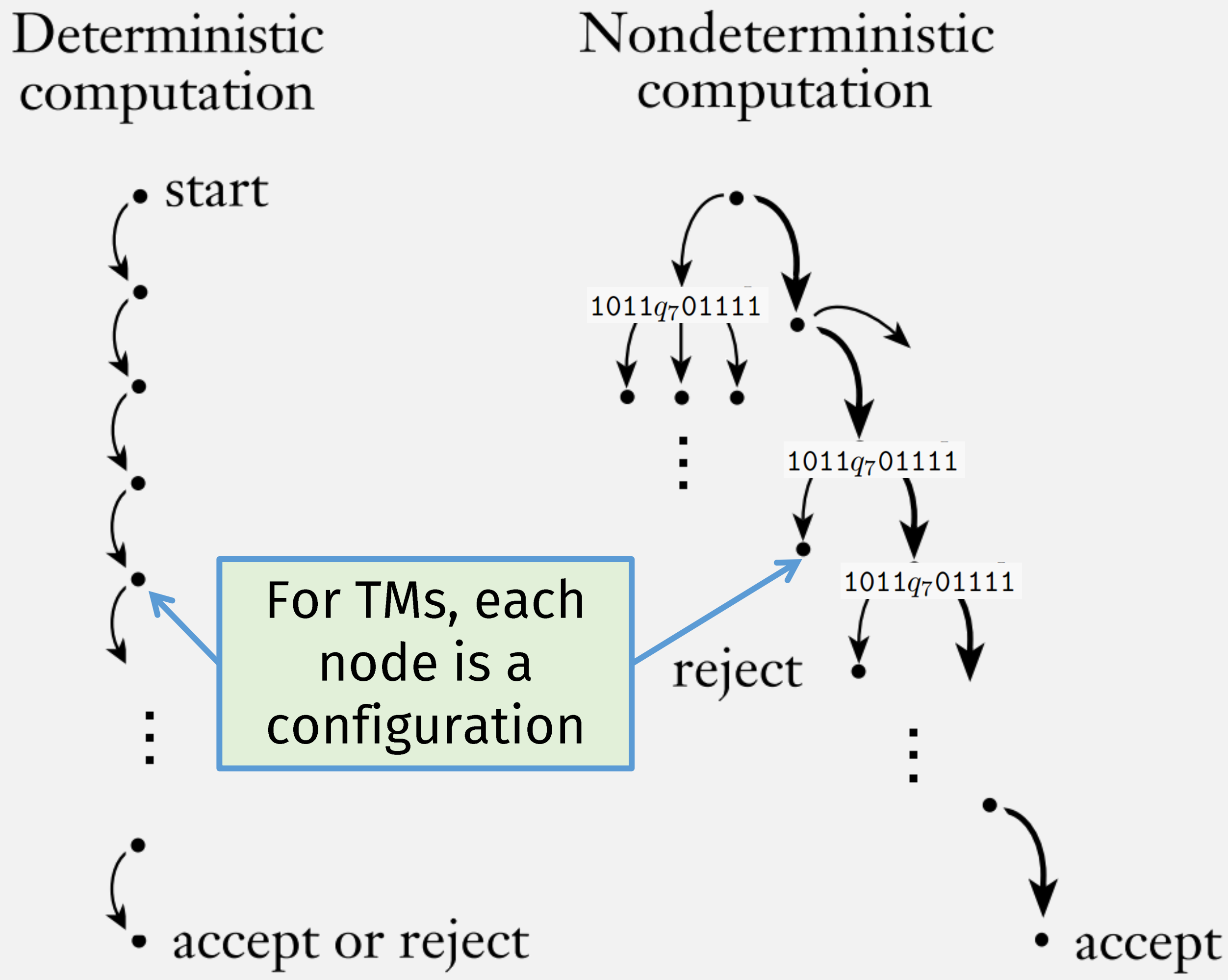
- Base Case

  $I \overset{*}{\vdash} I$ for any ID I

- Recursive Case

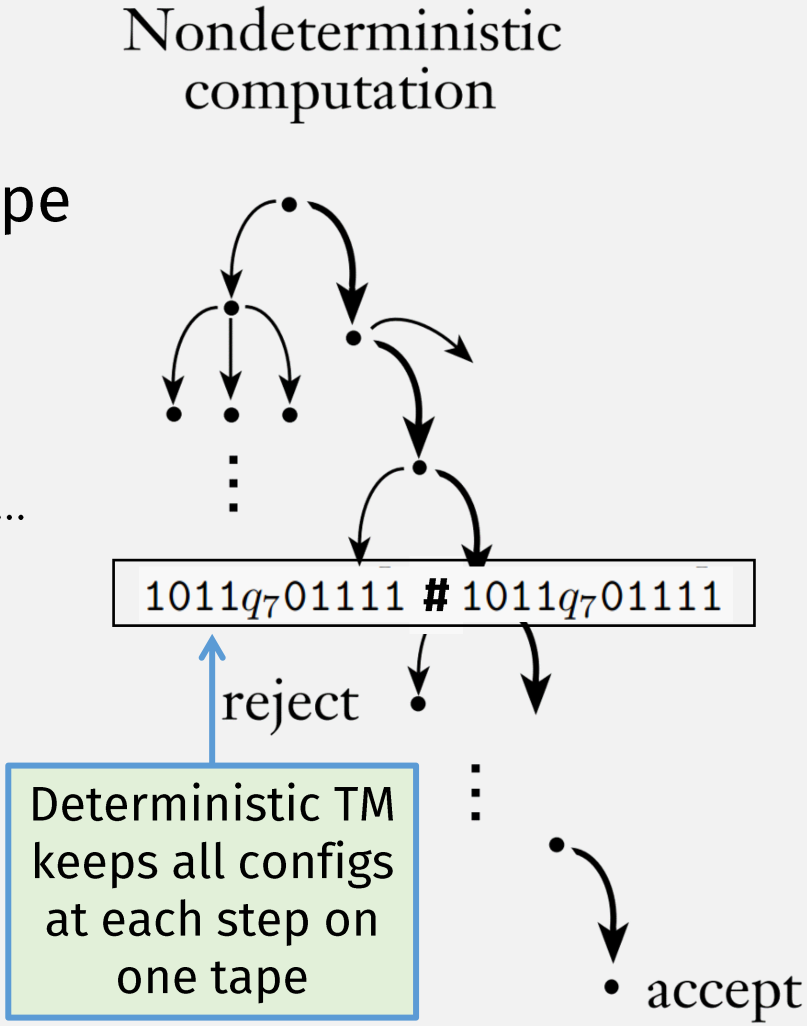
  $I \overset{*}{\vdash} J$ if there exists some ID K such that $I \vdash K$ and $K \overset{*}{\vdash} J$

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** (which halt on all inputs)

