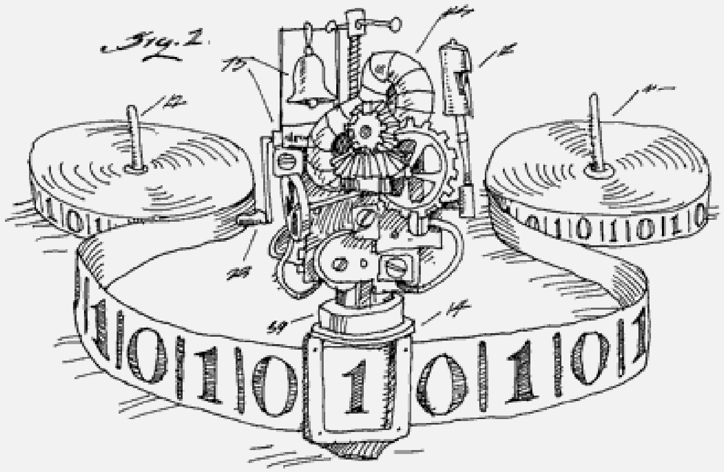
UMB CS420

Turing Machines (TMs)

Monday , April 1, 2024



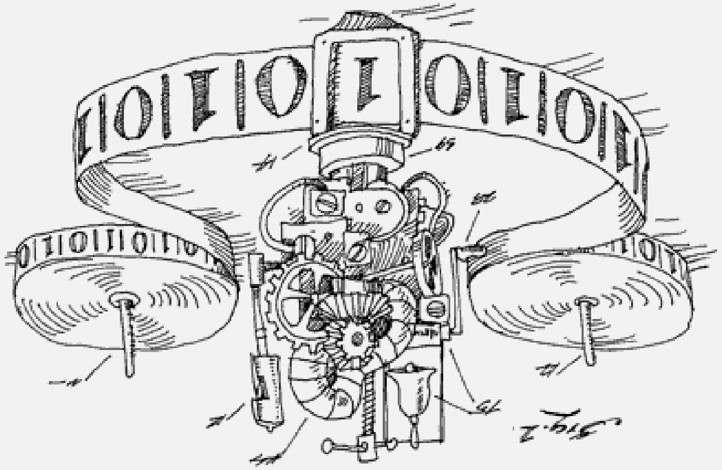
Announcements

noon Mon due 4/8 12pm

out 7 HW

12pm 4/1 noon due Mon

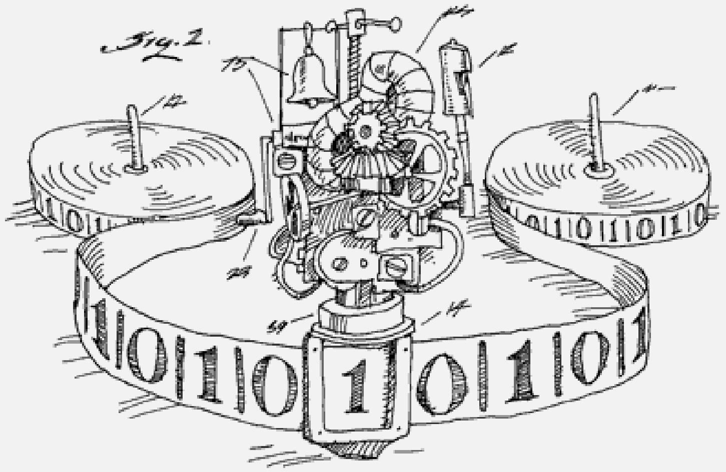
in 6 HW



Announcements

- HW 6 in
  - ~~due Mon 4/1 12pm noon~~

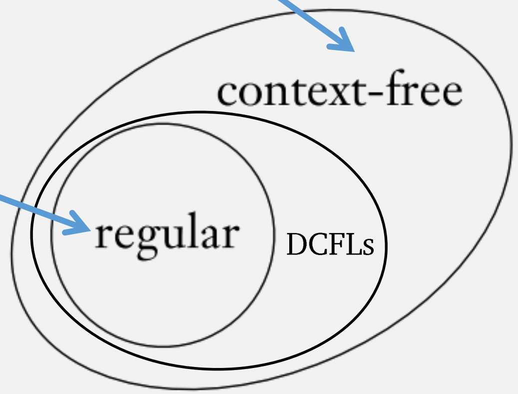
- HW 7 out
  - due Mon 4/8 12pm noon

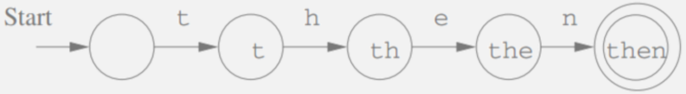


CS 420: Where We've Been, Where We're Going

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

- **PDAs**: recognize **context-free languages**
  - <u>`Memory`</u>: states + infinite **stack** (push/pop only)
  - Can't express: <u>arbitrary</u> **dependency**,
    - e.g., $\{ww \mid w \in \{0,1\}^*\}$
- **DFAs / NFAs**: recognize **regular langs**
  - <u>`Memory`</u>: finite states
  - Can't express: **dependency**
    e.g., $\{0^n1^n \mid n \geq 0\}$

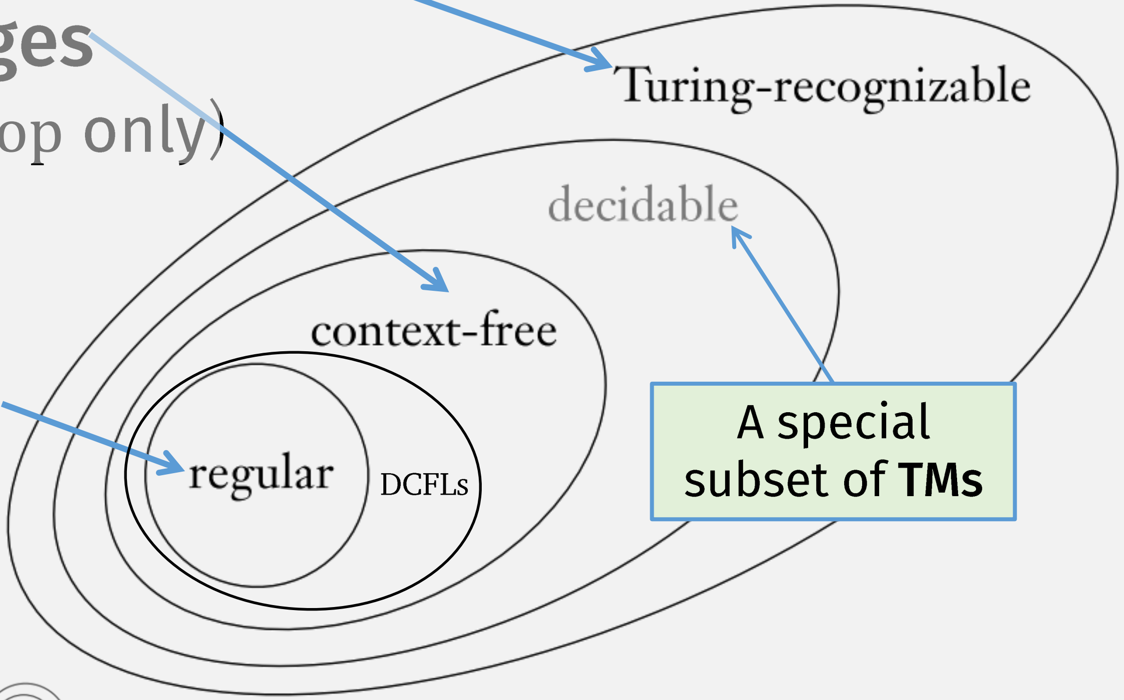




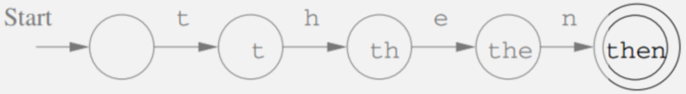
CS 420: Where We've Been, Where We're Going

- **Turing Machines** (TMs) 
  - <u>`Memory`</u>: states + infinite **tape**, (arbitrary read/write)
  - Expresses any "computation"
- **PDAs**: recognize **context-free languages**
  - <u>`Memory`</u>: states + infinite **stack** (push/pop only)
  - Can't express: <u>arbitrary</u> dependency,
    - e.g., $\{ww \mid w \in \{0,1\}^*\}$
- **DFAs / NFAs**: recognize **regular langs**
  - <u>`Memory`</u>: finite states
  - Can't express: **dependency**
    e.g., $\{0^n1^n \mid n \geq 0\}$

$A \rightarrow 0A1$
$A \rightarrow B$
$B \rightarrow \#$



A special subset of **TMs**

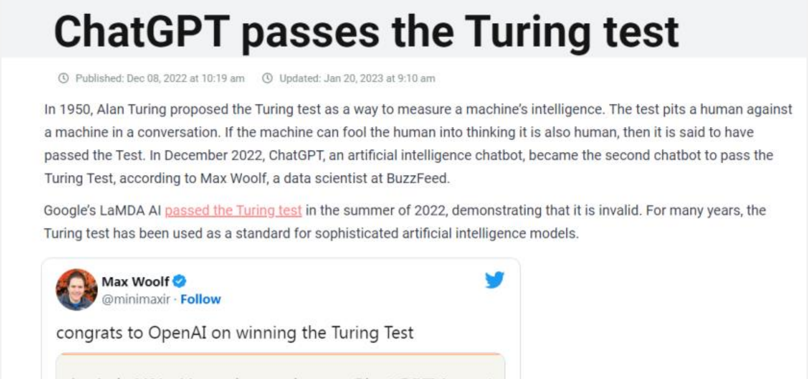


Alan Turing

- **First to formalize a model of computation**
    - 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**



ChatGPT passes the Turing test

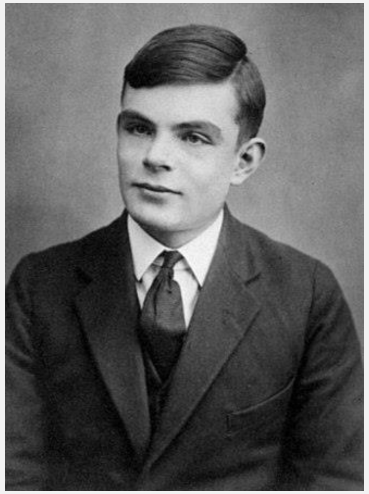
Published: Dec 08, 2022 at 10:19 am   Updated: Jan 20, 2023 at 9:10 am

In 1950, Alan Turing proposed the Turing test as a way to measure a machine's intelligence. The test pits a human against a machine in a conversation. If the machine can fool the human into thinking it is also human, then it is said to have passed the Test. In December 2022, ChatGPT, an artificial intelligence chatbot, became the second chatbot to pass the Turing Test, according to Max Woolf, a data scientist at BuzzFeed.

Google's LaMDA AI passed the Turing test in the summer of 2022, demonstrating that it is invalid. For many years, the Turing test has been used as a standard for sophisticated artificial intelligence models.

