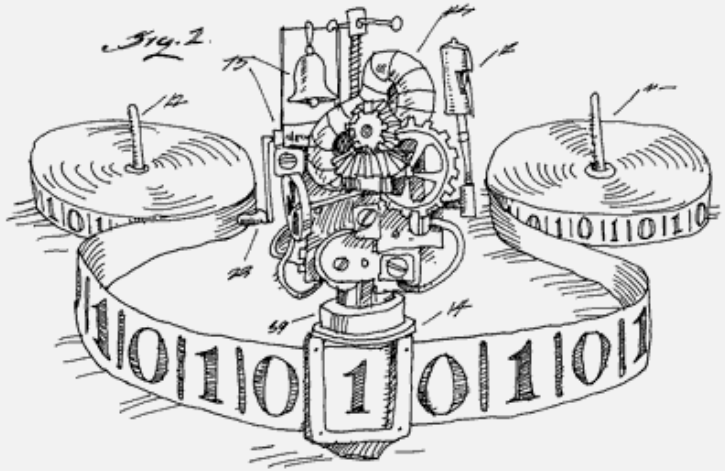
UMB CS420

Turing Machines (TMs)

Monday, March 27, 2023

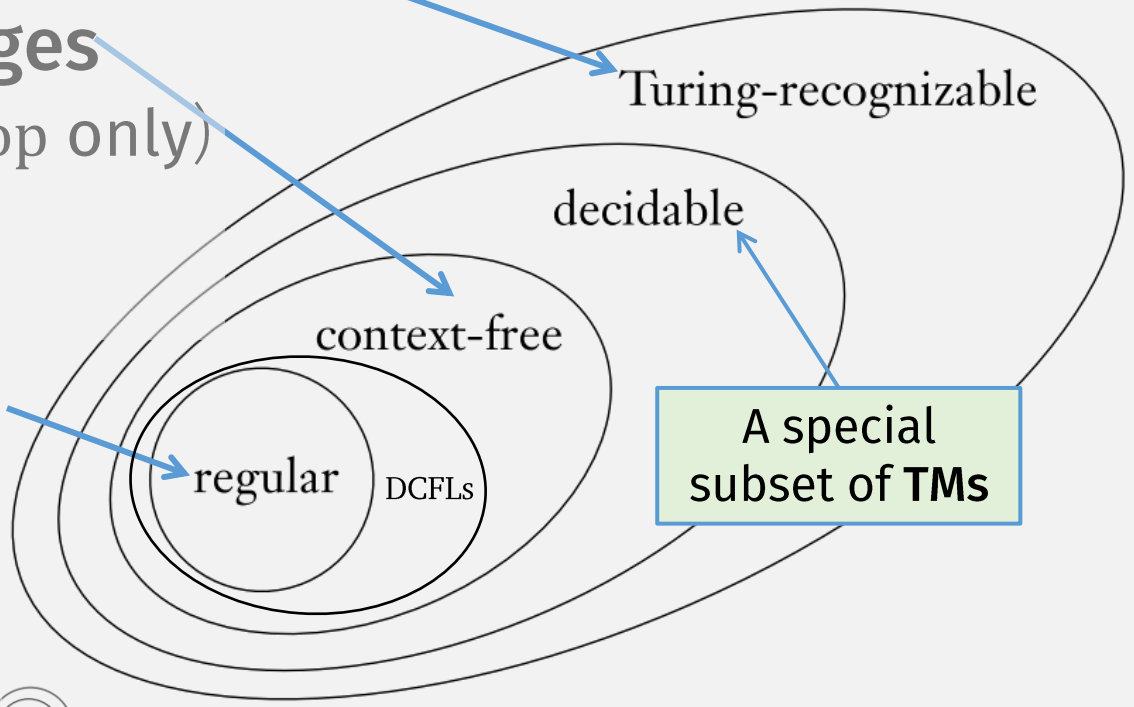


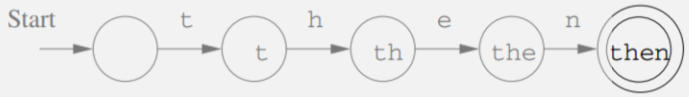
CS 420: Where We've Been, Where We're Going

- **Turing Machines** (TMs)
  - Memory: states + infinite **tape**, (arbitrary read/write)
  - Expresses any "computation"
- **PDAs**: recognize **context-free languages**
  - Memory: states + infinite **stack** (push/pop only)
  - Can't express: arbitrary dependency,
    - e.g., $\{ww \mid w \in \{0,1\}^*\}$

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

- **DFAs / NFAs**: recognize **regular langs**
  - Memory: finite states
  - Can't express: dependency
    e.g., $\{0^n 1^n \mid n \geq 0\}$

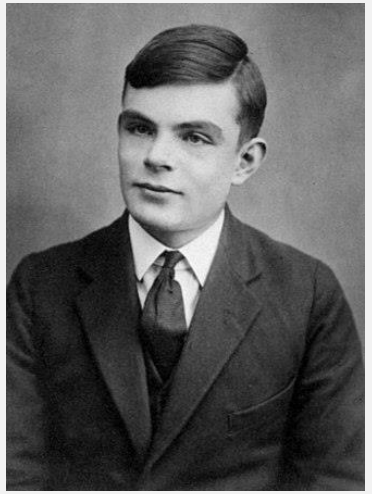




Alan Turing

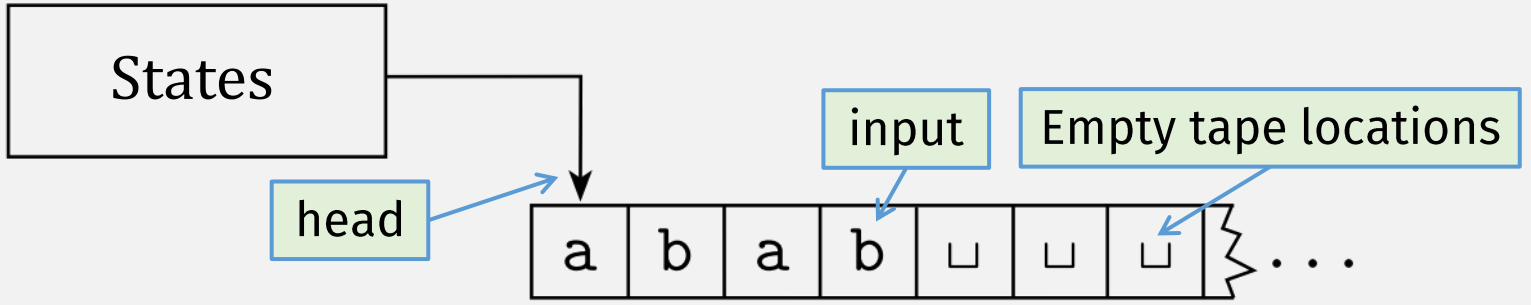
- First to formalize the models of computation we're studying
  - I.e., he invented many of the ideas in this course
- And worked as a codebreaker during WW2

- 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
- Tape is infinite
  - To the right



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

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

Turing Machine Example

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

Example input

tape

head

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.

High-level: "Cross off"
δ: write "x" char

This is a **high-level TM description**

It is **equivalent** to (but more concise than) our typical (low-level) tuple descriptions, i.e., one step = maybe multiple δ transitions

Analogy
"High-level": `Python`
"Low-level": `assembly language`

Turing Machine Example

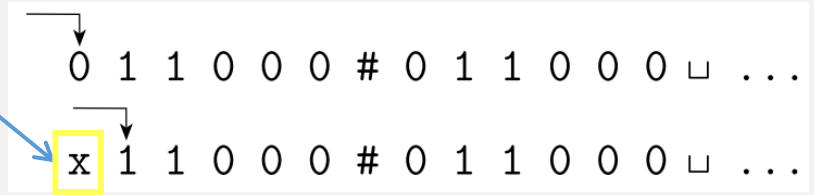
M_1 accepts inputs in language $B = \{w\#w \mid w \in \{0,1\}^*\}$

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

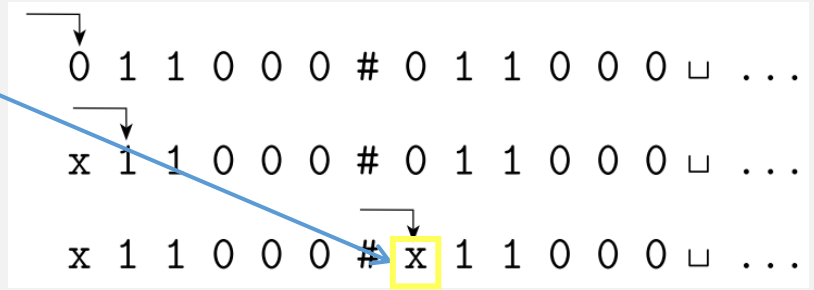
M_1 accepts inputs in language $B = \{w\#w \mid w \in \{0,1\}^*\}$

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

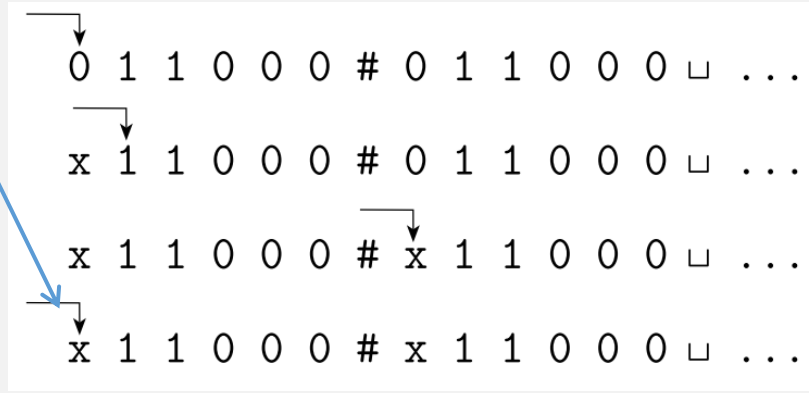
M_1 accepts inputs in language $B = \{w\#w \mid w \in \{0,1\}^*\}$

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.