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:

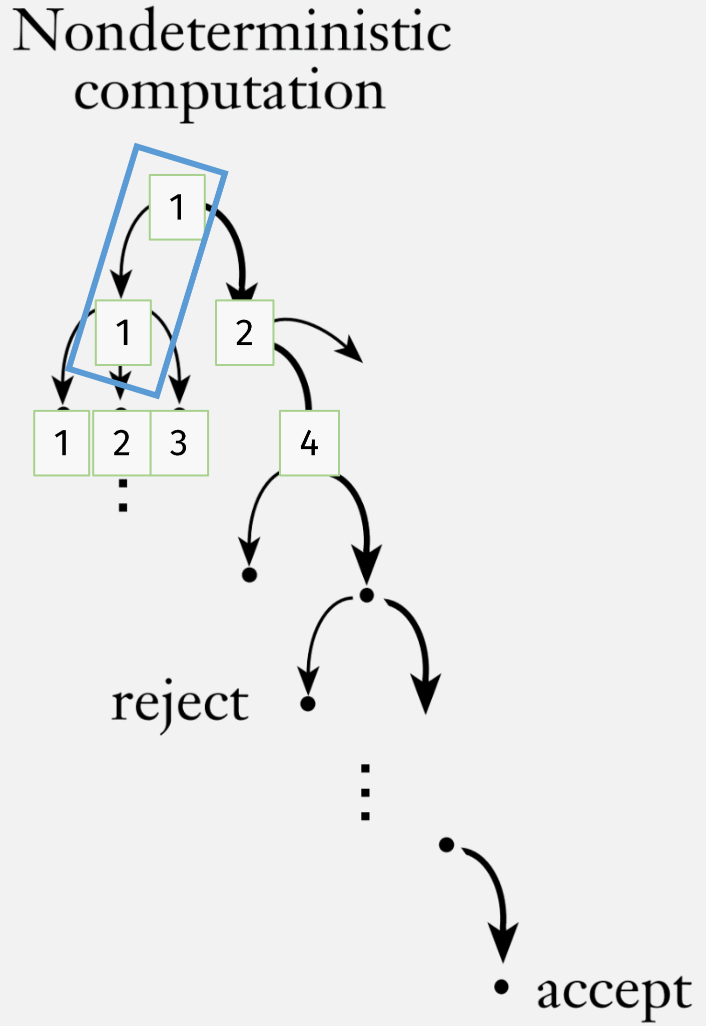
- M = on input x,
  - Run M_1 and M_2 on x in parallel, i.e.,
    - Run M_1 on x for 1 step, accept if M_1 accepts
    - Run M_2 on x for 1 step, accept if M_2 accepts
    - Repeat