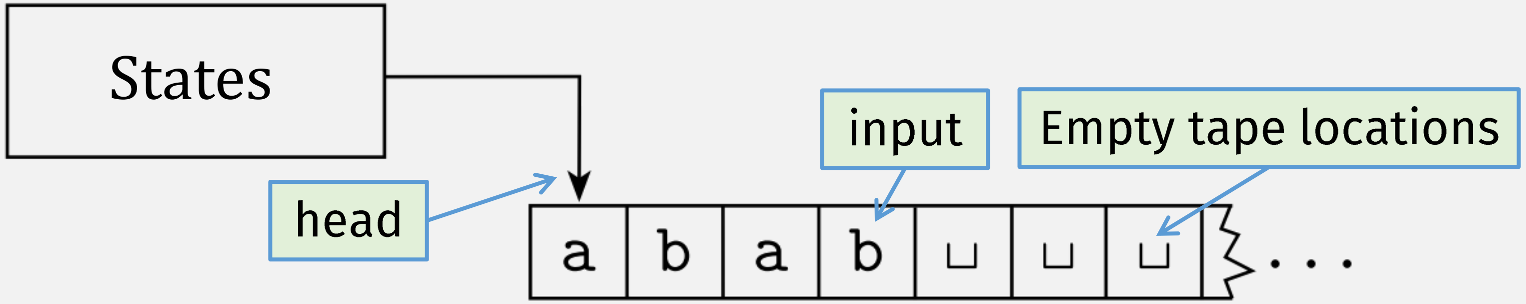
Max Woolf
@minimaxir · Follow

congrats to OpenAI on winning the Turing Test



Finite Automata vs Turing Machines

- **Turing Machines** can <u>read and write</u> to <u>arbitrary</u> **“tape” cells**
  - Tape initially contains **input string**
- **Tape is infinite**
  - To the right



- <u>Each step</u>: **“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

$0\ 1\ 1\ 0\ 0\ 0\ \#\ 0\ 1\ 1\ 0\ 0\ 0\ \sqcup\ \ldots$

$M_1 =$ "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"
Low-level 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

0 1 1 0 0 0 # 0 1 1 0 0 0 ␣ ...

x 1 1 0 0 0 # 0 1 1 0 0 0 ␣ ...

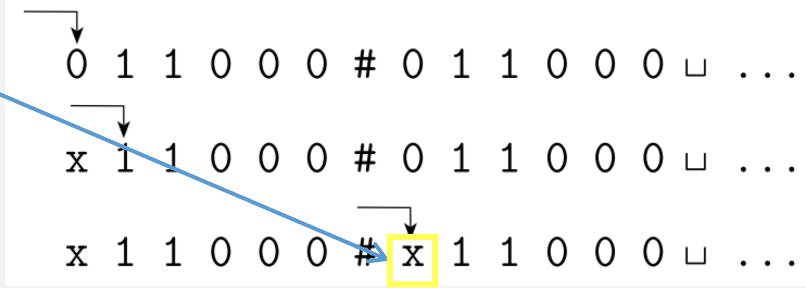
Turing Machine Example

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.

"Cross off" = write "x" char



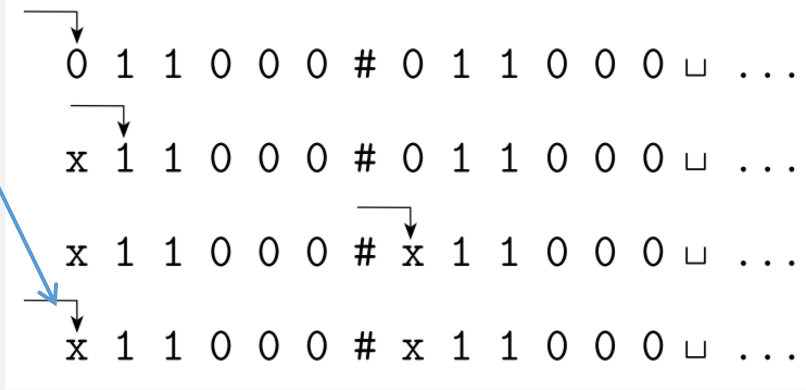
Turing Machine Example

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.

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.

Continue crossing off

```
0 1 1 0 0 0 # 0 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # 0 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # x 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # x 1 1 0 0 0 ␣ ...
x x 1 0 0 0 # x 1 1 0 0 0 ␣ ...
```

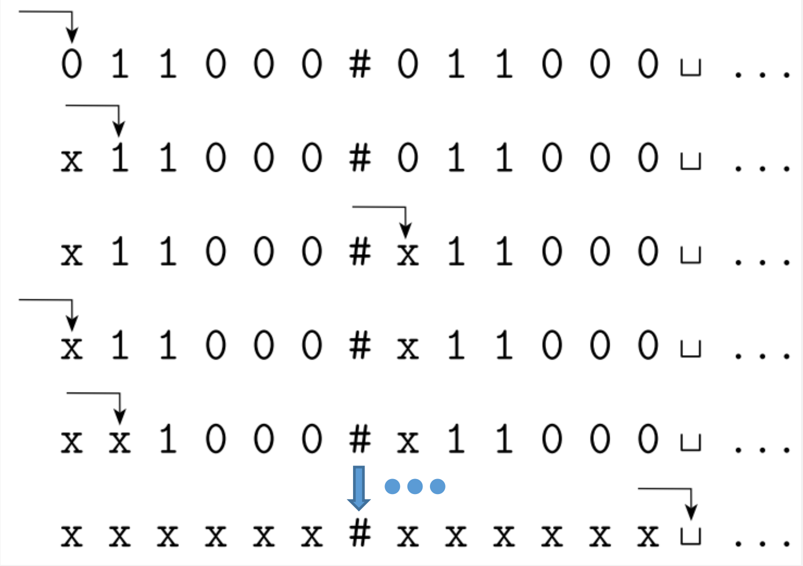
Turing Machine Example

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. $\boxed{\text{When all symbols to the left of the \# have been crossed off,}}$ check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise, *accept*."



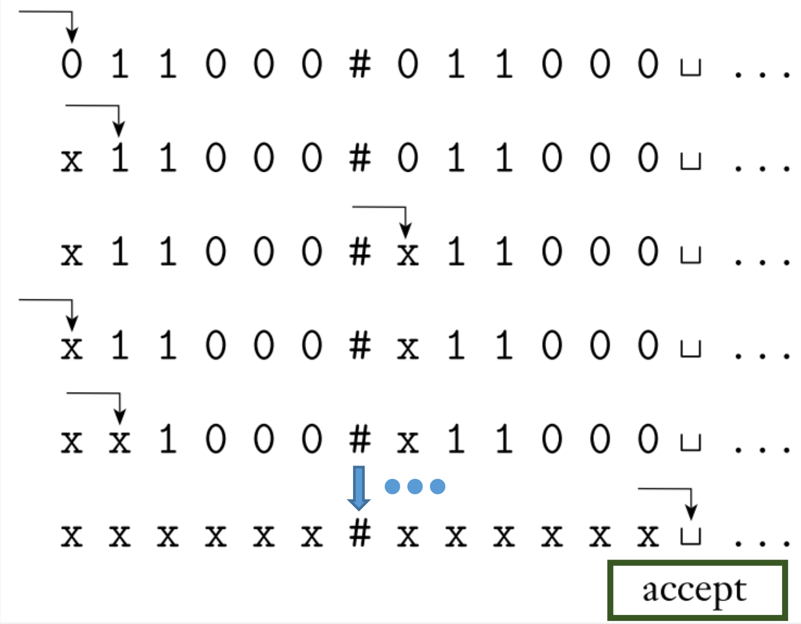
Turing Machine Example

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

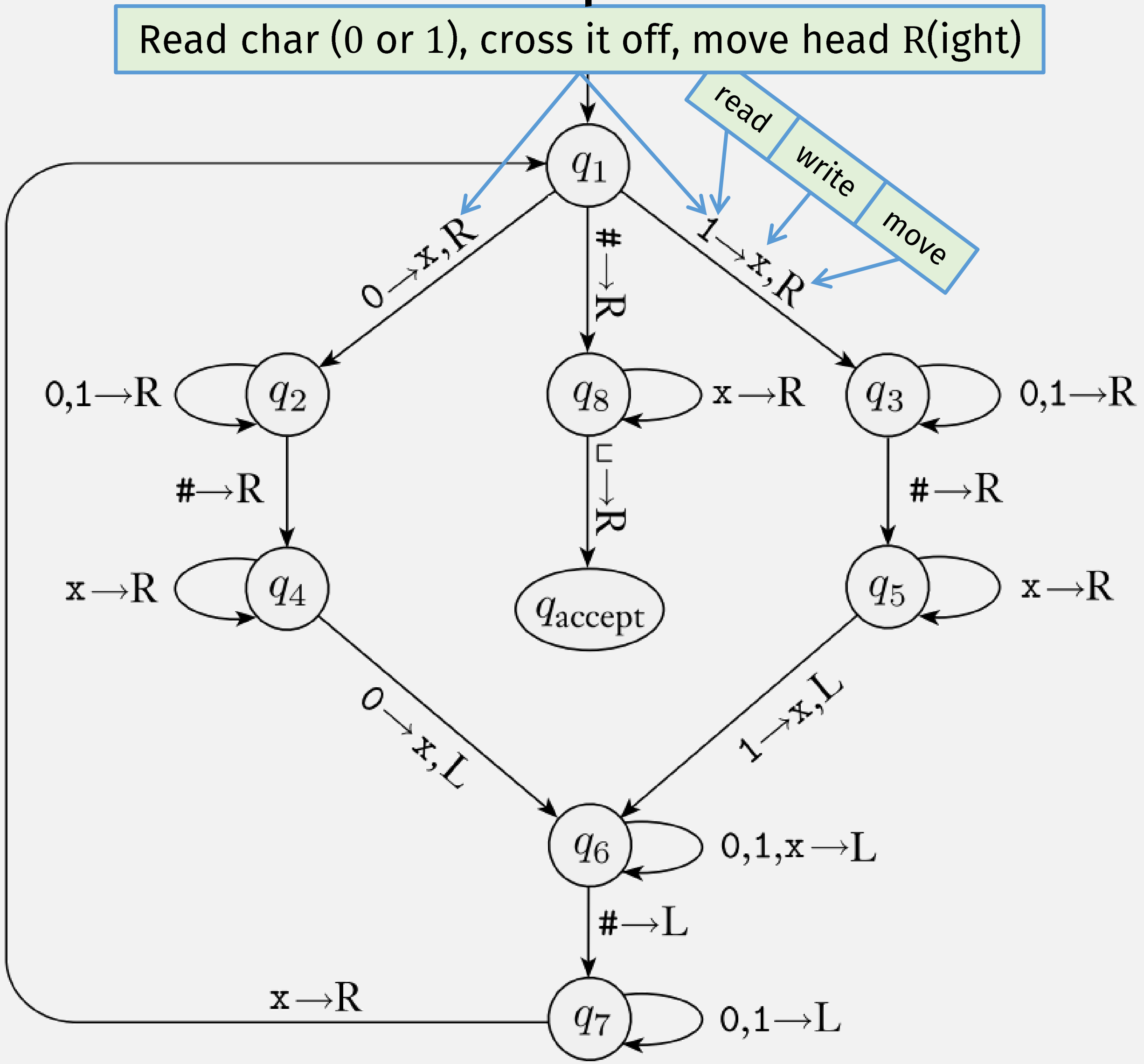
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the $\boxed{\textbf{\textit{blank symbol}}\ \sqcup}$,
3. Γ is the tape alphabet, where $\boxed{\sqcup} \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\boxed{\delta\colon Q \times \underbrace{\Gamma}_{\text{read}} \longrightarrow Q \times \underbrace{\Gamma}_{\text{write}} \times \underbrace{\{\mathrm{L}, \mathrm{R}\}}_{\text{move}}}$ 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 $q_{\text{reject}} \neq q_{\text{accept}}$.

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

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

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. $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ 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 $q_{reject} \neq q_{accept}$.