"Cross off" = write "x" char

Head "zags" back to start



Turing Machine Example

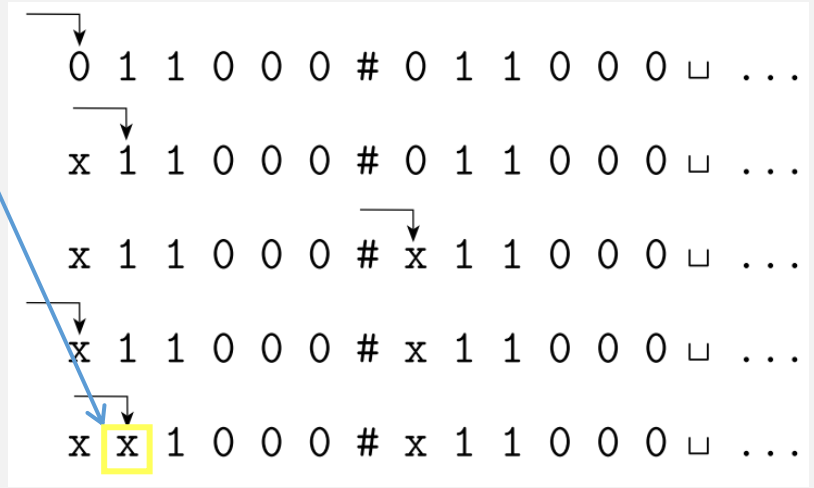
M_1 accepts inputs in language $B = \{w\#w \mid w \in \{0,1\}^*\}$

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



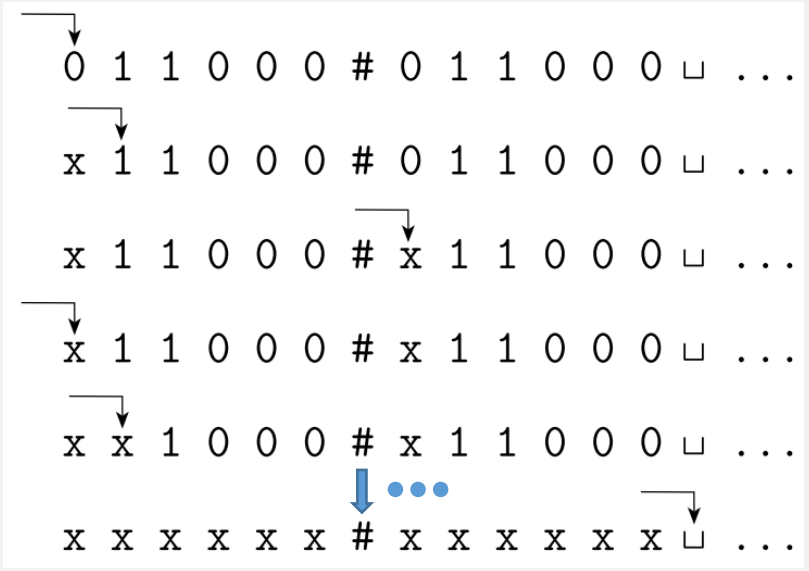
Turing Machine Example

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

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*."



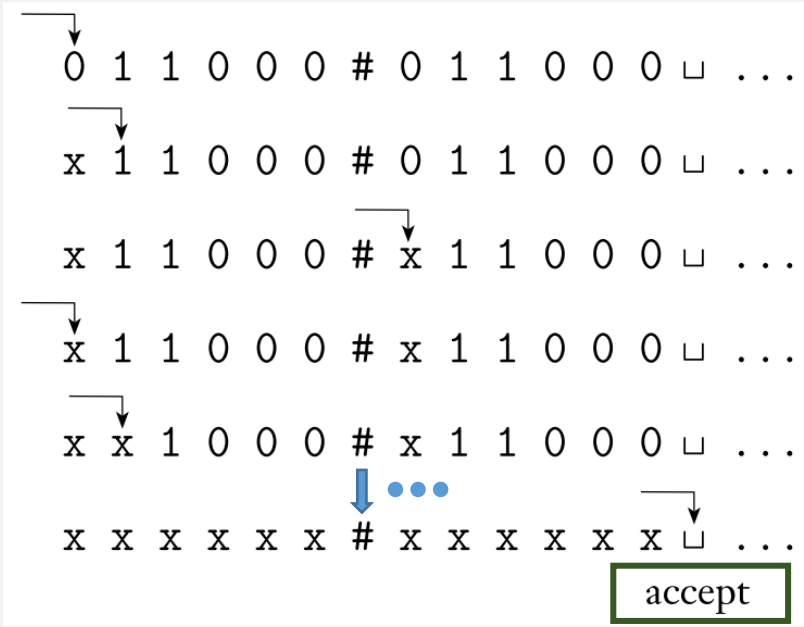
Turing Machine Example

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

This is a "**low-level**" TM description

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

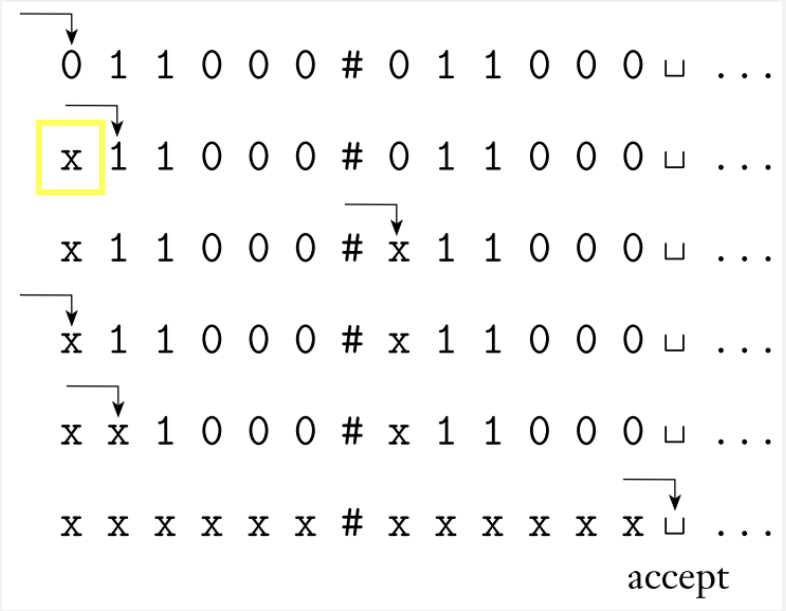
1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,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

read write move

Is this machine **deterministic**?
Or **non-deterministic**?

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

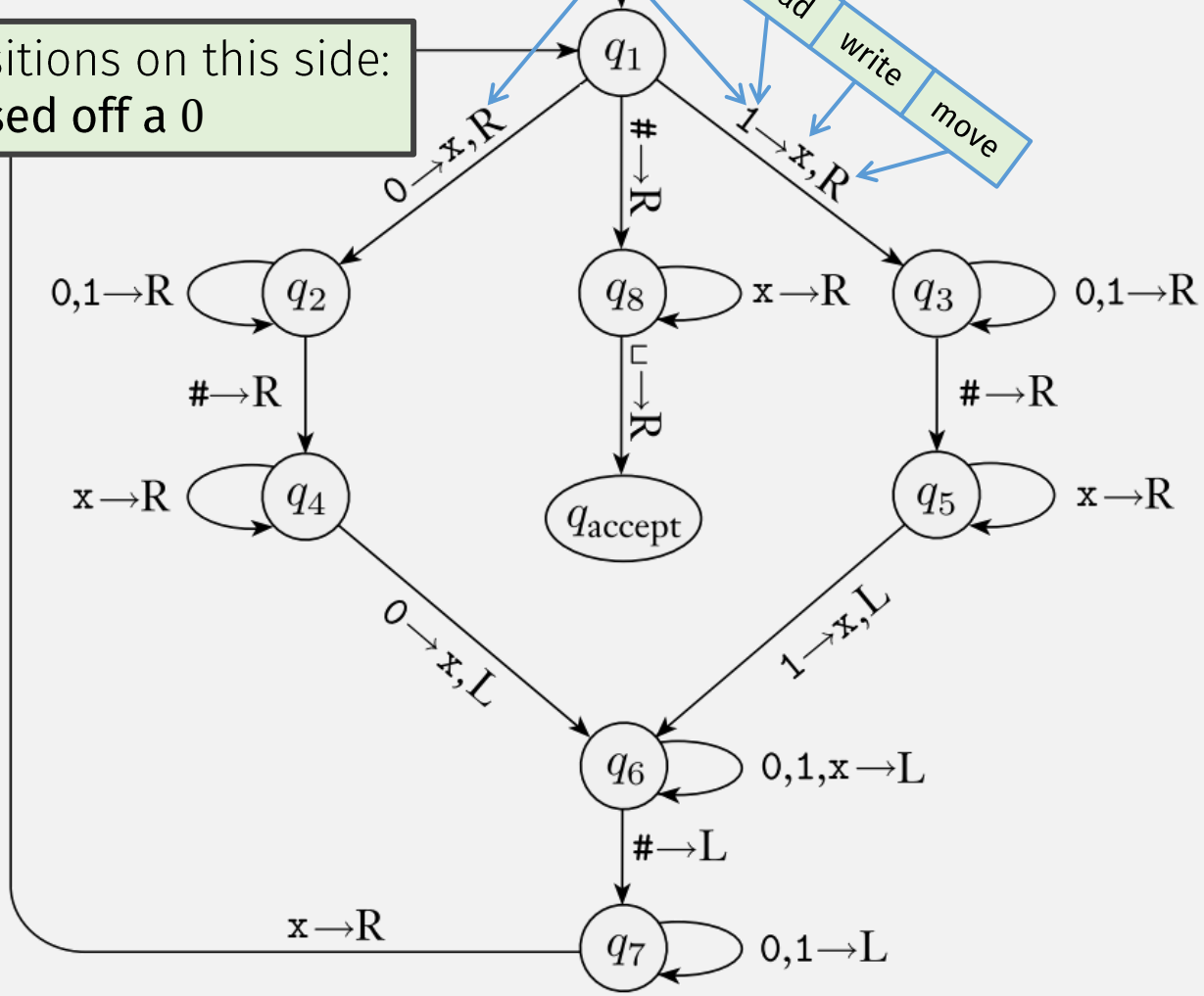
Formal Turing Machine Example



Read char (0 or 1), cross it off, move head R(ight)