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

Nondeterministic TM → Deterministic

2nd way
(Sipser)

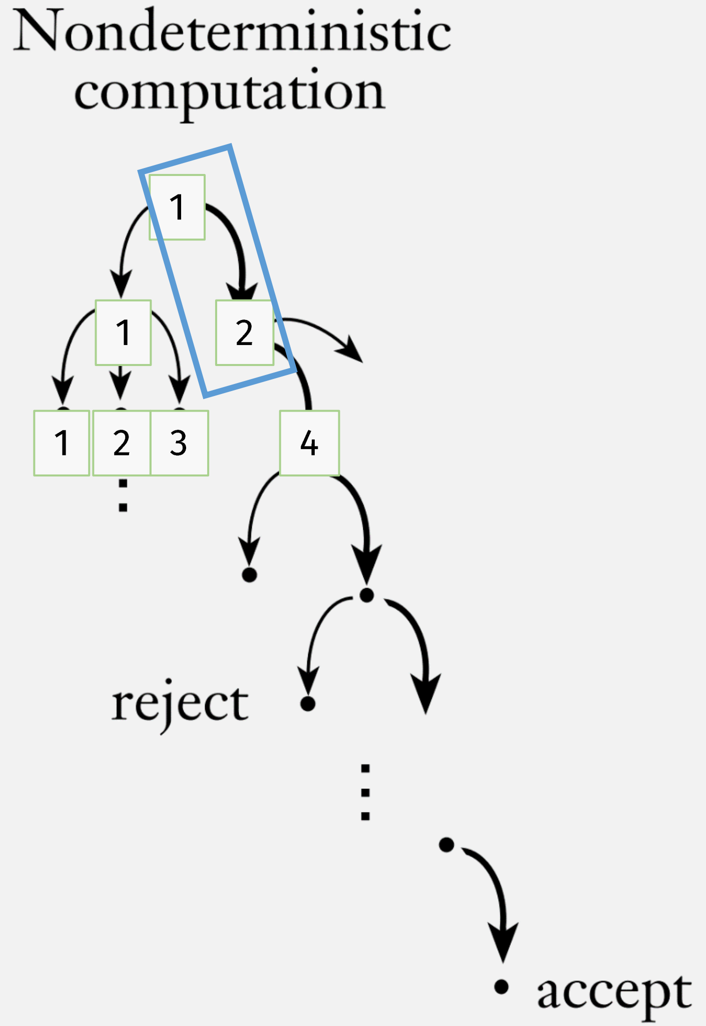
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - **1-1**



Nondeterministic TM → Deterministic

2nd way
(Sipser)