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

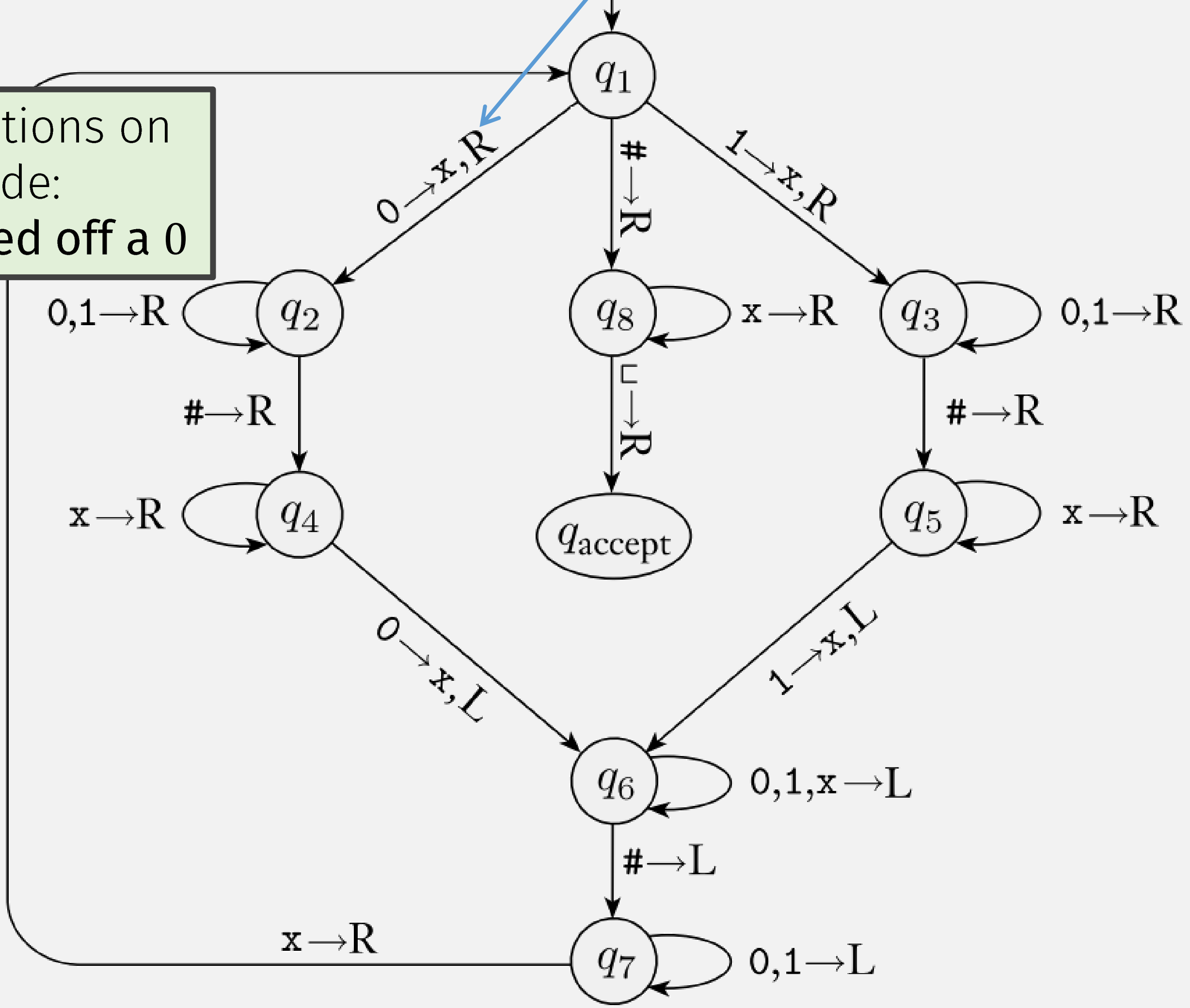
Formal Turing Machine Example

0 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ . . .

x 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ . . .

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

Transitions on this side:
Crossed off a 0

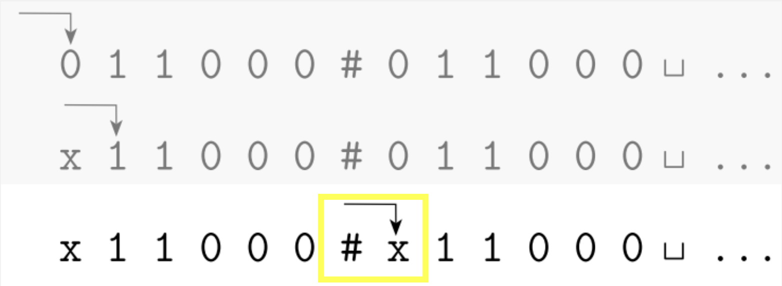


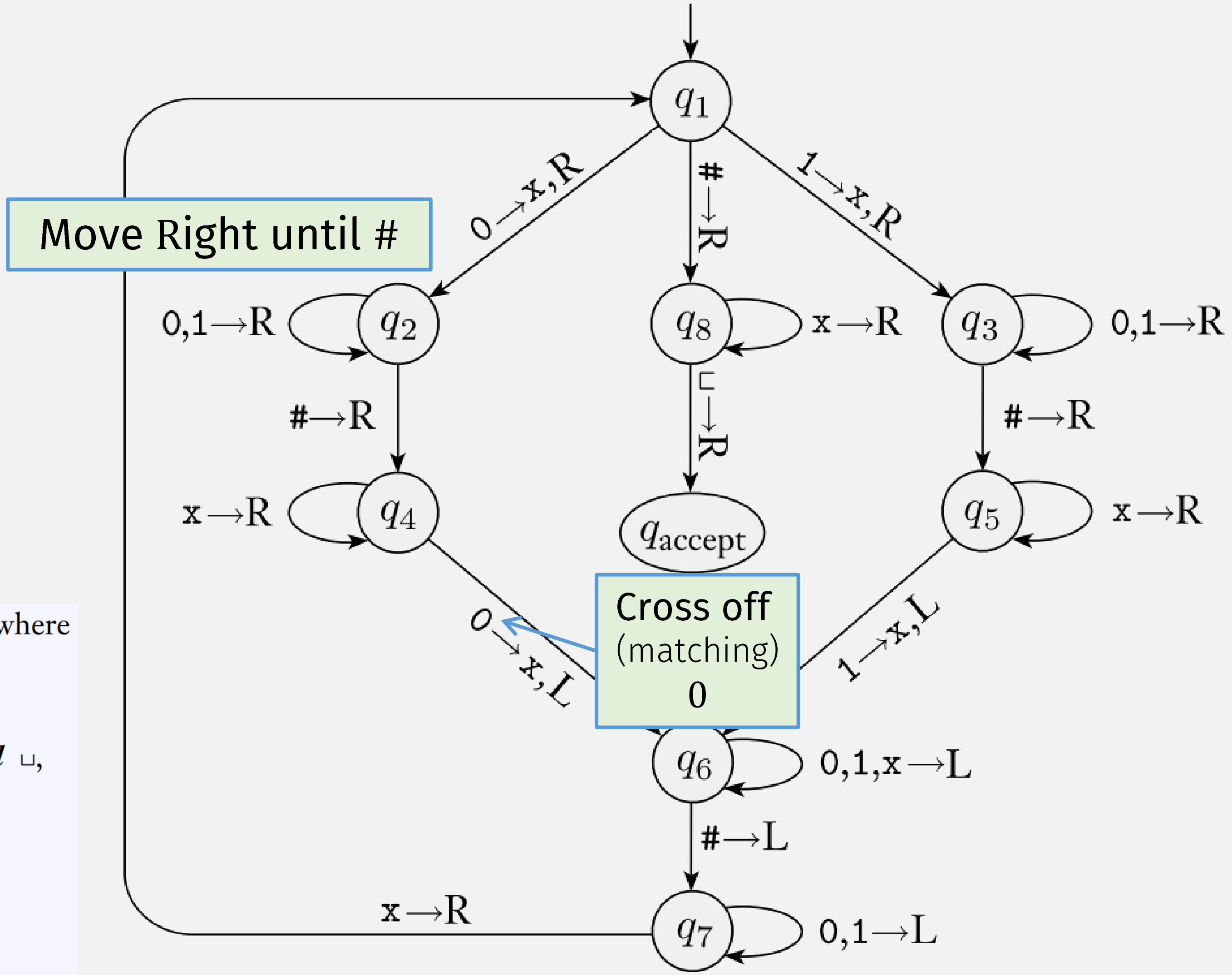
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

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, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, 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*** ⊔,
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. $q_0 \in$ read e 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 $q_{\text{reject}} \neq q_{\text{accept}}$.

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

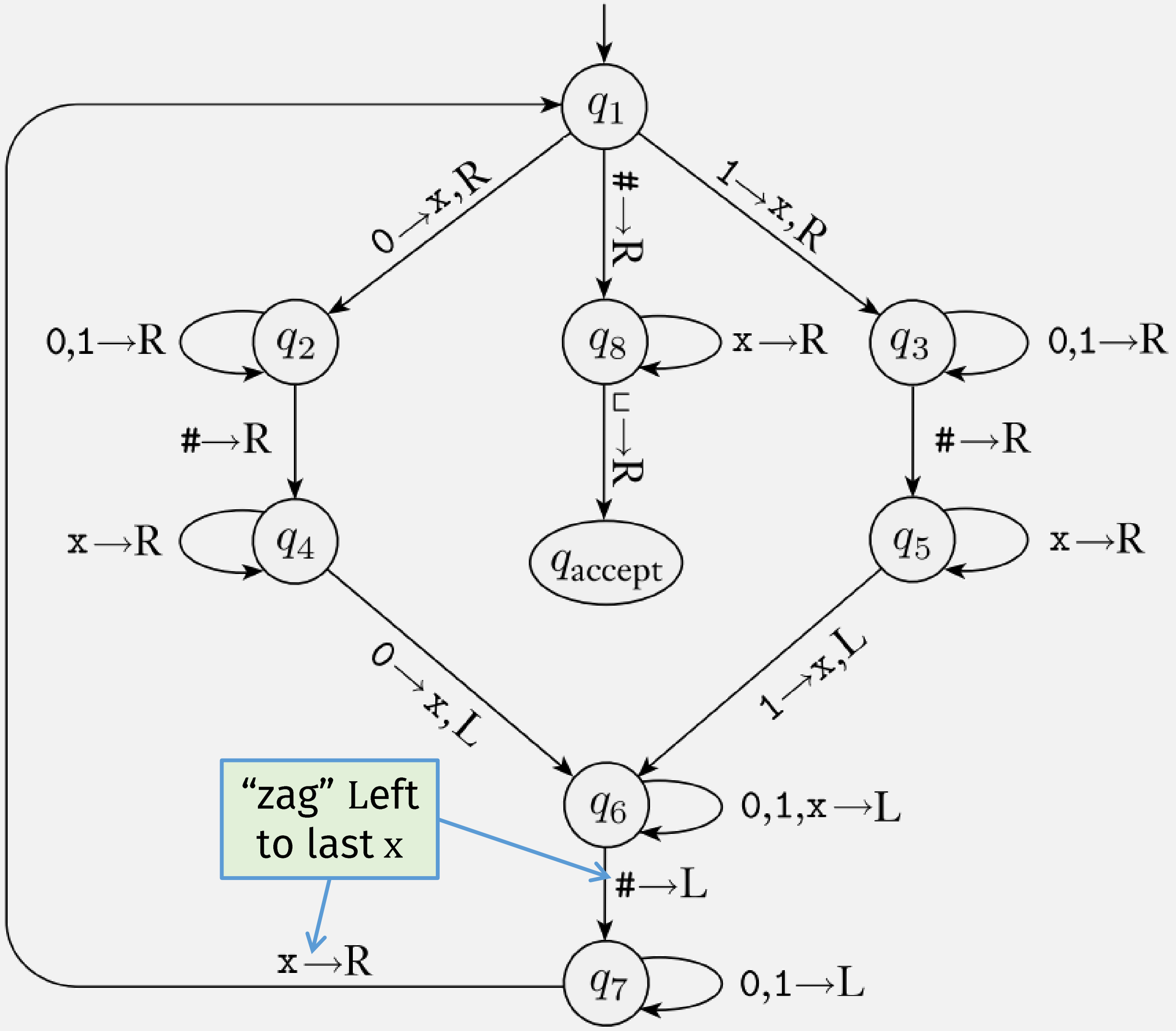
Formal Turing Machine Example

0 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ . . .

x 1 1 0 0 0 # 0 1 1 0 0 0 ⊔ . . .

x 1 1 0 0 0 # x 1 1 0 0 0 ⊔ . . .

x 1 1 0 0 0 # x 1 1 0 0 0 ⊔ . . .