Transitions on this side:
Crossed off a 0

read | write | move

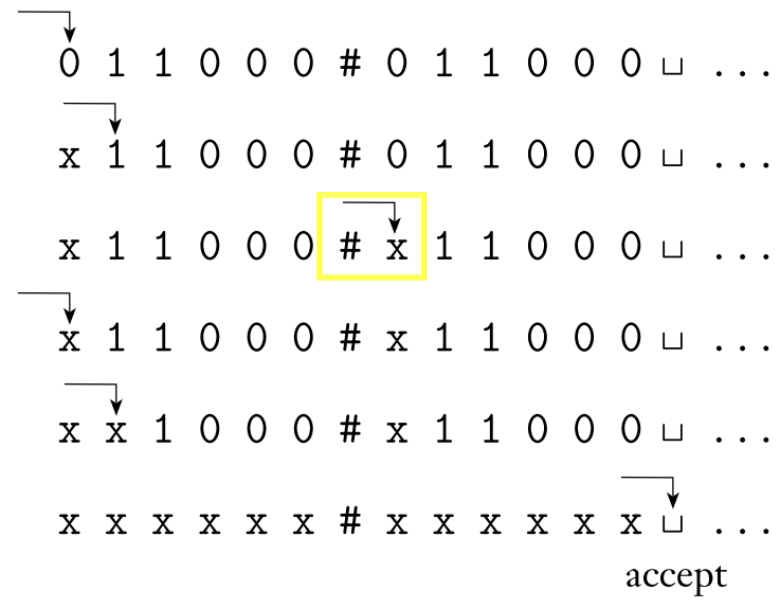


A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

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

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

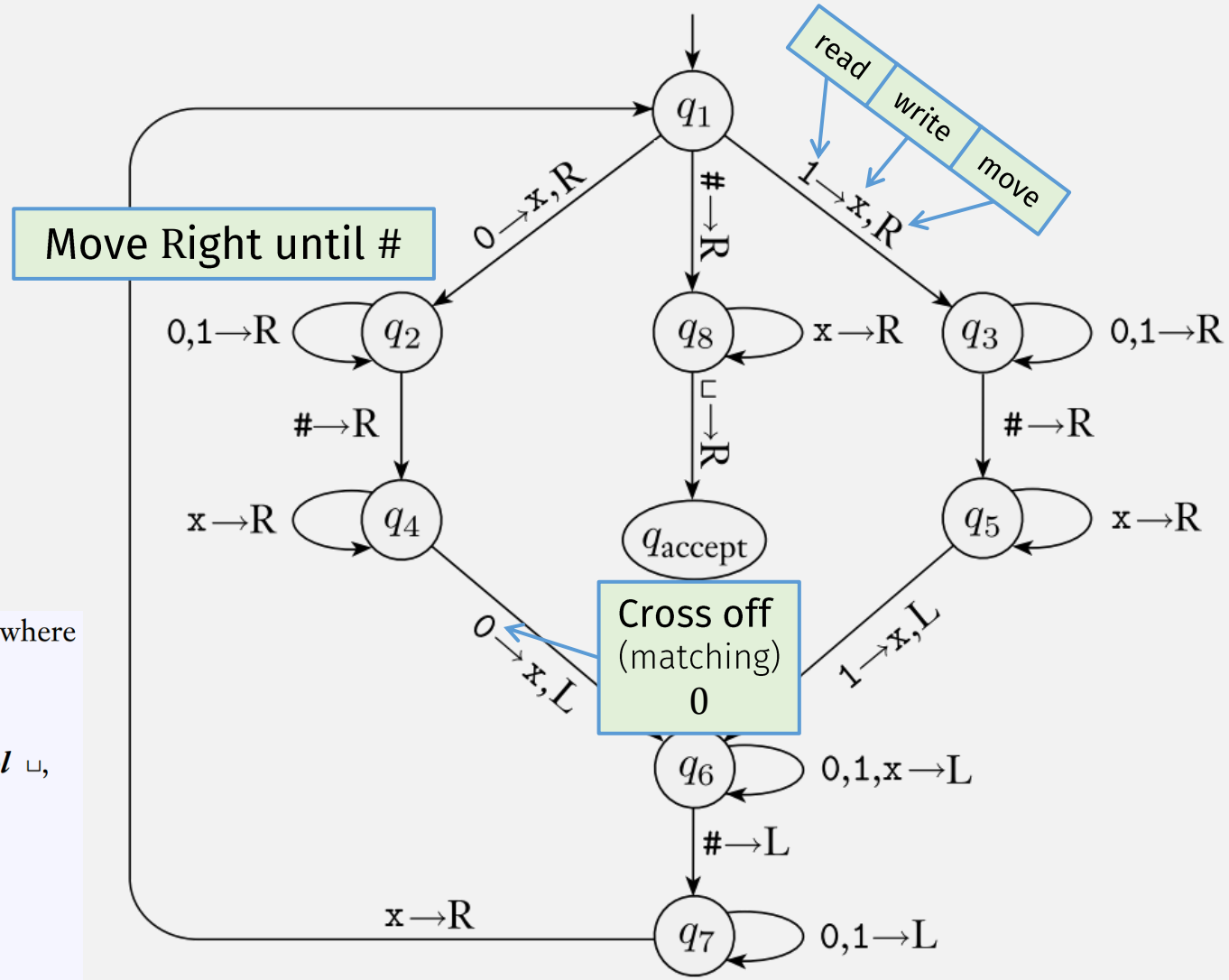
Formal Turing Machine Example



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

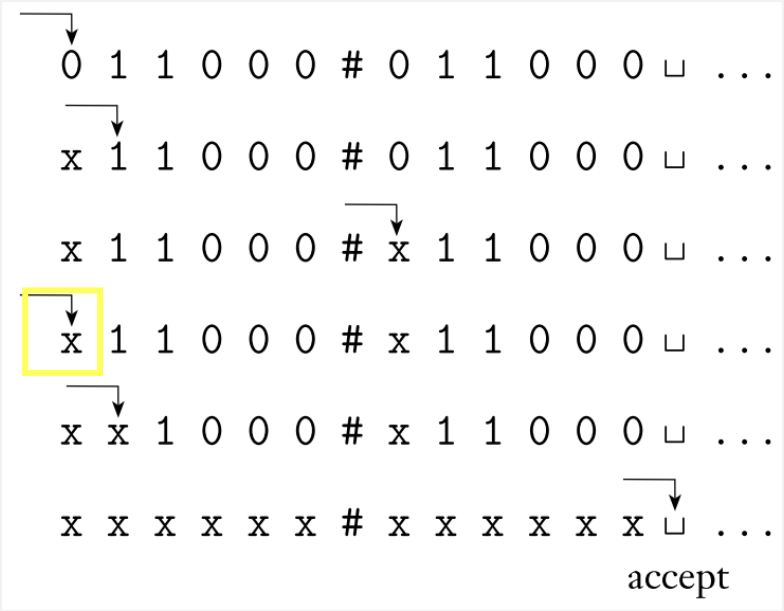
1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,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

read | write | move



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

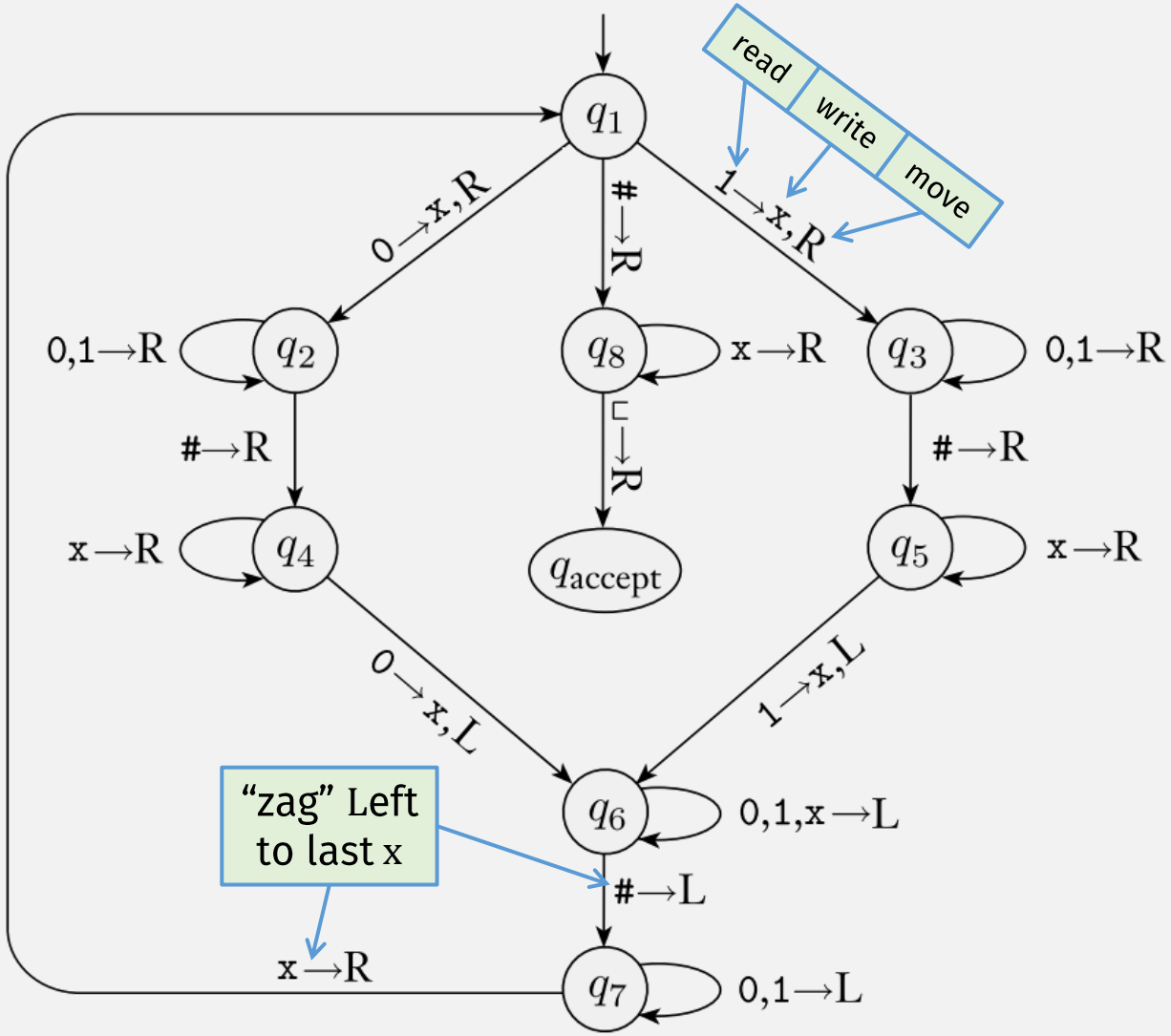
Formal Turing Machine Example



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

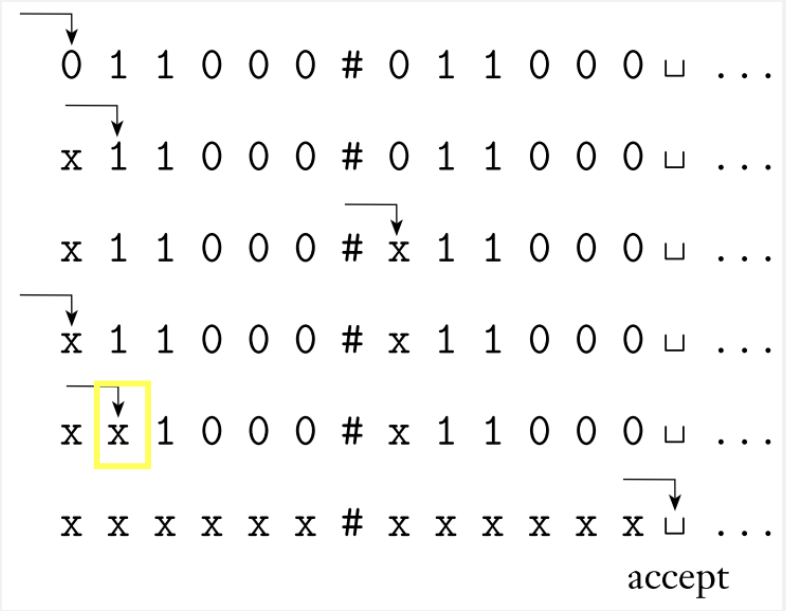
1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,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

