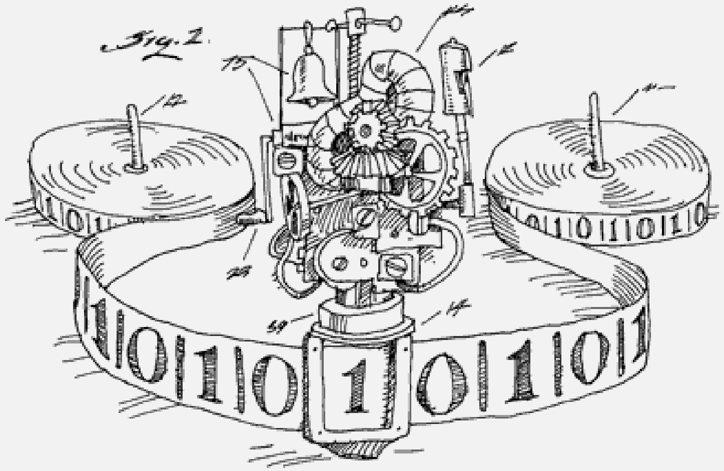
UMB CS420

Turing Machines (TMs)

Thursday, October 27, 2022



Announcements

- HW 6 out
  - due Sun 10/30 11:59pm EST

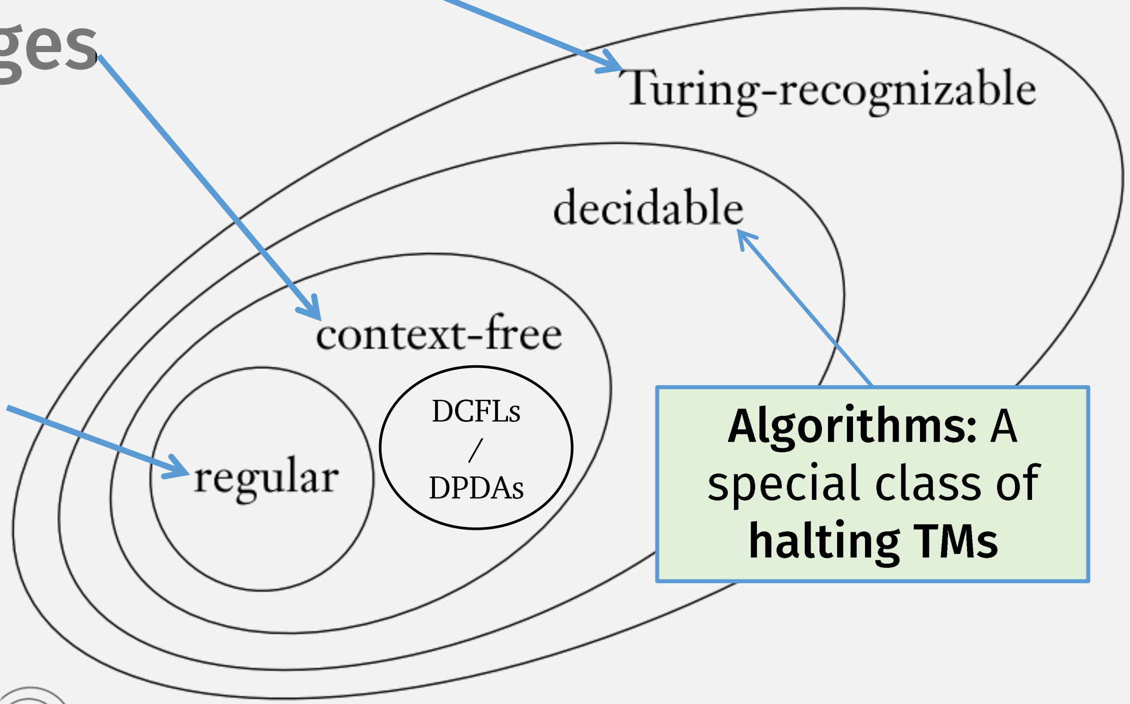
CS 420: Where We've Been, Where We're Going

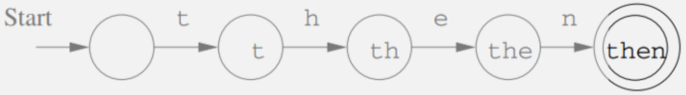
- **Turing Machines** (TMs)

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

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

- **DFAs / NFAs**: recognize **regular langs**
  - Memory: finite states
  - Can't express: **dependency**
    e.g., $\{0^n1^n \mid n \geq 0\}$