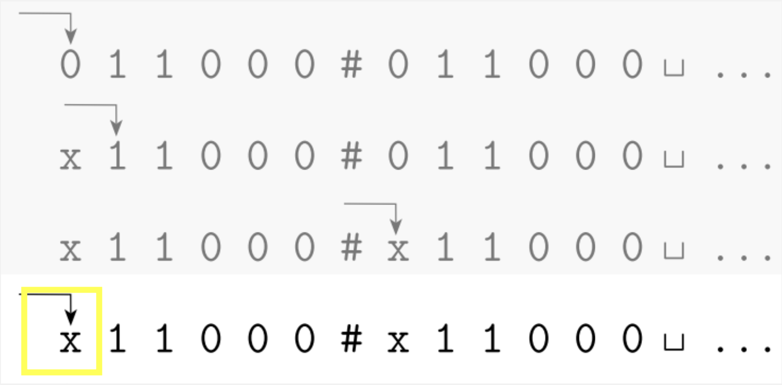


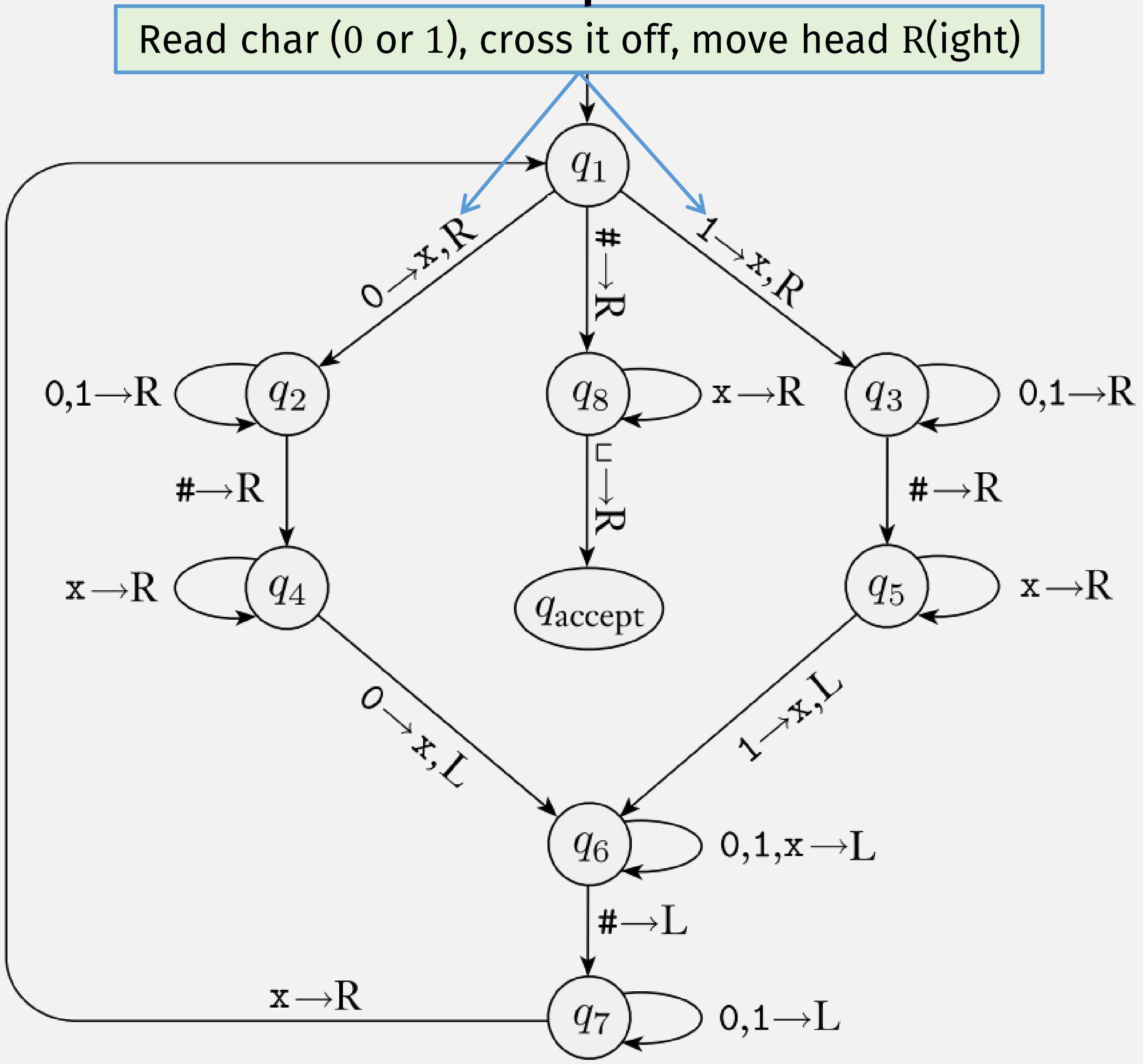
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

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

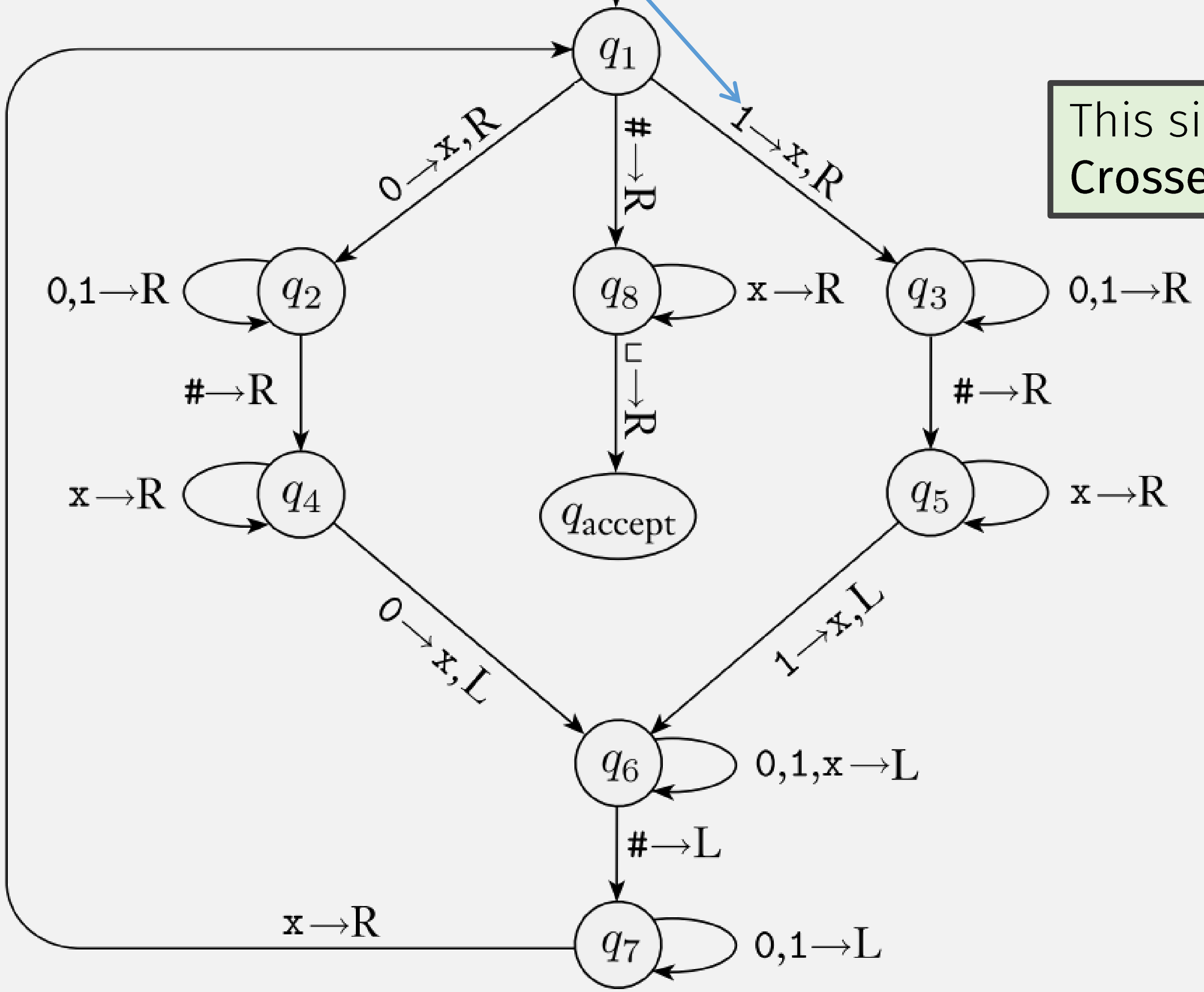
1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** ␣,
3. Γ is the tape alphabet, where ␣ $\in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is the transition function,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

```
0 1 1 0 0 0 # 0 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # 0 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # x 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # x 1 1 0 0 0 ␣ ...
x x 1 0 0 0 # x 1 1 0 0 0 ␣ ...
```