read | write | move



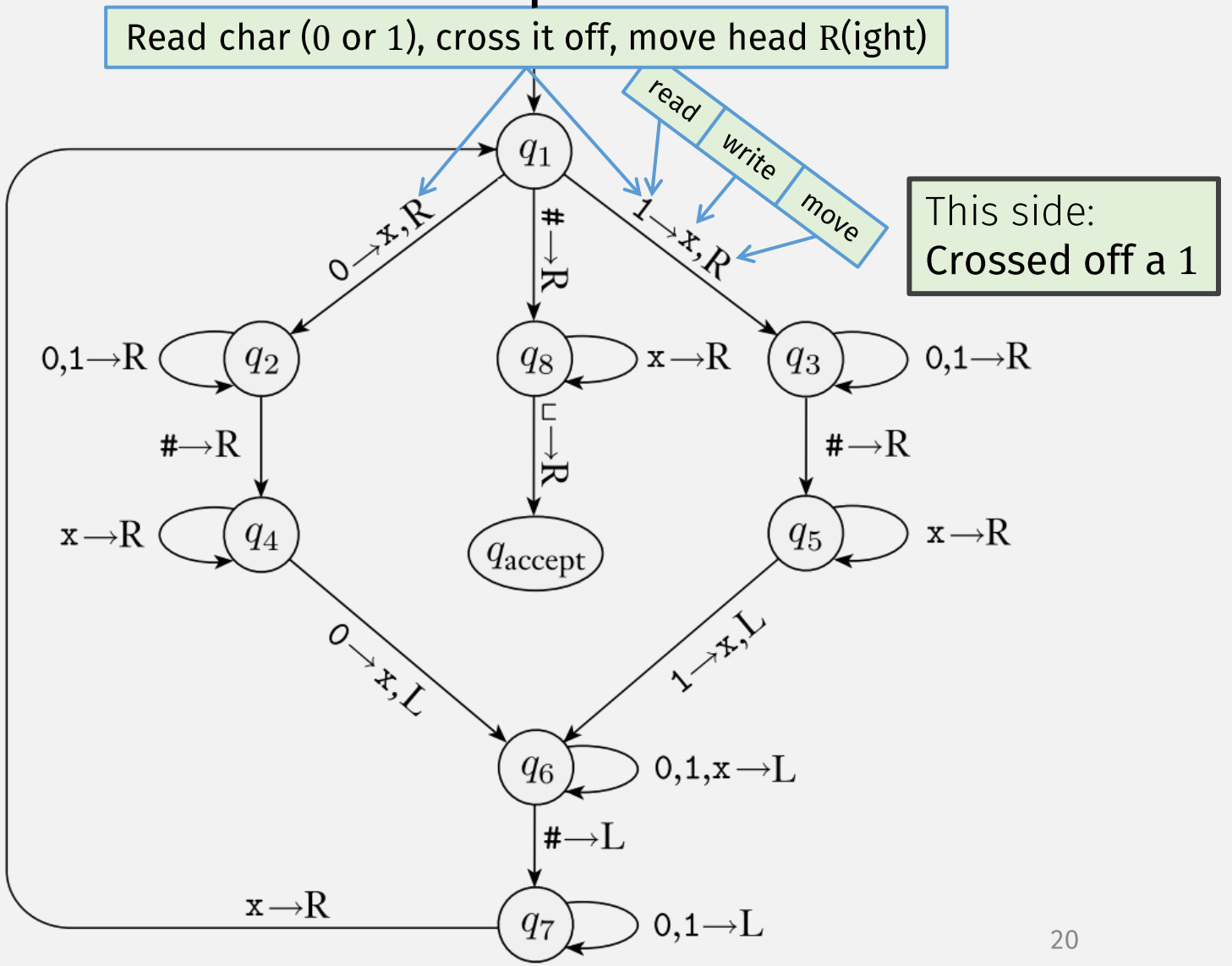
$$B = \{w\#w \mid w \in \{0,1\}^*\}$$

Formal Turing Machine Example



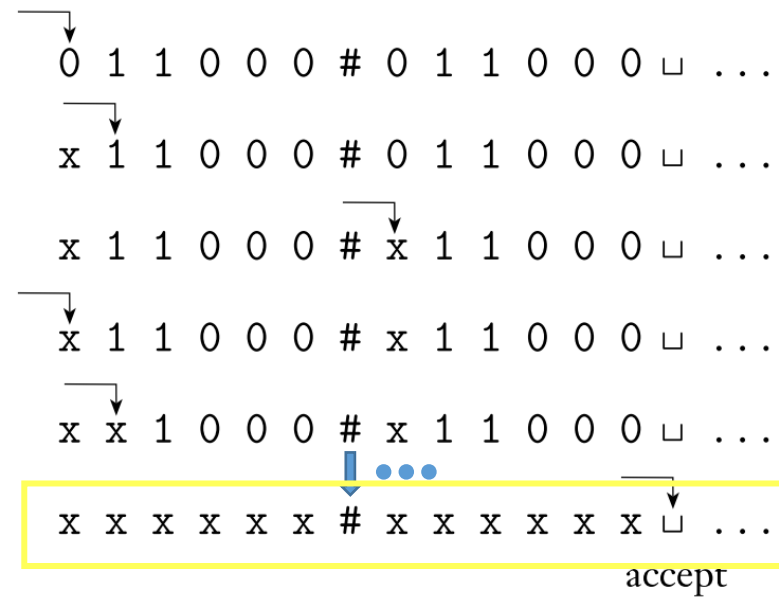
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

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



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

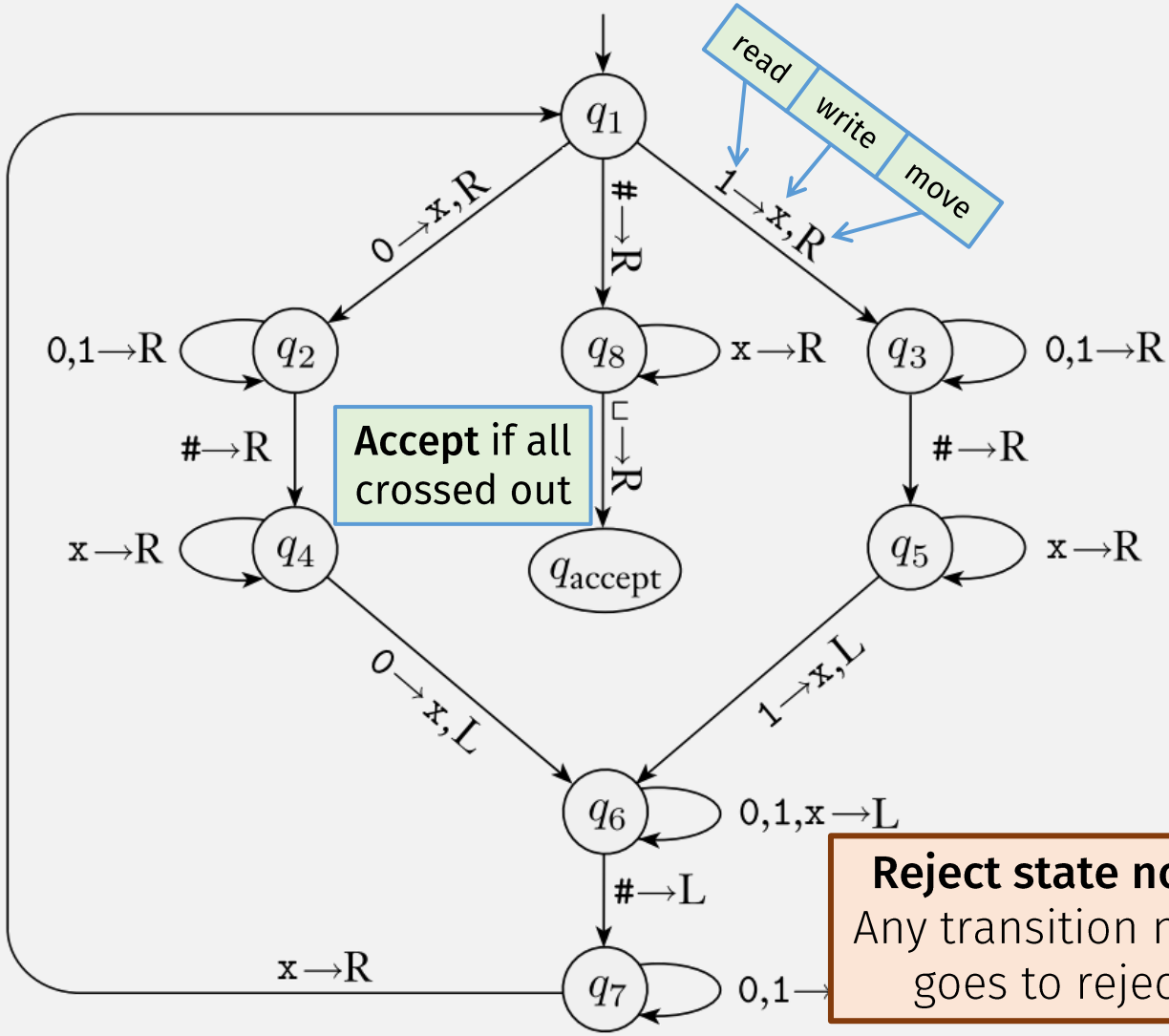
Formal Turing Machine Example



A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, 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. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,
5. $q_0 \in$ e s
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

read | write | move

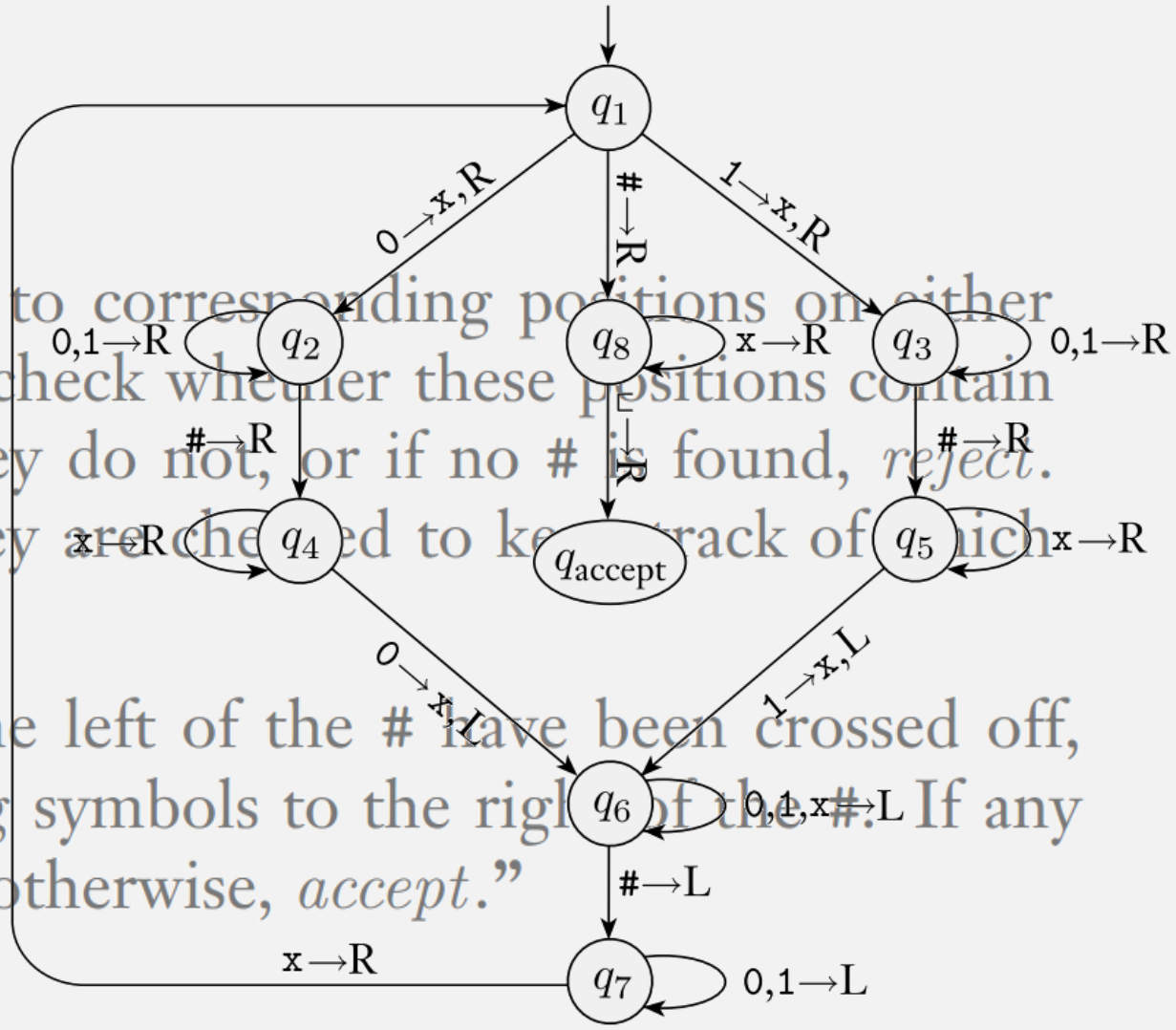


Reject state not shown
Any transition not shown goes to reject state

TMs: High-level vs Low-level?