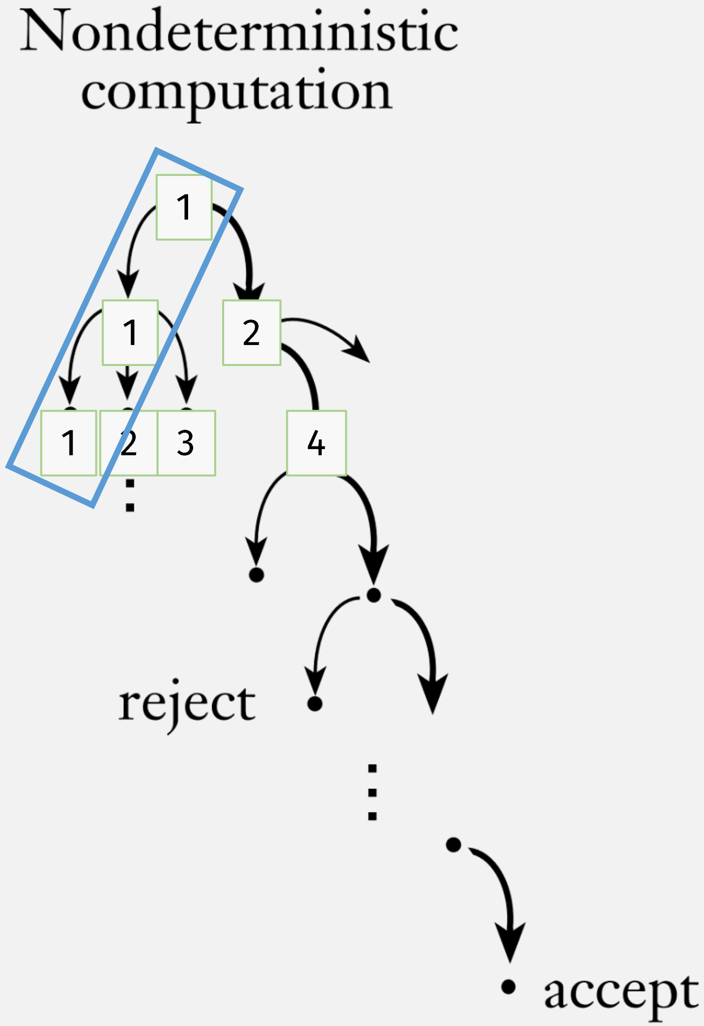
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - 1-1
    - **1-2**



Nondeterministic TM → Deterministic

2nd way
(Sipser)

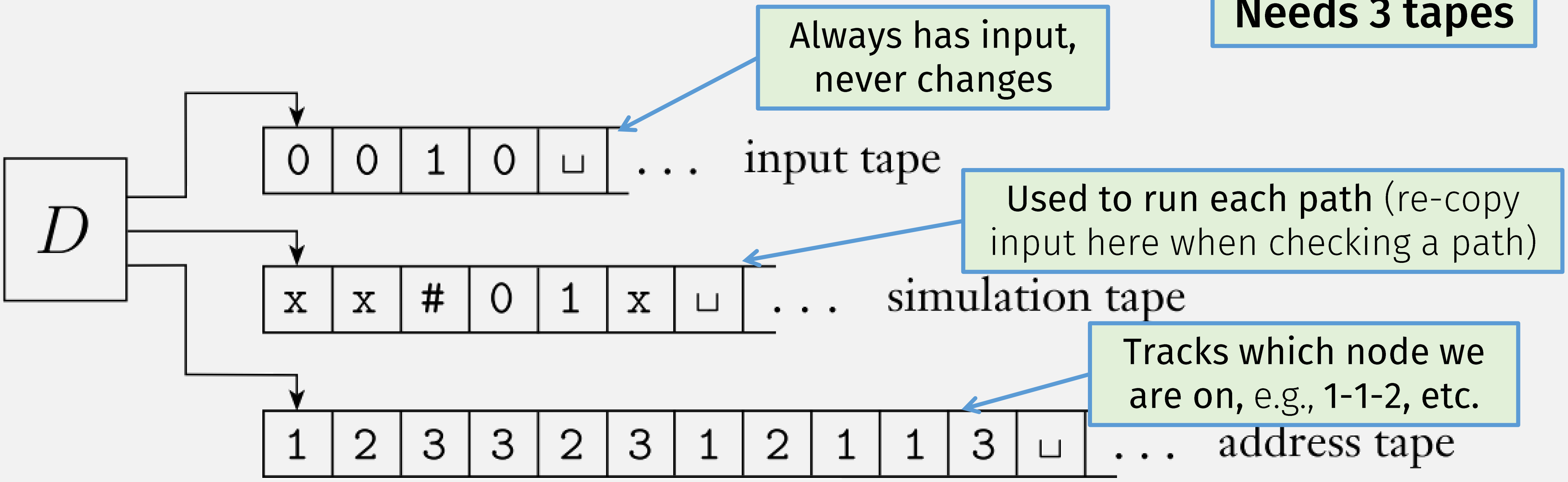
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Deterministically check every tree path, in breadth-first order
    - 1
    - 1-1
    - 1-2
    - **1-1-1**



Nondeterministic TM → Deterministic

2nd way
(Sipser)

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 δ fn output to a one-element set**
  - (just like conversion of DFA to NFA)

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

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

Conclusion: These are All Equivalent TMs!

- Single-tape Turing Machine

- Multi-tape Turing Machine

- Non-deterministic Turing Machine

Check-in Quiz 3/2

On gradescope