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



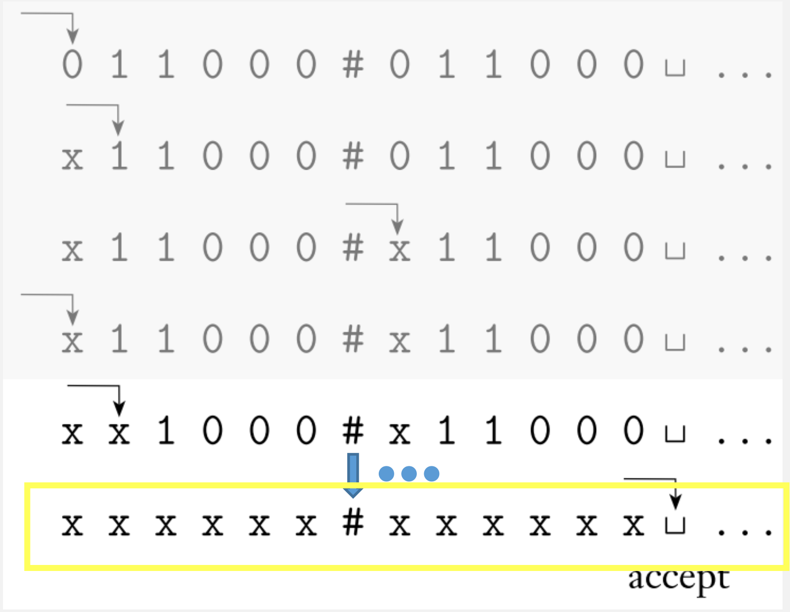
This side:
Crossed off a 1

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

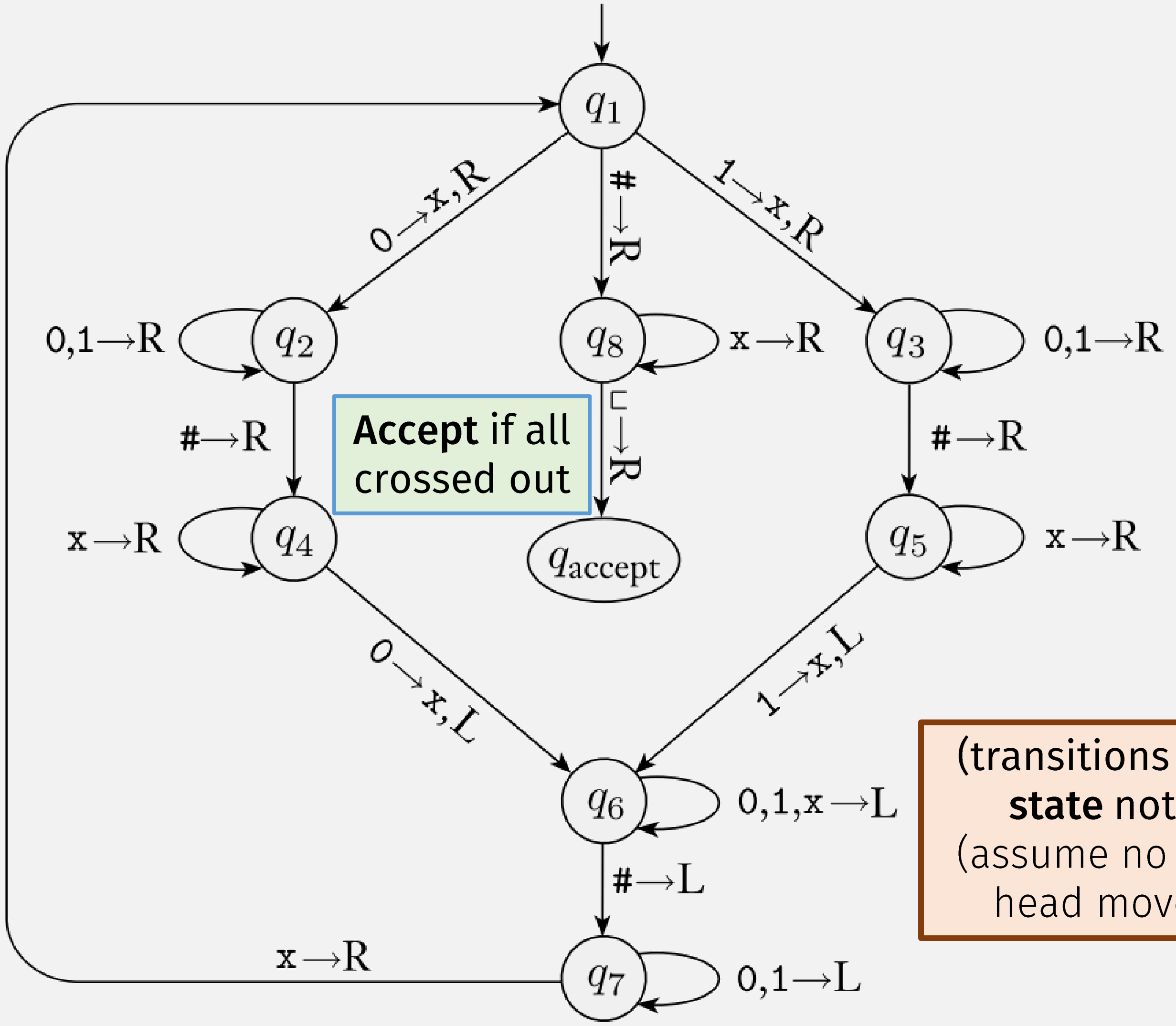
$$B = \{w\#w \mid w \in \{0,1\}^*\}$$

Formal Turing Machine Example



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



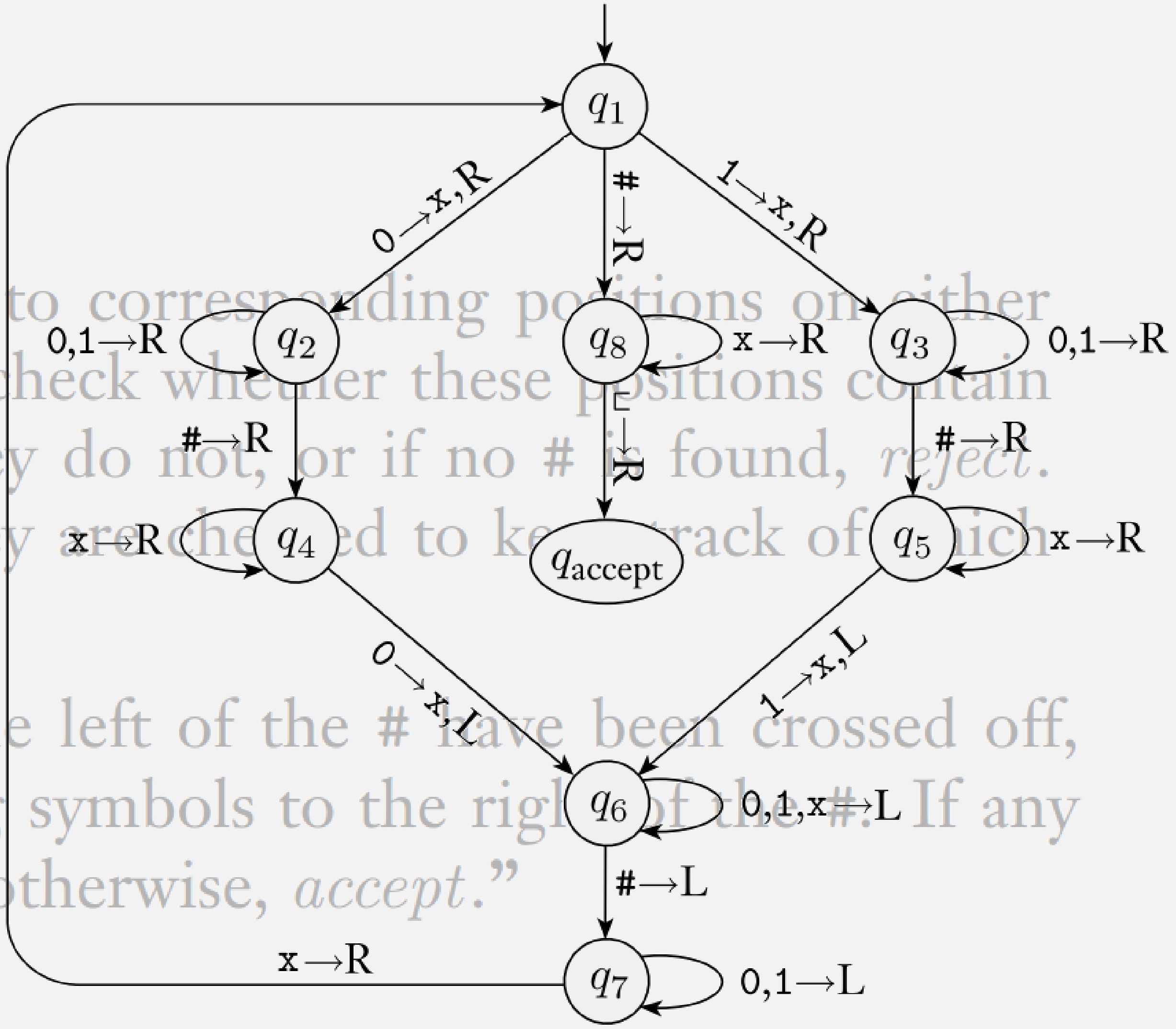
(transitions to) **Reject state not shown** (assume no write, and head moves right)

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

$M_1 =$ "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 … track of which symbols correspond.
2. When all symbols to … n crossed off, check for any remaining … t of the #. If any symbols remain, *reject*; otherwise, … ccept."

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 <u>not</u> a "do whatever" card, some rules:

1. TMs and input strings must be <u>named</u> (like function definitions)
2. Steps must be numbered
3. TMs can "call" or "simulate" other TMs (if they pass appropriate arguments!)
   - e.g., step for a TM M can say: "call TM M_2 with argument string w, if M_2 accepts w then ..., else ..."
   - Can split input into substrings and pass to different TMs
4. Follow typical programming "scoping" rules
   - can assume functions we've already defined are in "global" scope, RE2NFA ...
5. Other variables must also be defined before use
   - e.g., can define a TM inside another TM
6. must be **equivalent** to a low-level formal tuple
   - high-level "<u>step</u>" represents a <u>finite</u> # of low-level δ transitions
   - So one step cannot run forever
   - E.g., can't say "try all numbers" as a "step"

$M_1 =$ "On input string w:

$M =$ "On input w
1. Simulate B on input w.
2. If simulation ends in accept state,

$N =$ "On input $\langle B, w\rangle$, where B is an NFA and w is a string:
1. Convert NFA B to an equivalent DFA C, using the procedur
this conversion given in Theorem 1.39.
2. Run TM M from Theorem 4.1 on input $\langle C, w\rangle$.

$S =$ "On input w
1. Construct the following TM M_2.
$M_2 =$ "On input x:

Non-halting Turing Machines (TMs)

IMPORTANT SLIDE

So: TM computation has 3 possible results:
- Accept
- Reject
- Loop forever

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

- We will work with 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.

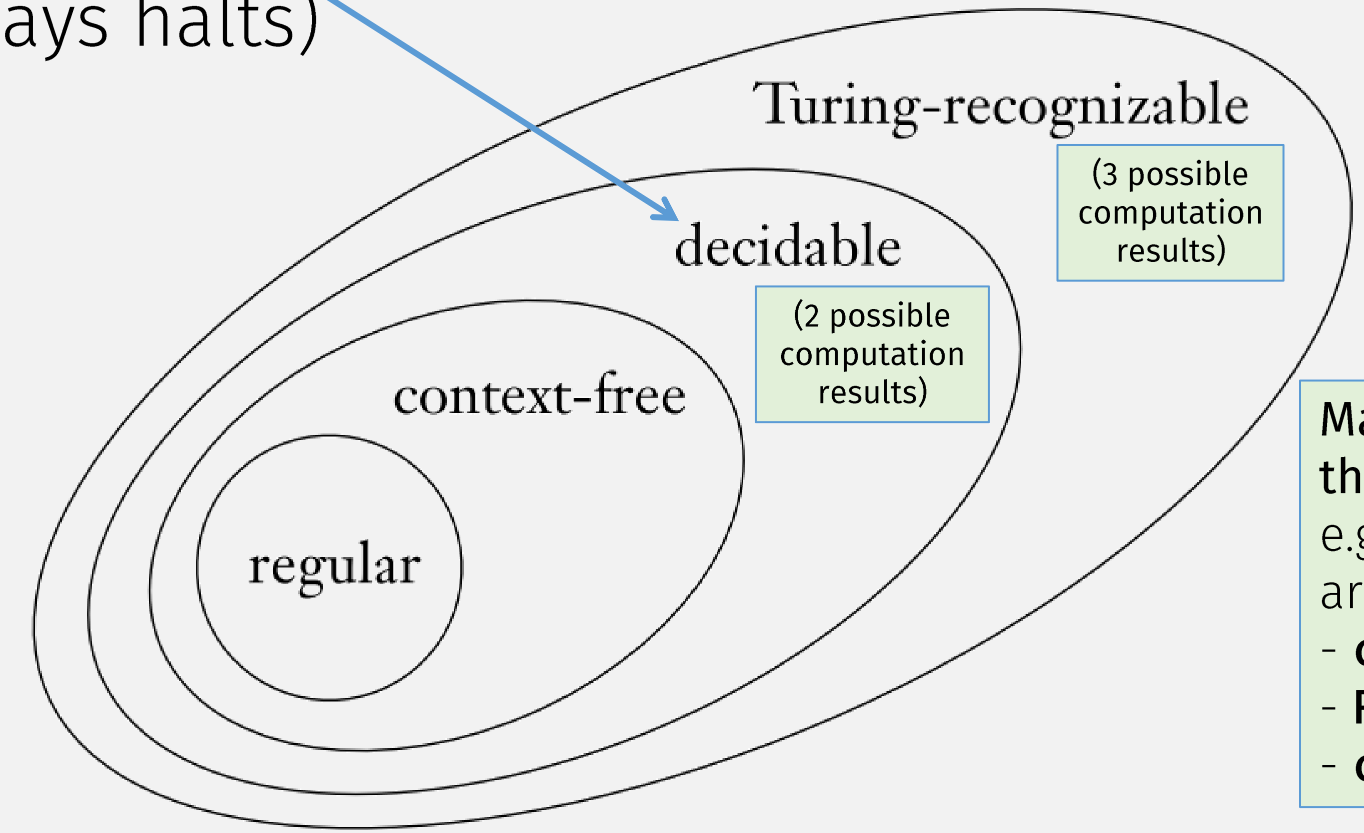
(3 possible computation results)