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: High-level Description

- M_1 accepts if input is 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 no[illegible] if no # is found, *reject*. Cross off symbols as the[illegible] track of which symbols correspond.
2. When all symbols to[illegible] crossed off, check for any remaining s[illegible] of the #. If any symbols remain, *reject*; otherwise, *accept*."

We will (mostly) stick to **high-level** descriptions of Turing machines, like this one

TM High-level Description Tips

Analogy:

- **High-level** TM description **~** function definition in **"high level" language,** e.g. **Python**
- Low-level TM tuple **~** function definition in **bytecode** or **assembly**

TM high-level descriptions are not a **"do whatever" card,** some rules:

1. All TMs must have a name, e.g., M
2. Input strings must also be named (like a **function parameter**), e.g., w
3. TMs can **"call" other TMs, by their name, if they pass appropriate arguments** (like function calls)
   - e.g., a step for a TM M can say: "call TM M_2 with argument string w, if M_2 accepts w then …, else …"
4. Other variables must also be defined (named) **before they are used**
   - e.g., can define a TM inside another TM
5. Follow typical programming "scoping" rules
   - can assume other functions we've already defined are in the "global" scope, `RE2NFA`, etc
6. must be **equivalent** to a low-level formal tuple
   - high-level "step" represents a finite # of low-level δ transitions
   - So one step cannot run forever
   - E.g., can't say "try all numbers" as a "step"

Non-halting Turing Machines (TMs)

IMPORTANT SLIDE

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

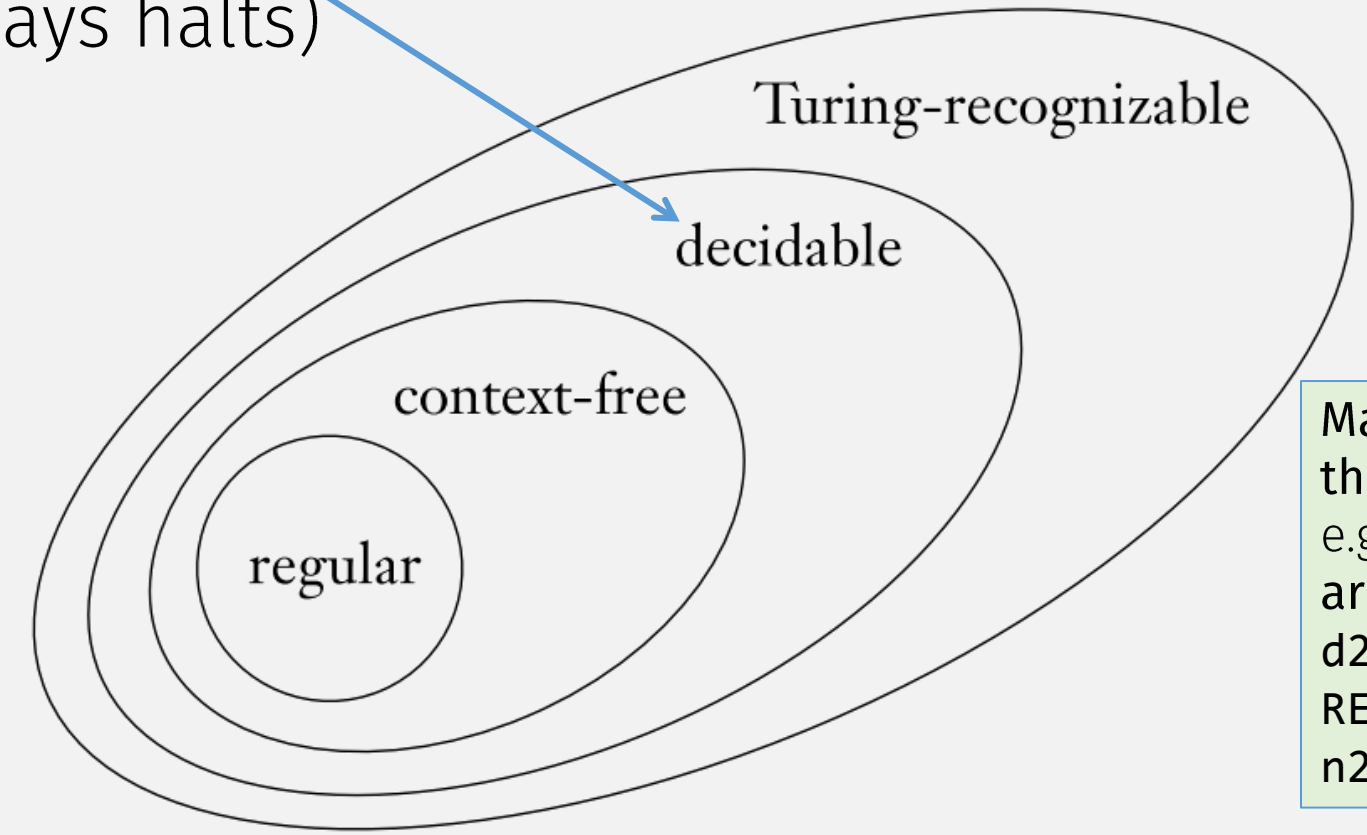
- Thus, there are two classes of Turing Machines:
  - A **recognizer** is a Turing Machine that may run forever (all possible TMs)
  - A **decider** is a Turing Machine that always halts.

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.

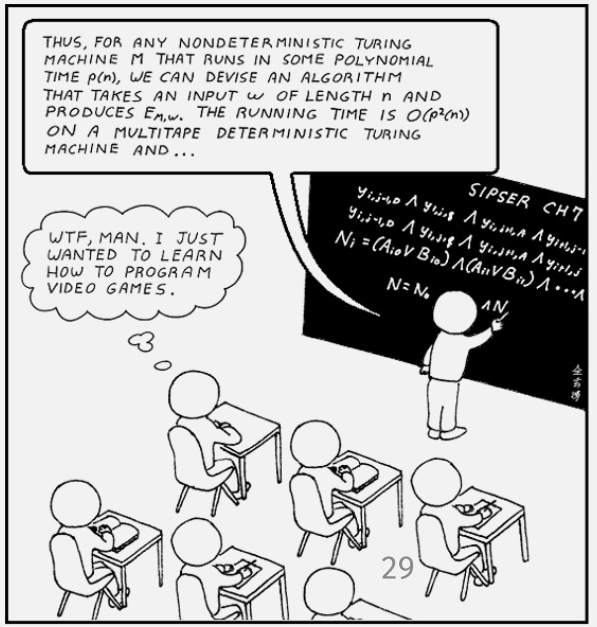
Formal Definition of an “Algorithm”

- An **algorithm** is equivalent to a **Turing-decidable** Language (always halts)

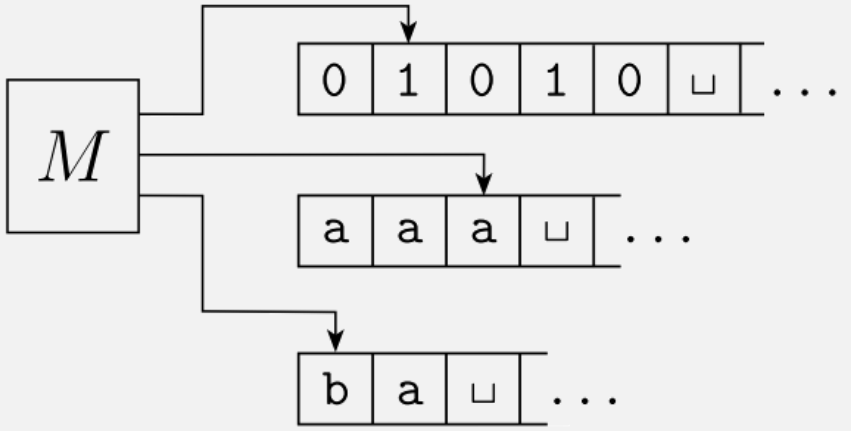


Many <u>functions</u> we have defined this semester are **algorithms**!
e.g., all our conversion functions are **deciders**!!
d2n
RE2NFA
n2p

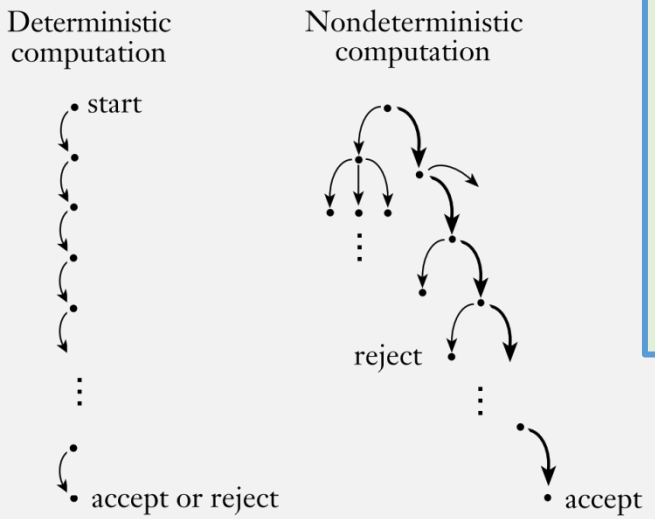
Turing Machine Variations



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

- Single-tape TM is equivalent to …
- … multi-tape TM that only uses one of its tapes
- (could you write out the formal conversion?)