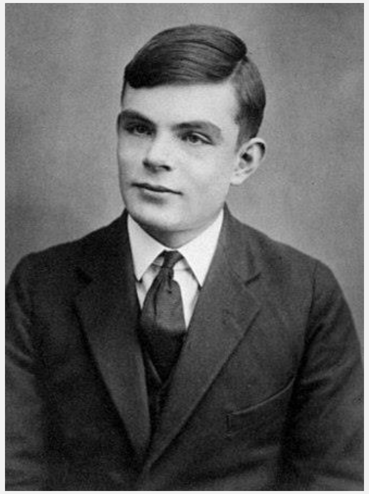


Alan Turing

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

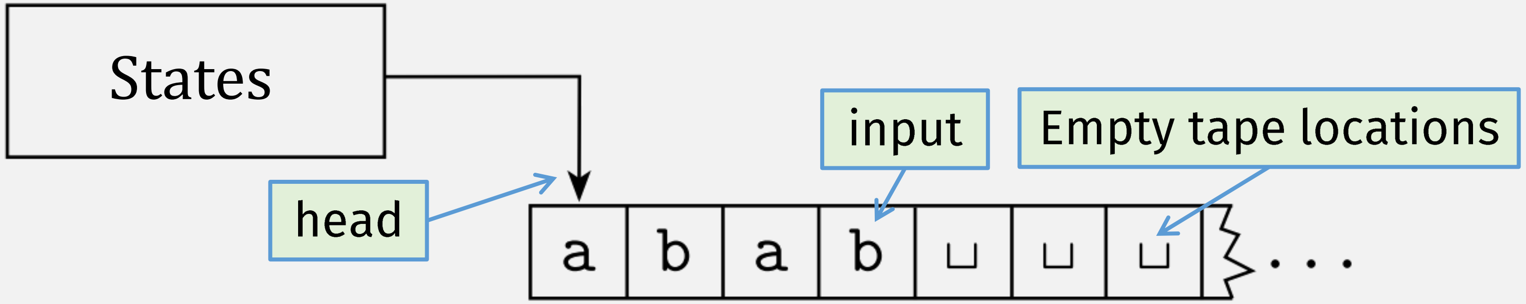
- Worked as codebreaker during WW2

- Also studied Artificial Intelligence
  - The Turing Test



Finite Automata vs Turing Machines

- **Turing Machines** can read <u>and write</u> to <u>arbitrary</u> "tape" cells
  - Tape initially contains input string
- Tape is infinite



- <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

This is an **informal TM description**
one "step" =
many 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

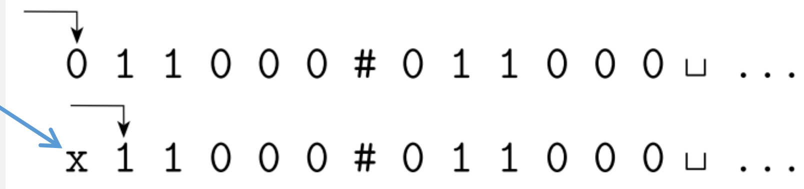
Turing Machine Example

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

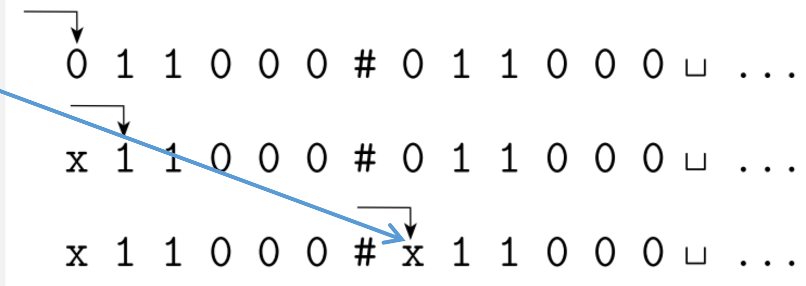
Turing Machine Example

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

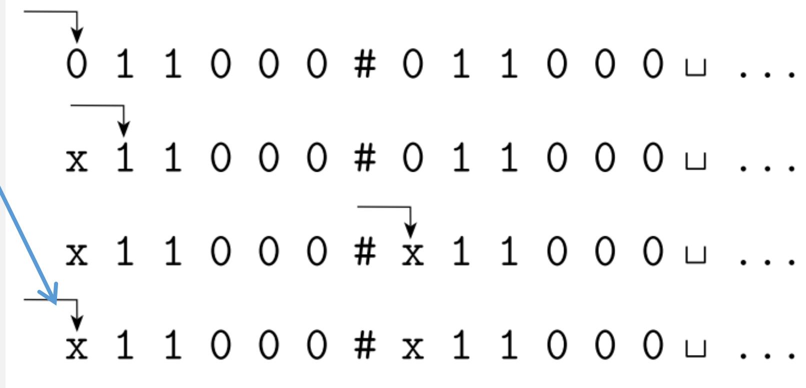
M_1 accepts inputs in language $B = \{w\#w \mid w \in \{0,1\}^*\}$

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.