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

(2 possible computation results)

Formal Definition of an "Algorithm"

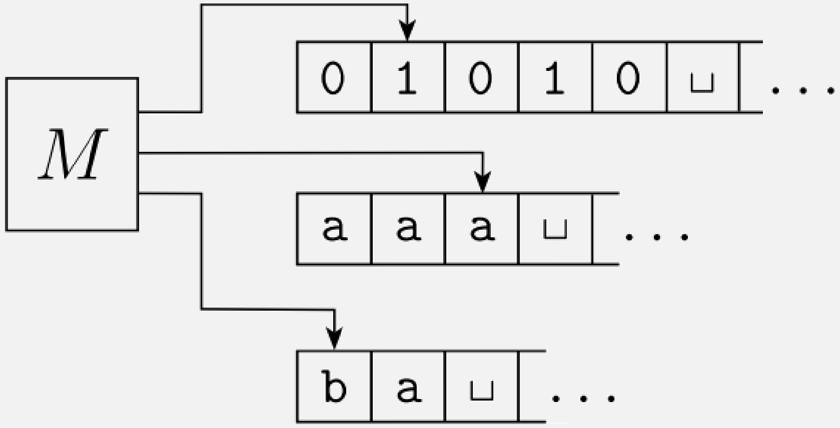
- An **algorithm** is equivalent to a **Turing-decidable** Language (always halts)



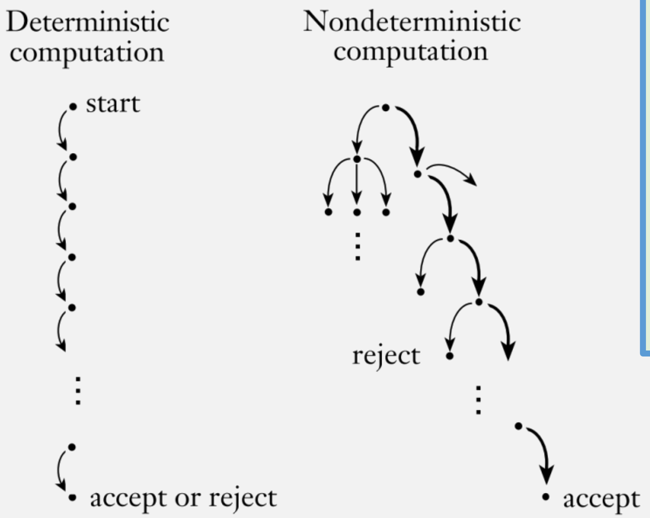
Many <u>functions</u> we have **defined** this semester are **algorithms**!
e.g., all our <u>conversion</u> functions are **deciders**!!
- `convertD2N`
- `RegExpr2NFA`
- `convertD2P`

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?)

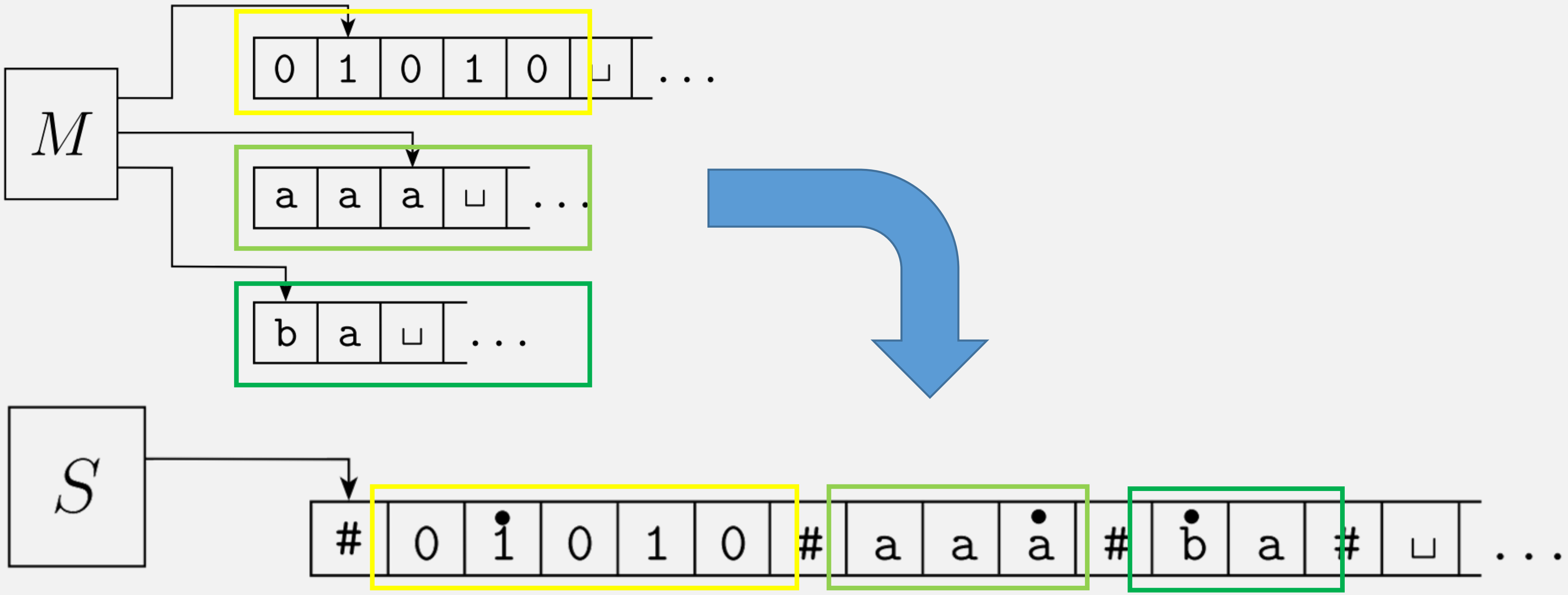
⇐ If a multi-tape TM recognizes a language,
then a single-tape TM recognizes the language

- 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

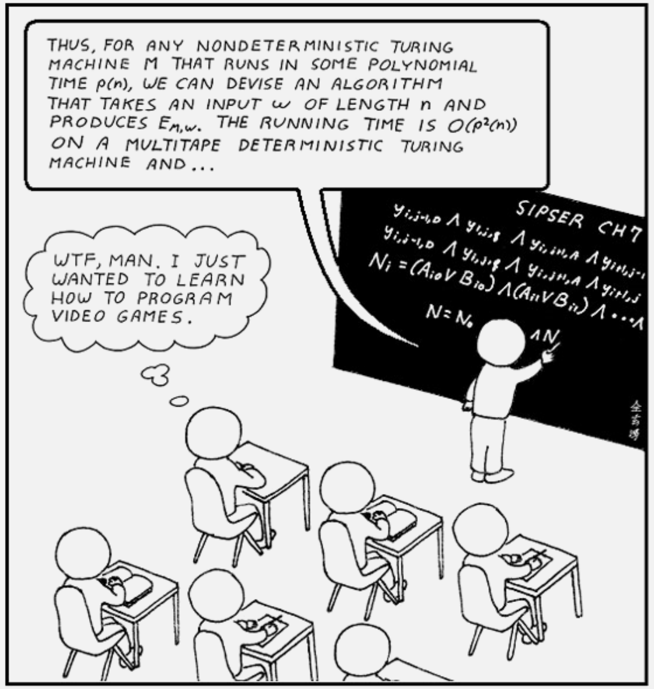
- Single-tape TM is equivalent to …
- … multi-tape TM that only uses one of its tapes

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

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

∎

Nondeterministic TMs



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

Nondeterministic transition produces <u>set</u> 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. $\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ 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 $q_{\text{reject}} \neq q_{\text{accept}}$.

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. $\cancel{\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}}$ $\boxed{\delta \colon Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\})}$
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

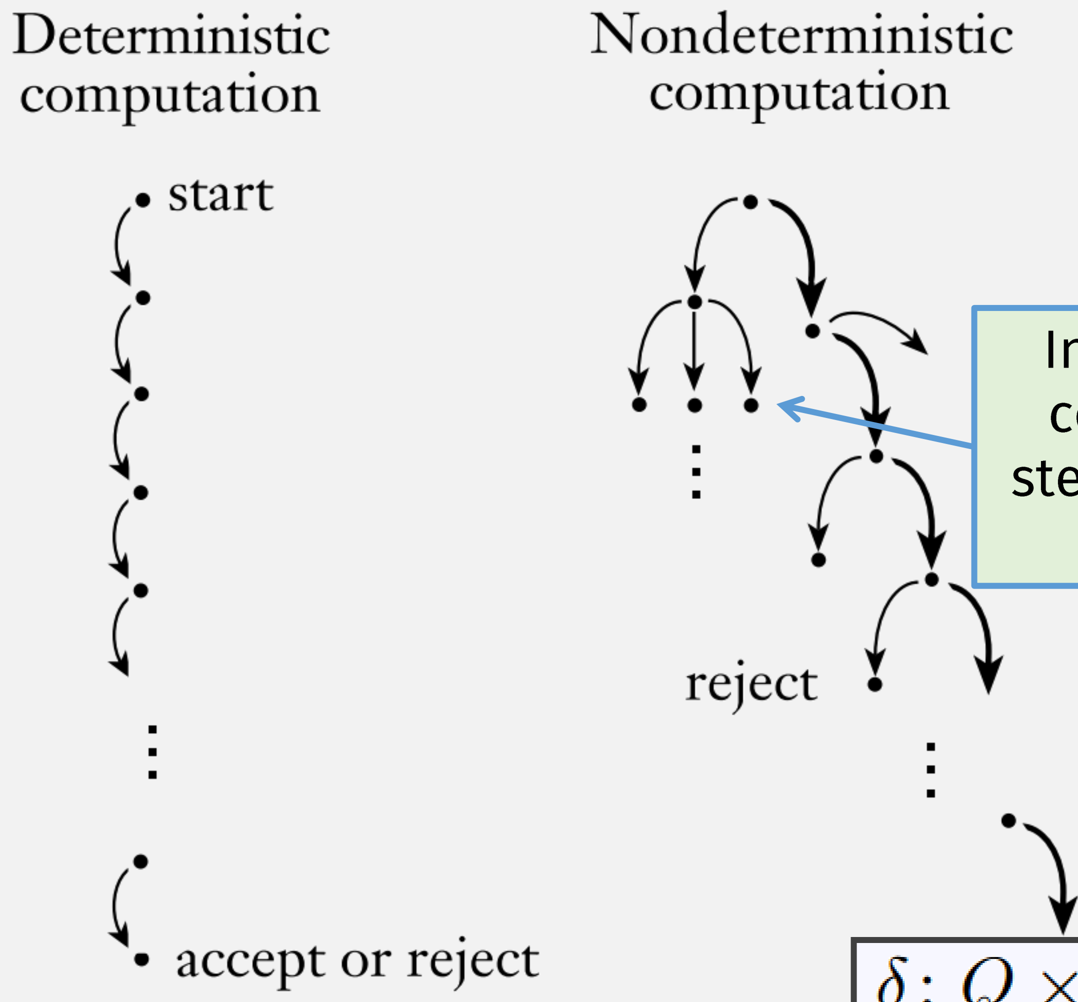
⇒ If a **deterministic TM** recognizes a language,
then a **non-deterministic TM** recognizes the language

- Convert: Deterministic TM → Non-deterministic TM …
- … change **Deterministic TM δ fn output to a one-element set**
  - $\delta_{\text{ntm}}(q, a) = \{\delta_{\text{dtm}}(q, a)\}$
  - (just like conversion of DFA to NFA --- HW 3, Problem 1)
- **DONE!**

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

- 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

- 3 components (q, w, γ) :

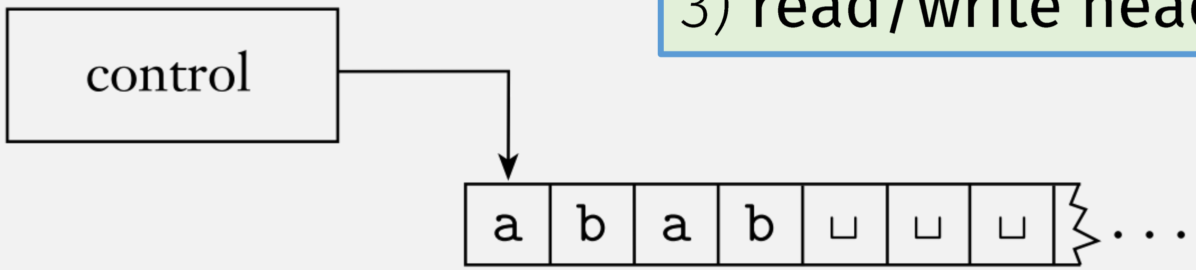
  q = the current state
  w = the remaining input string
  γ = the stack contents

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

TM Configuration (ID) = ???