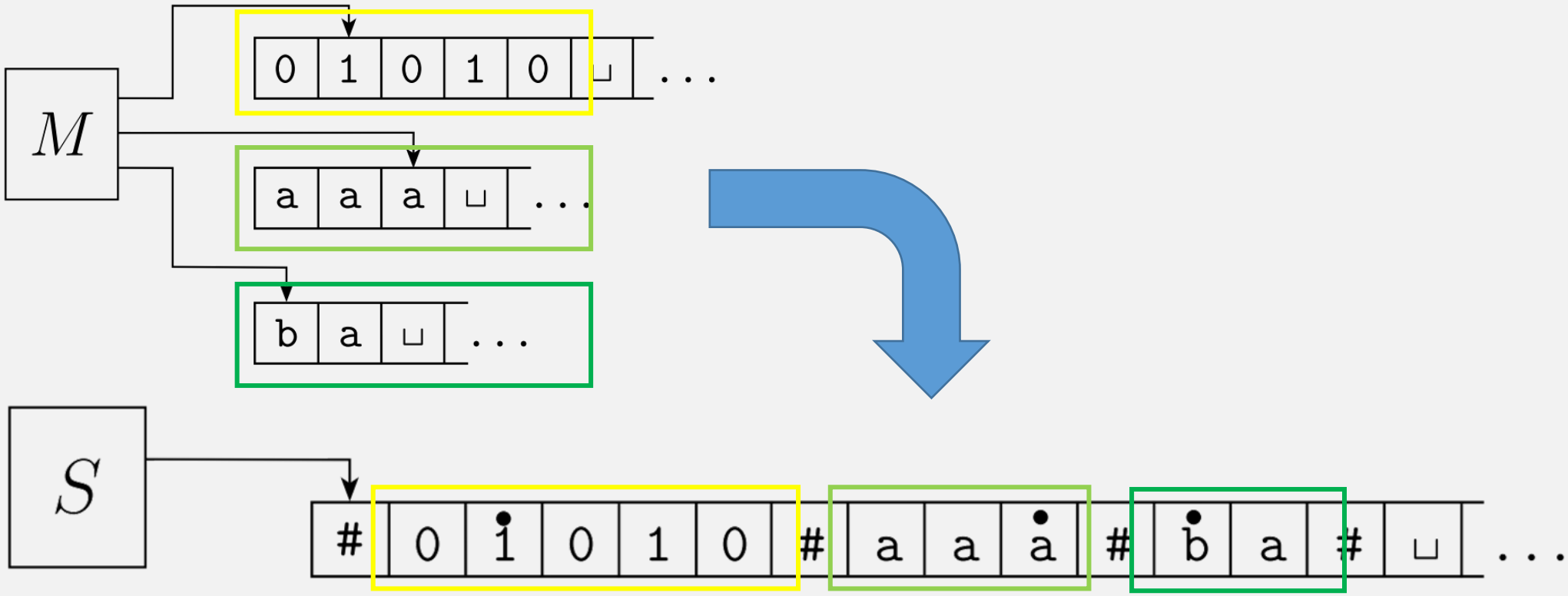
⇐ If a multi-tape TM recognizes a language,
then a single-tape TM recognizes the language

- Convert: multi-tape TM → 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

- Single-tape TM is equivalent to …
- … 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

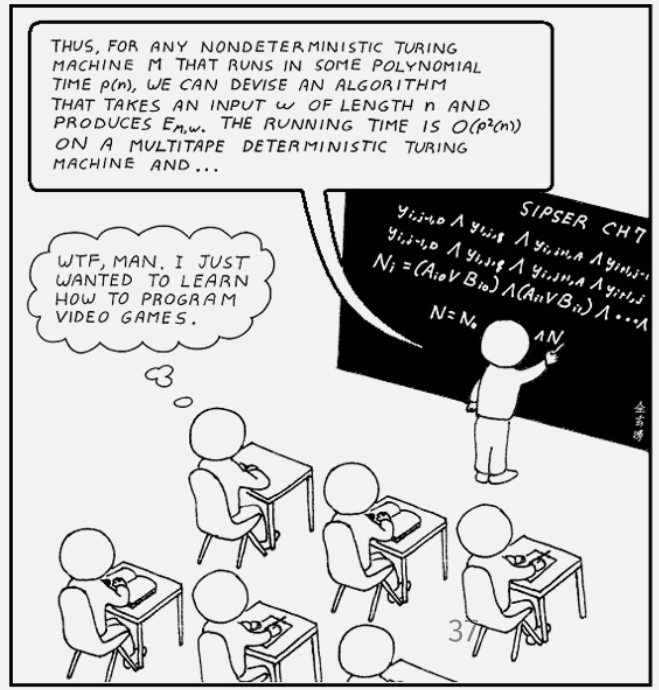
- Convert: multi-tape TM → single-tape TM

■

Check-in Quiz 3/27

On gradescope

Nondeterministic TMs



Turing Machines: Formal Tuple 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. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ 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 $q_{\text{reject}} \neq q_{\text{accept}}$.

read write move

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

Nondeterministic transition produces set of possible next states

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.

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. $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ 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

Nondeterministic Turing Machine

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

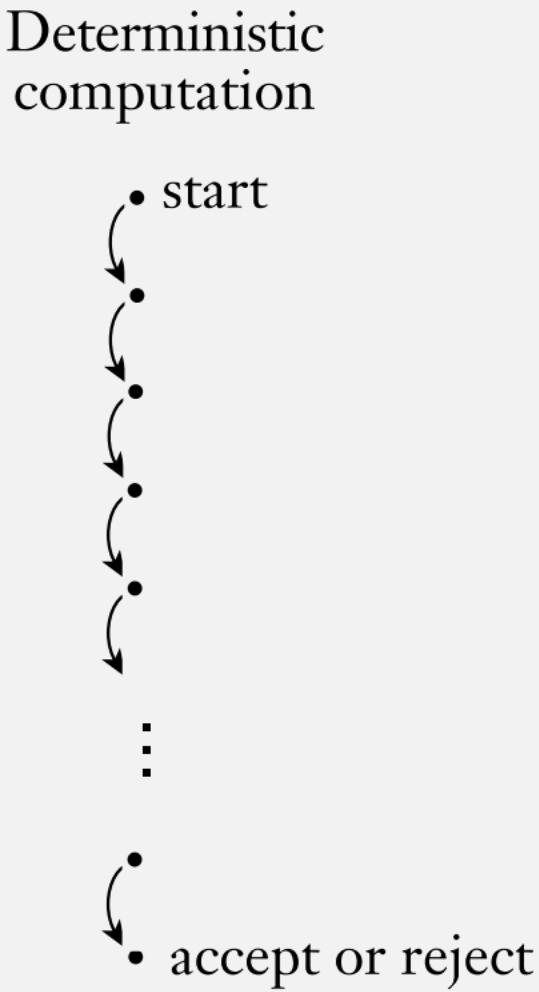
⇒ If a deterministic TM recognizes a language,
then a non-deterministic TM recognizes the language

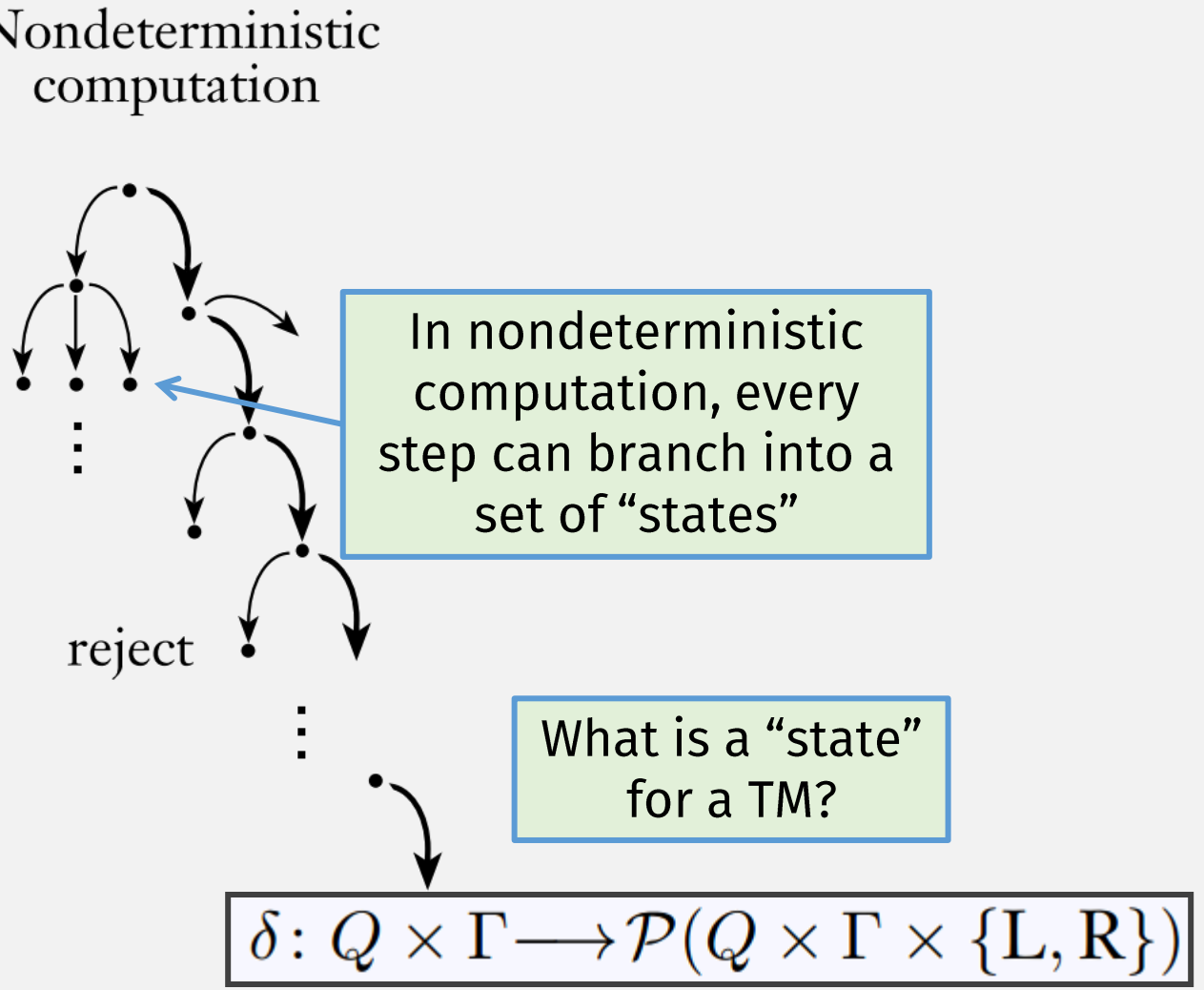
- Convert: Deterministic TM → Non-deterministic TM …
- … change Deterministic TM δ fn output to a one-element set
  - (just like **d2n** conversion of DFA to NFA --- HW 2, Problem 2)
- **DONE!**

⇐ If a non-deterministic TM recognizes a language,
then a deterministic TM recognizes the language

- Convert: Non-deterministic TM → Deterministic TM …
- … ???

Review: Nondeterminism





Flashback: PDA Configurations (IDs)

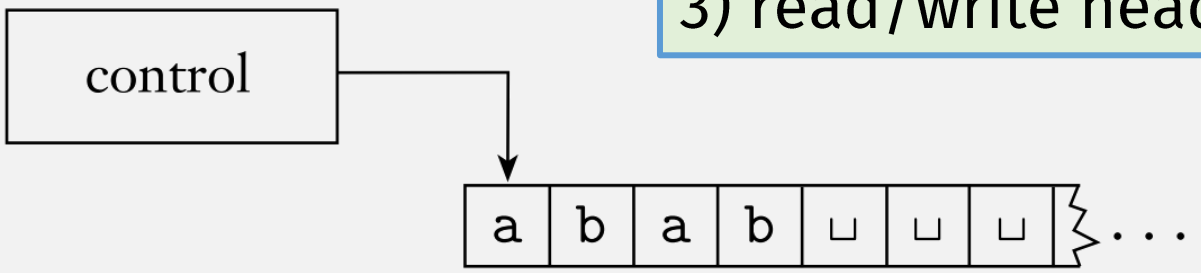
- A **configuration** (or **ID**) is a "snapshot" of a PDA's computation

- 3 components (q, w, γ) :
  q = the **current state**
  w = the **remaining input string**
  γ = the **stack contents**

A **sequence of configurations** represents a **PDA** computation

TM Configuration (ID) = ???