1) states

3) read/write head

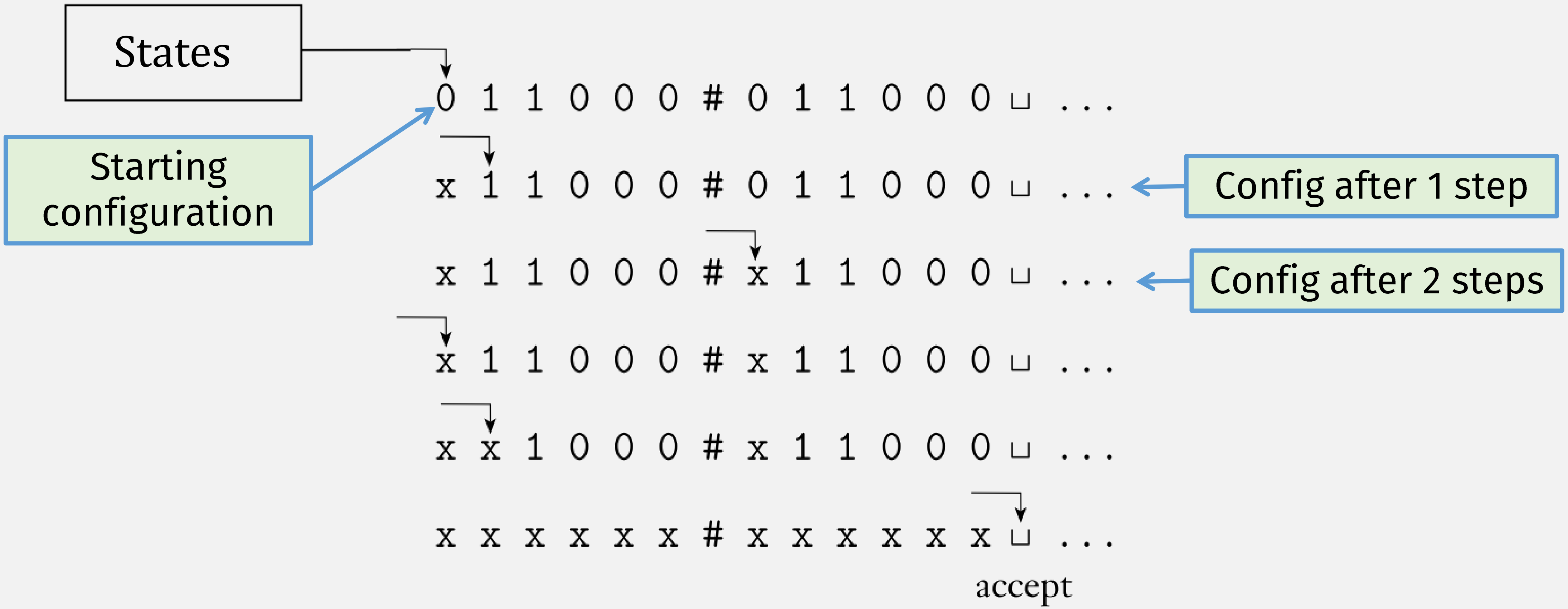


2) Tape contents

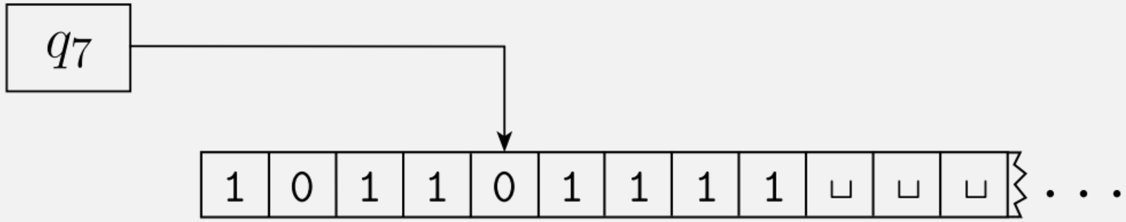
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

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. $\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ 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 $q_{\text{reject}} \neq q_{\text{accept}}$.

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

<u>Edge cases</u>:

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

Head stays at leftmost cell

(L move, when already at leftmost cell)

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

Add blank symbol to config

(R move, when at rightmost filled cell)

Extended

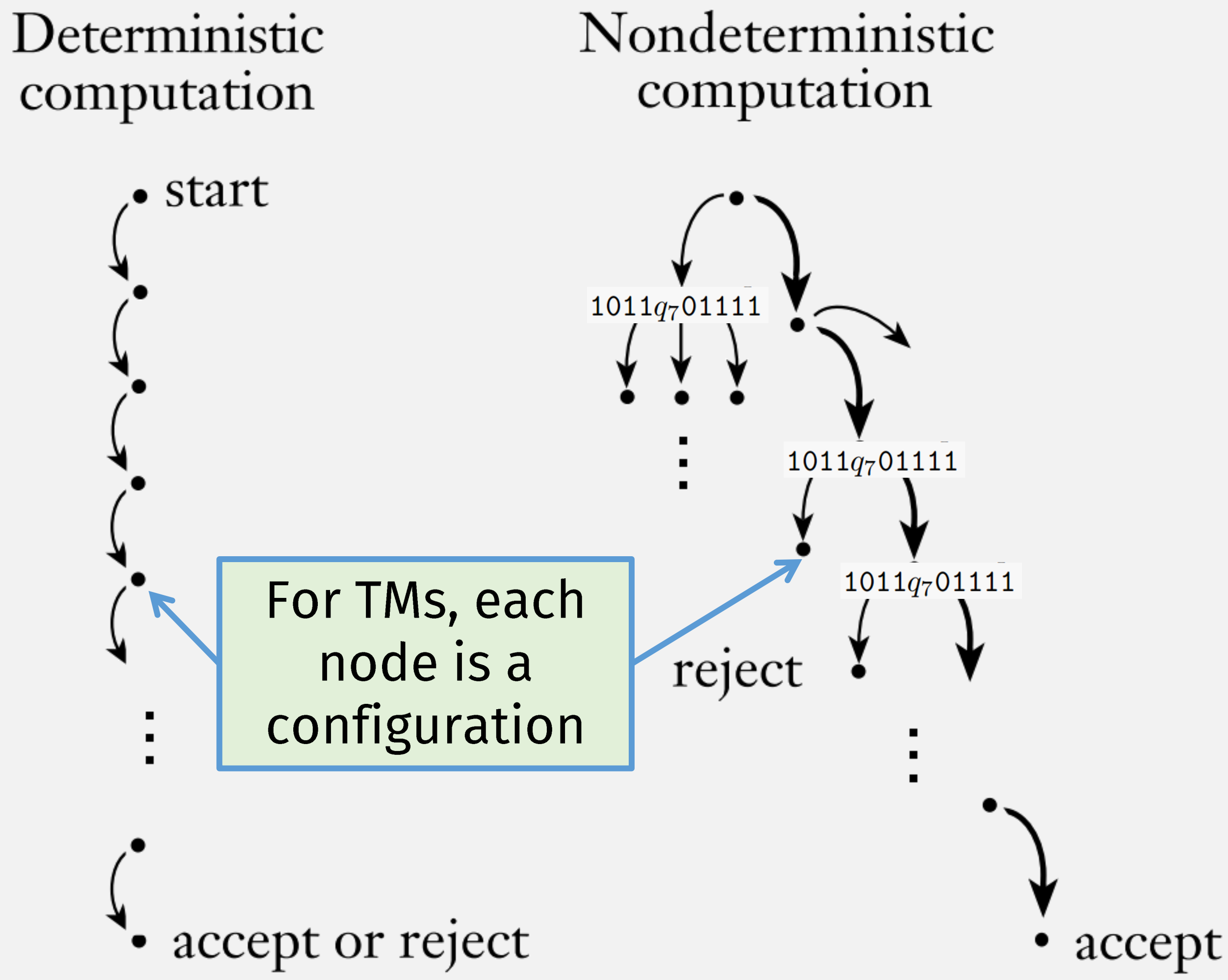
- Base Case

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

- Recursive Case

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

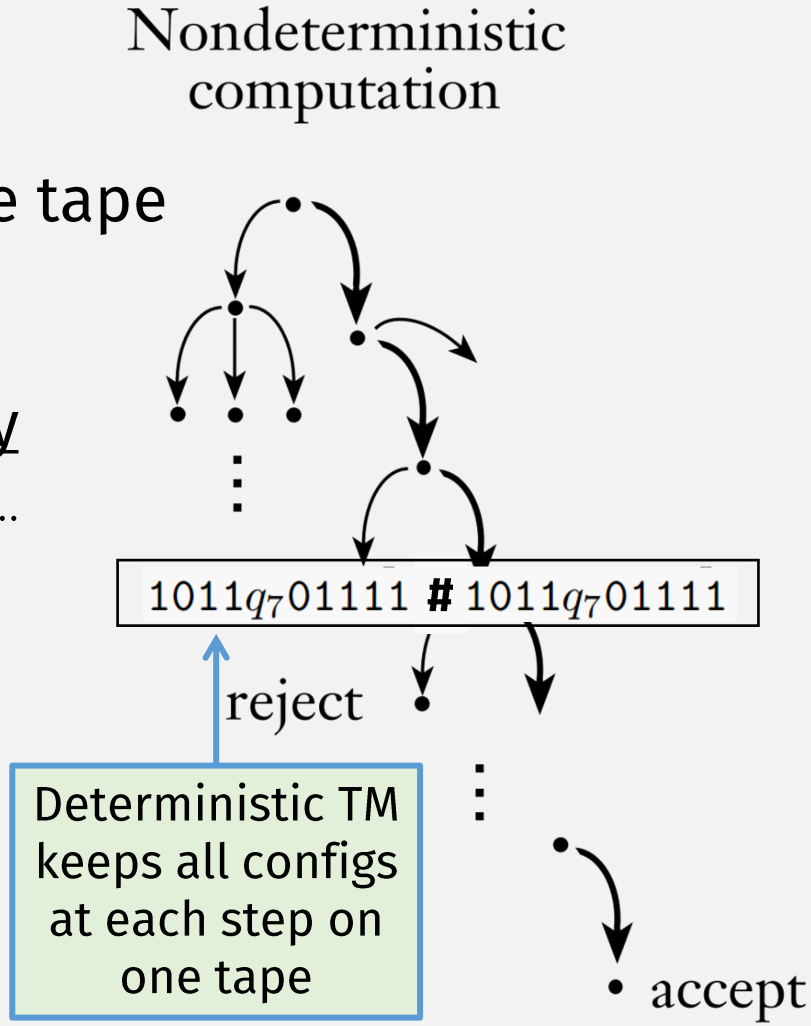
Nondeterminism in TMs



Nondeterministic TM → Deterministic 1st way

- Simulate NTM with Det. TM:
  - Det. TM keeps multiple configs on 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?