1) states



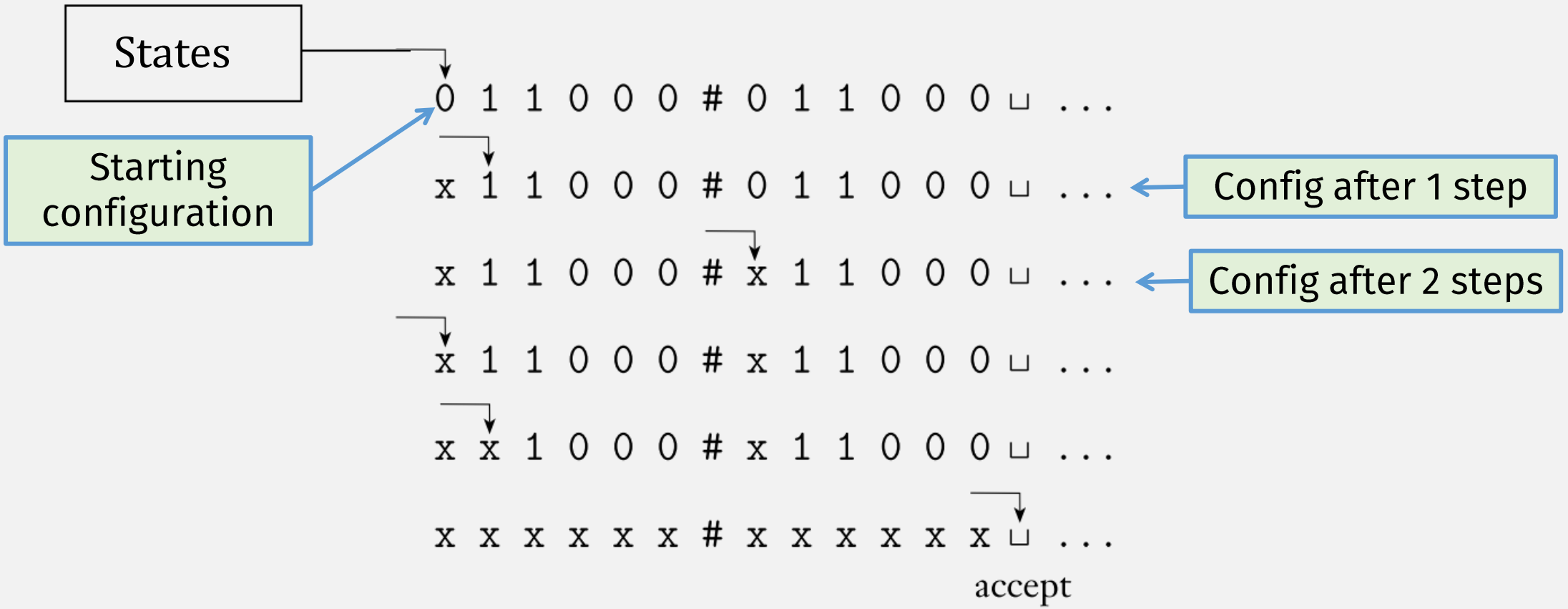
3) read/write head

2) Tape contents

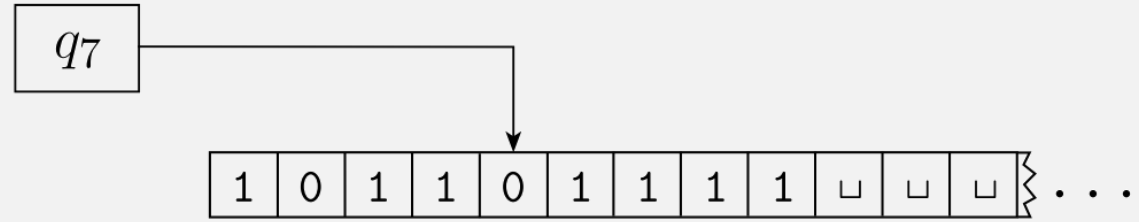
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

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

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

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

head

Next config

write

read

if $q_1, q_2 \in Q$

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

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

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

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

Extended

- Base Case

$I \vdash^* I$ for any ID I

- Recursive Case

$I \vdash^* J$ if there exists some ID K such that $I \vdash K$ and $K \vdash^* J$

Edge cases: $q_1 \mathbf{a} \beta \vdash q_2 \mathbf{x} \beta$

Head stays at leftmost cell

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

(L move, when already at leftmost cell)

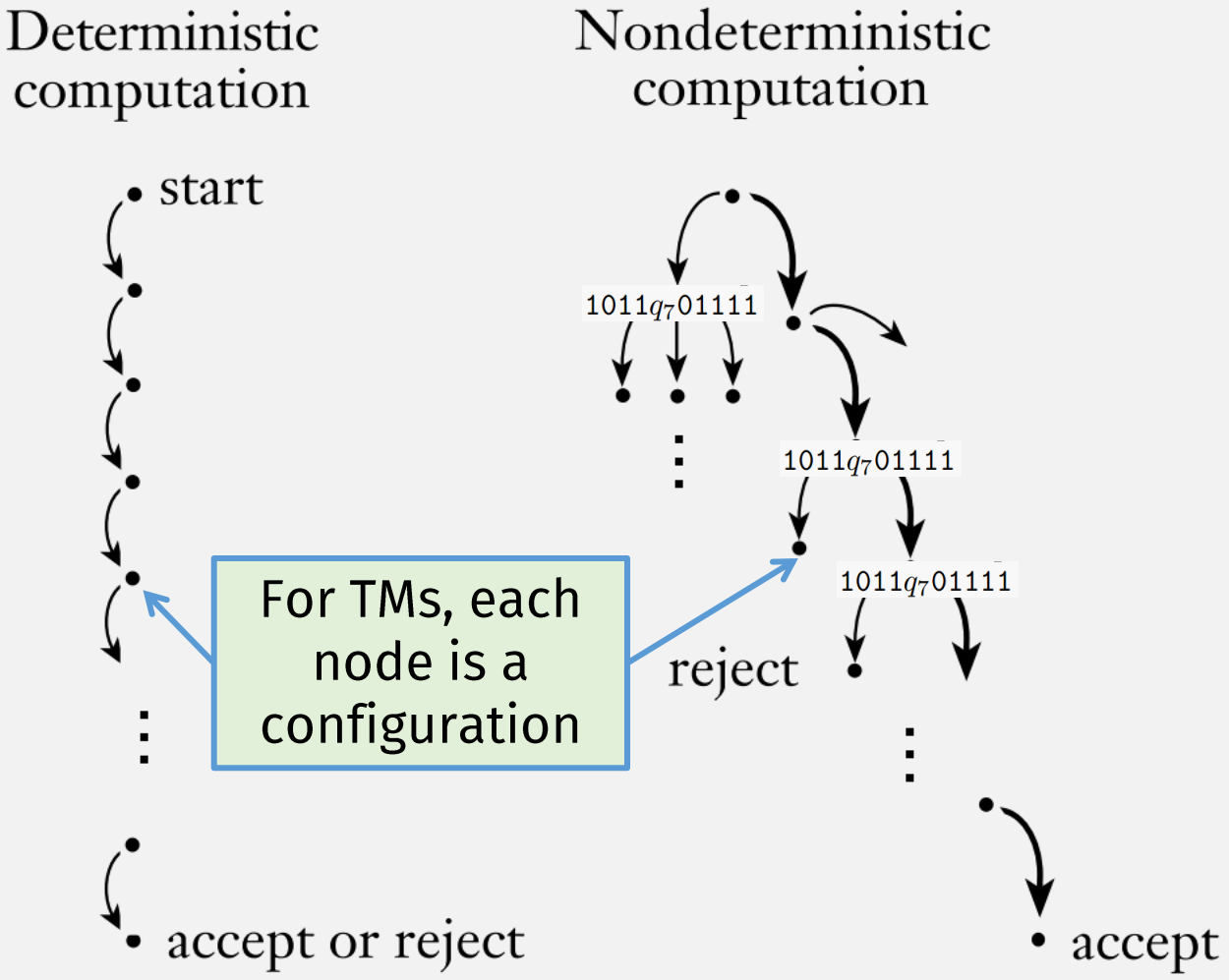
$\alpha q_1 \vdash \alpha \sqcup q_2$

Add blank symbol to config

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

(R move, when at rightmost filled cell)

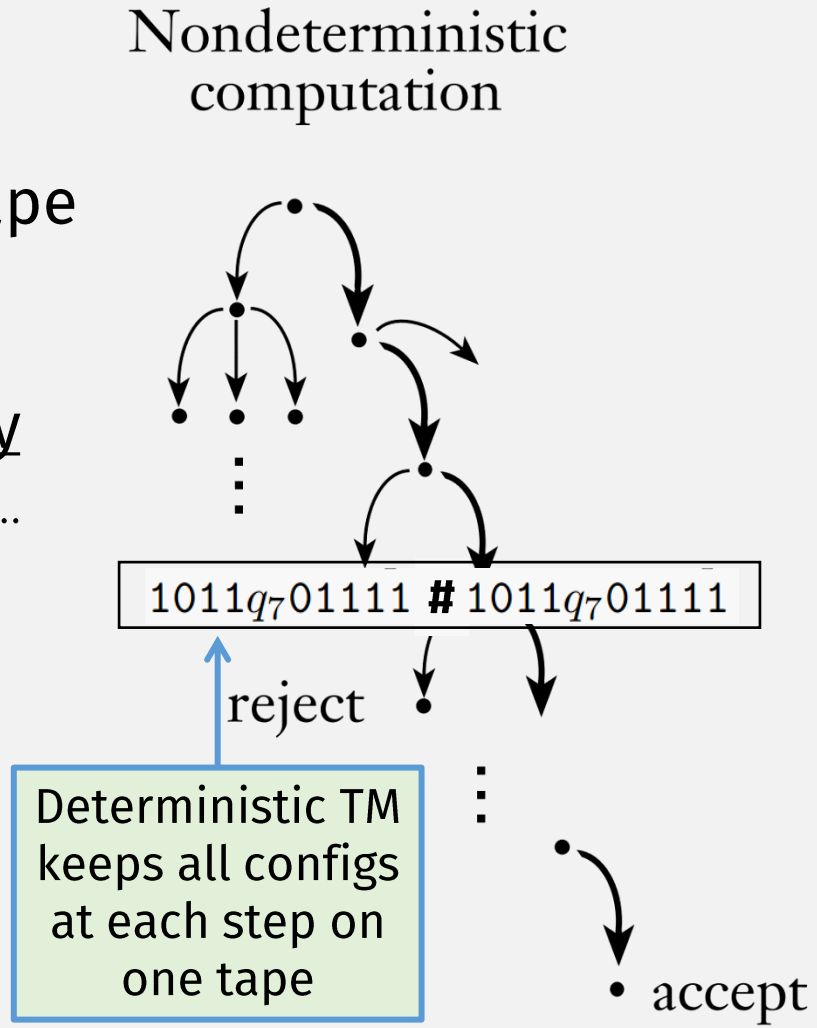
Nondeterminism in TMs



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 computations, concurrently
    - 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 concurrently?



Interlude: Running TMs inside other TMs

If **TMs are function definitions,** then **they can be *called* like functions …**

Exercise:

- Given: TMs M_1 and M_2
- Create: TM M that accepts if either M_1 or M_2 accept

M_1	M_2	M
reject	accept	accept
accept	reject	accept

"loop" means input string not accepted

Possible solution #1:

M = on input x,

1. `Call` M_1 with arg x; accept if M_1 accepts
2. `Call` M_2 with arg x; accept if M_2 accepts

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

If **TMs are function definitions,** then **they can be *called* like functions …**

Exercise:

- Given: TMs M_1 and M_2
- Create: TM M that accepts if either M_1 or M_2 accept

… with concurrency!

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

Possible solution #1:

M = on input x,

1. Call M_1 with arg x; accept if M_1 accepts
2. Call M_2 with arg x; accept if M_2 accepts

Possible solution #2:

M = on input x,

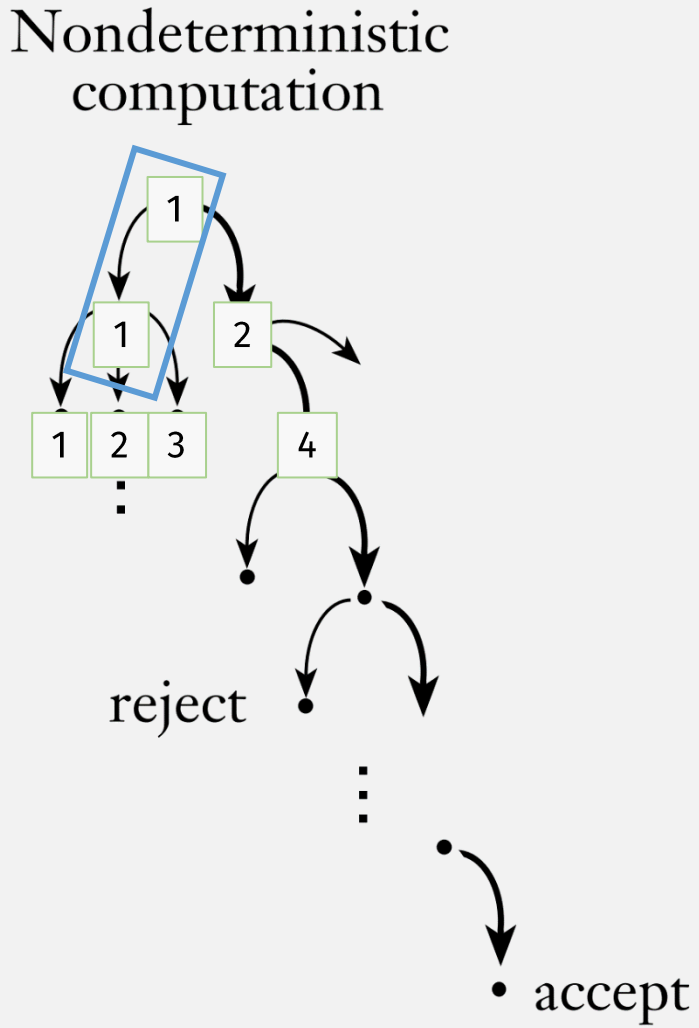
1. Call M_1 and M_2 with x concurrently, i.e.,
   a) Run M_1 with x for 1 step; accept if M_1 accepts
   b) Run M_2 with x for 1 step; accept if M_2 accepts
   c) Repeat

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

Nondeterministic TM → Deterministic

2nd way (Sipser)

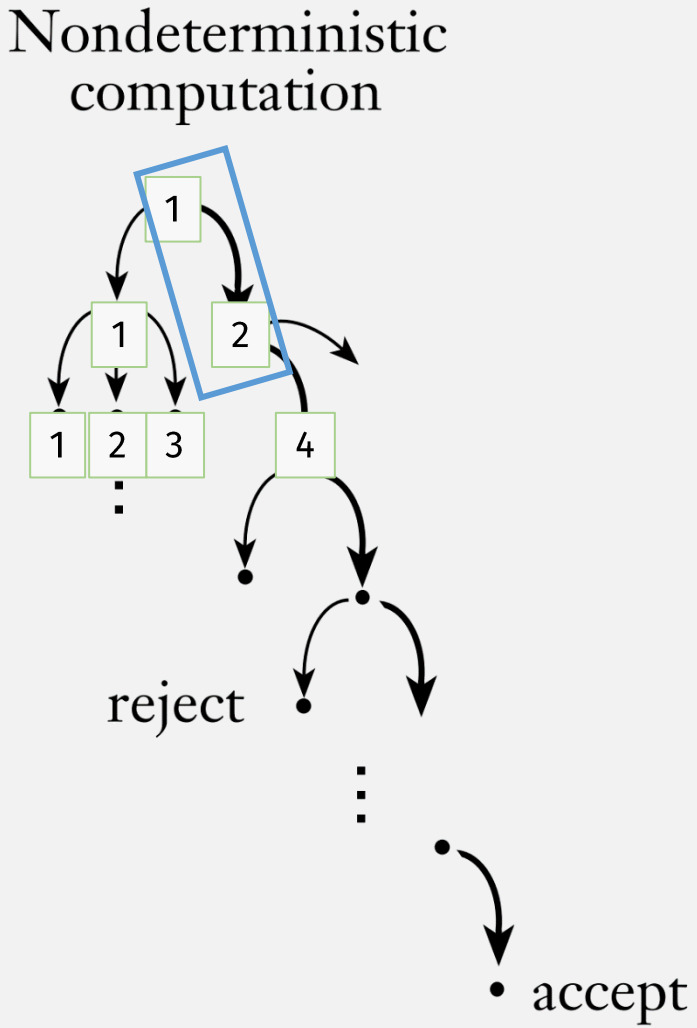
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Check all tree paths (in breadth-first order)
    - 1
    - **1-1**



Nondeterministic TM → Deterministic

2nd way (Sipser)

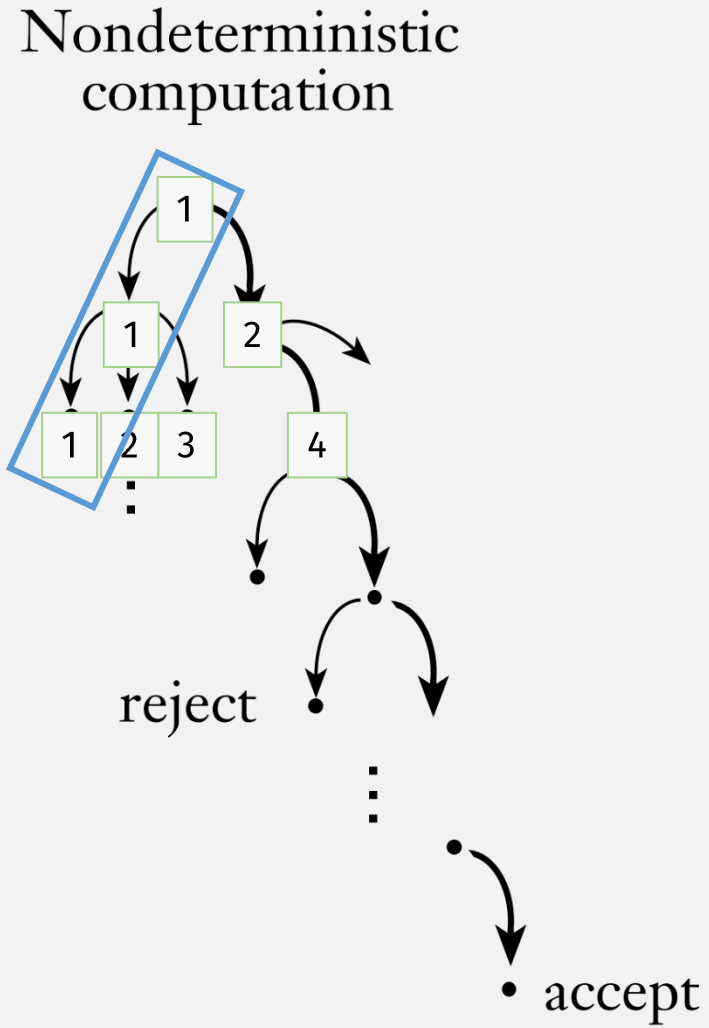
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Check all tree paths (in breadth-first order)
    - 1
    - 1-1
    - 1-2



Nondeterministic TM → Deterministic

2nd way (Sipser)

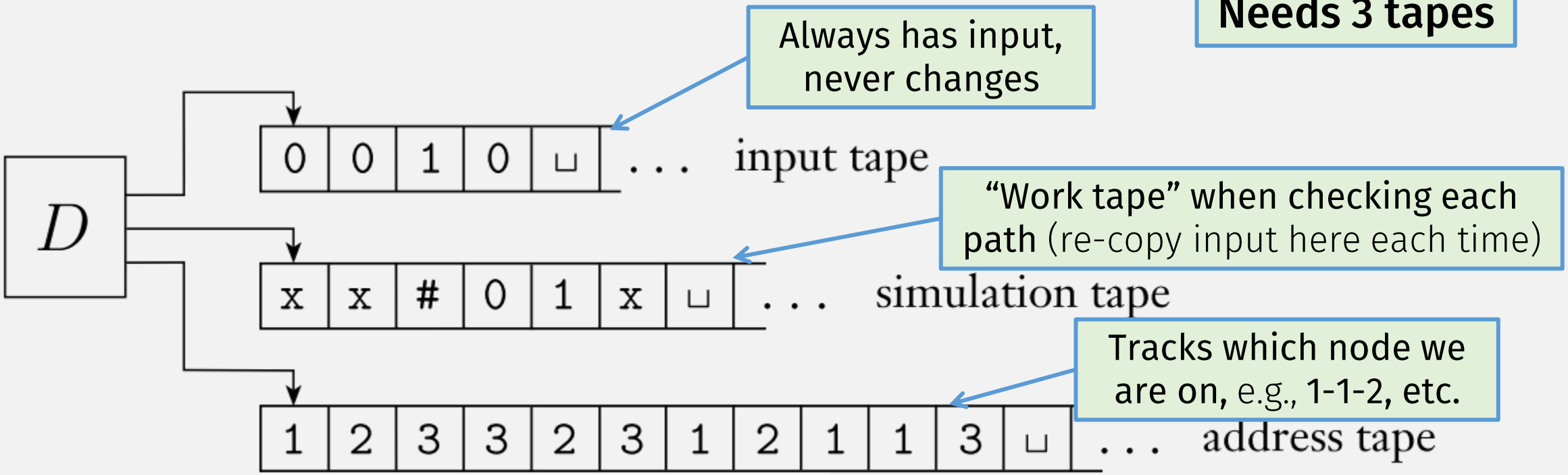
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Check all tree paths (in breadth-first order)
    - 1
    - 1-1
    - 1-2
    - **1-1-1**



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

- Convert Deterministic TM → Non-deterministic TM

☑ ⇐ 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

Turing Machines as Algorithms