Because any one path can go on forever!



Interlude: Running TMs inside other TMs

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

Exercise:

- Given: TMs M_1 and M_2
- Create: TM $\boldsymbol{M}$ that **accepts** if either M_1 or M_2 accept

“loop” means input string not accepted

M_1	M_2	$\boldsymbol{M}$
reject	accept	accept
accept	reject	accept
accept	loops	accept
loops	accept	loops

Possible solution #1:

$\boldsymbol{M}$ = on input x,

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

... with concurrency!

Exercise:

- Given: TMs M_1 and M_2
- Create: TM $\boldsymbol{M}$ that **accepts** if either M_1 or M_2 accept

Possible solution #1:

M = on input x,

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

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

Possible solution #2:

$\boldsymbol{M}$ = on input x,

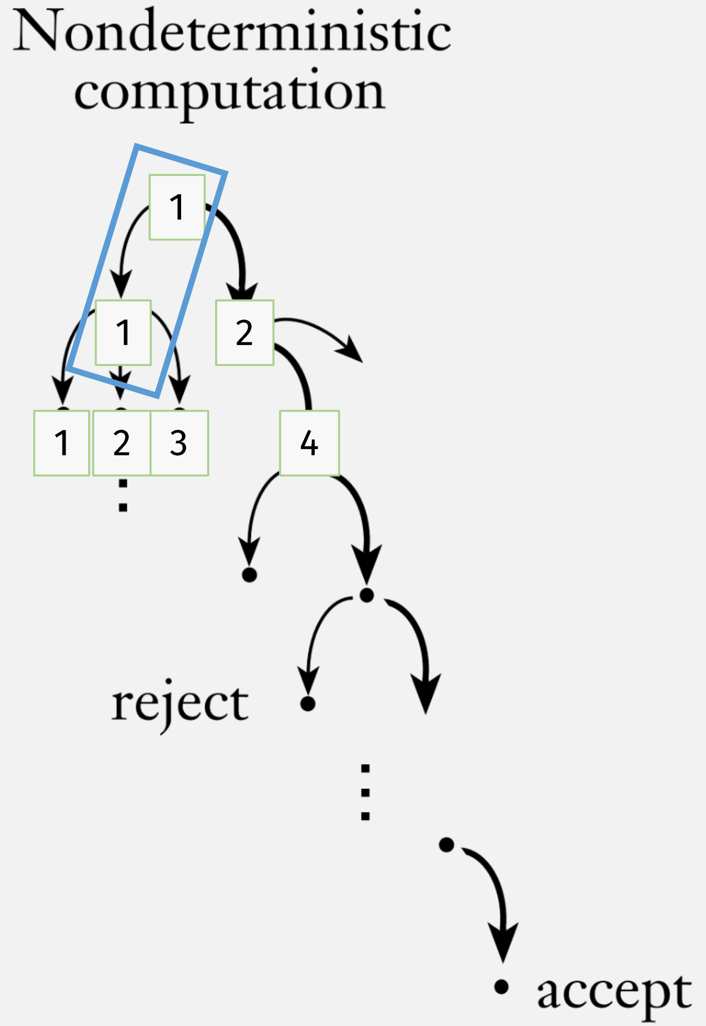
1. `Call` M_1 and M_2, each with x, concurrently, i.e.,
   a) Run M_1 with x for 1 step; **accept** if M_1 accepts
   b) Run M_2 with x for 1 step; **accept** if M_2 accepts
   c) Repeat

M_1	M_2	$\boldsymbol{M}$
reject	accept	**accept**
accept	reject	**accept**
accept	loops	**accept**
loops	accept	**accept**

Nondeterministic TM → Deterministic

2nd way
(Sipser)

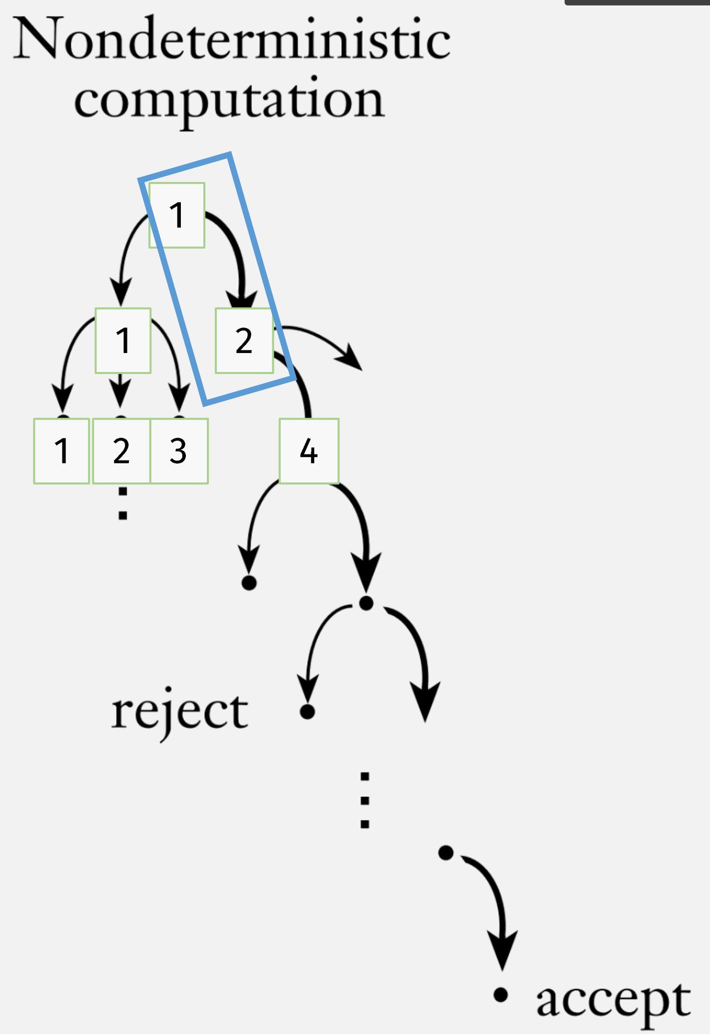
- **Simulate NTM with Det. TM:**
  - **Number the nodes at each step**
  - **Check all tree paths** (in breadth-first order)
    - 1
    - **1-1**



Nondeterministic TM → Deterministic

2nd way
(Sipser)

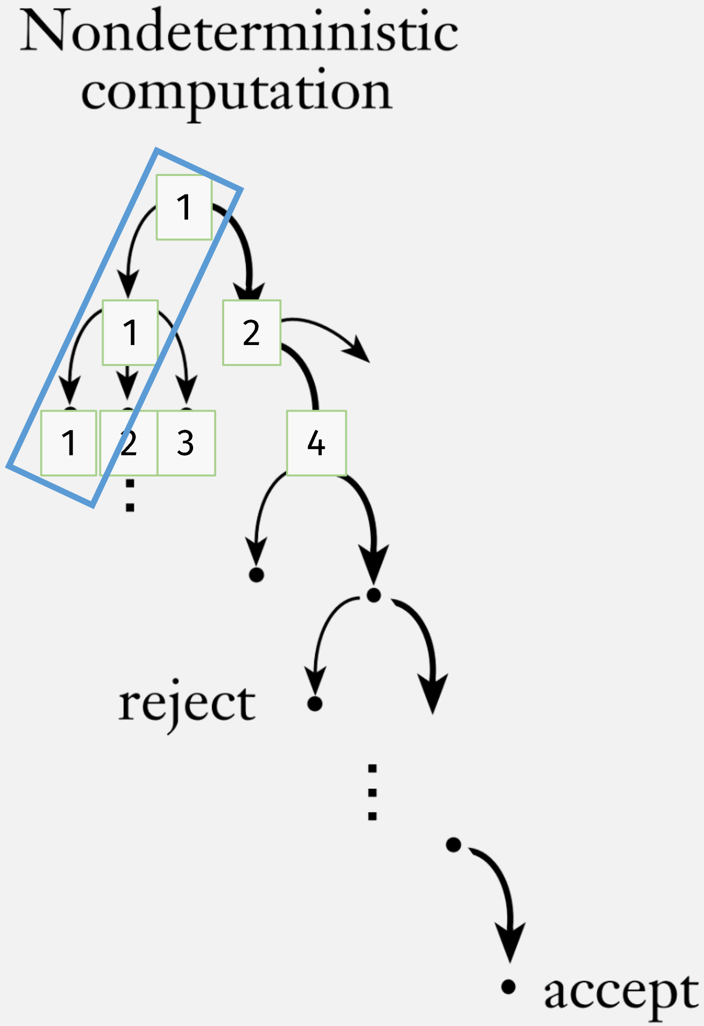
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Check all tree paths (in breadth-first order)
    - 1
    - 1-1
    - **1-2**



Nondeterministic TM → Deterministic

2nd way
(Sipser)

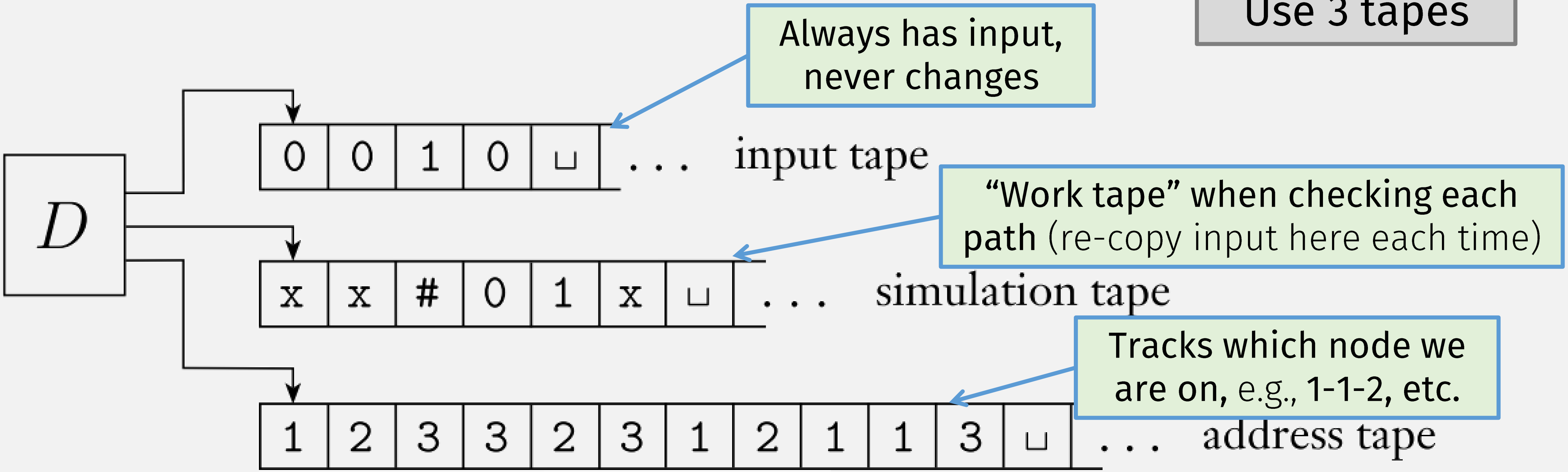
- Simulate NTM with Det. TM:
  - Number the nodes at each step
  - Check all tree paths (in breadth-first order)
    - 1
    - 1-1
    - 1-2
    - **1-1-1**



Nondeterministic TM → Deterministic

2nd way
(Sipser)

Use 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