Head "zags" back to start

"Cross off" = write "x" char



Turing Machine Example

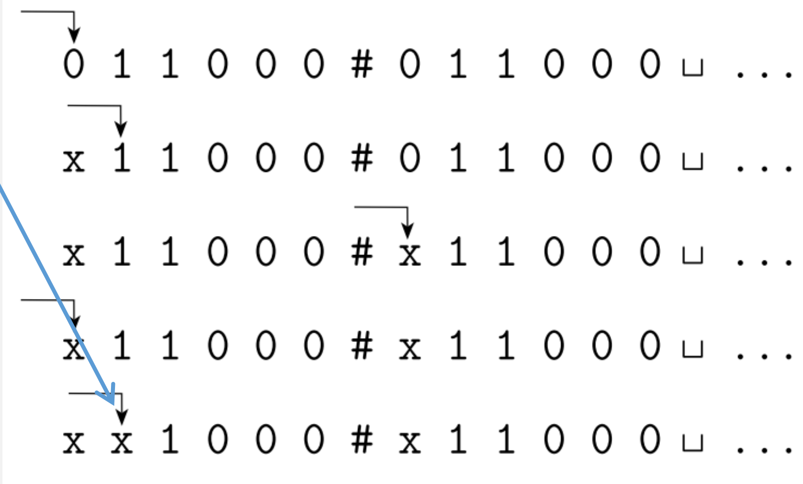
M_1 accepts inputs in language $B = \{w\#w \mid w \in \{0,1\}^*\}$

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



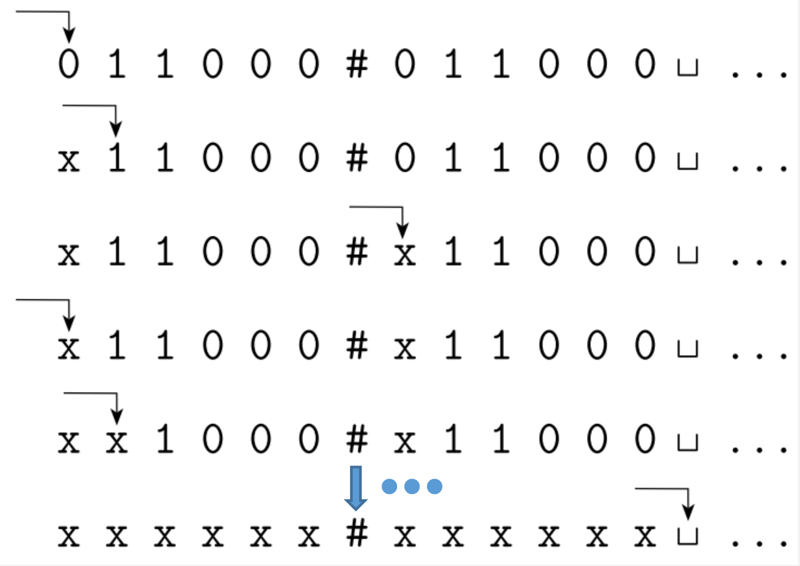
Turing Machine Example

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

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*."

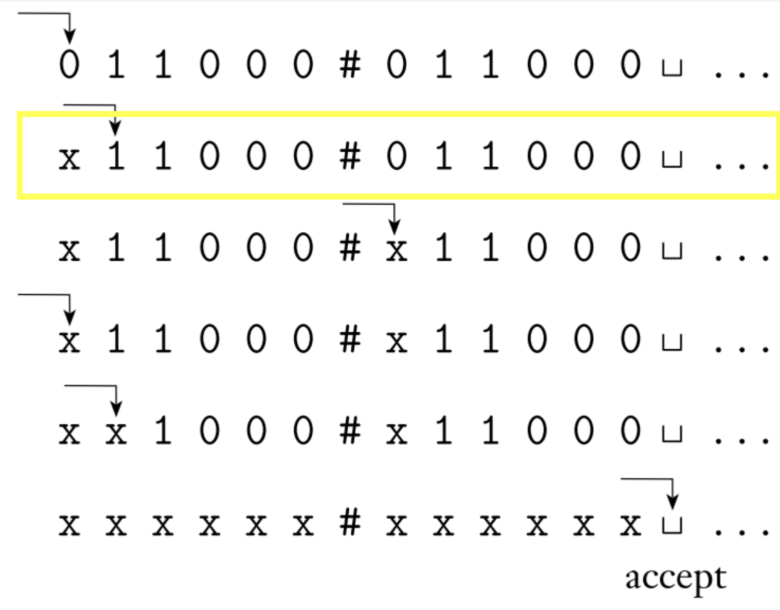
Turing Machines: Formal Definition

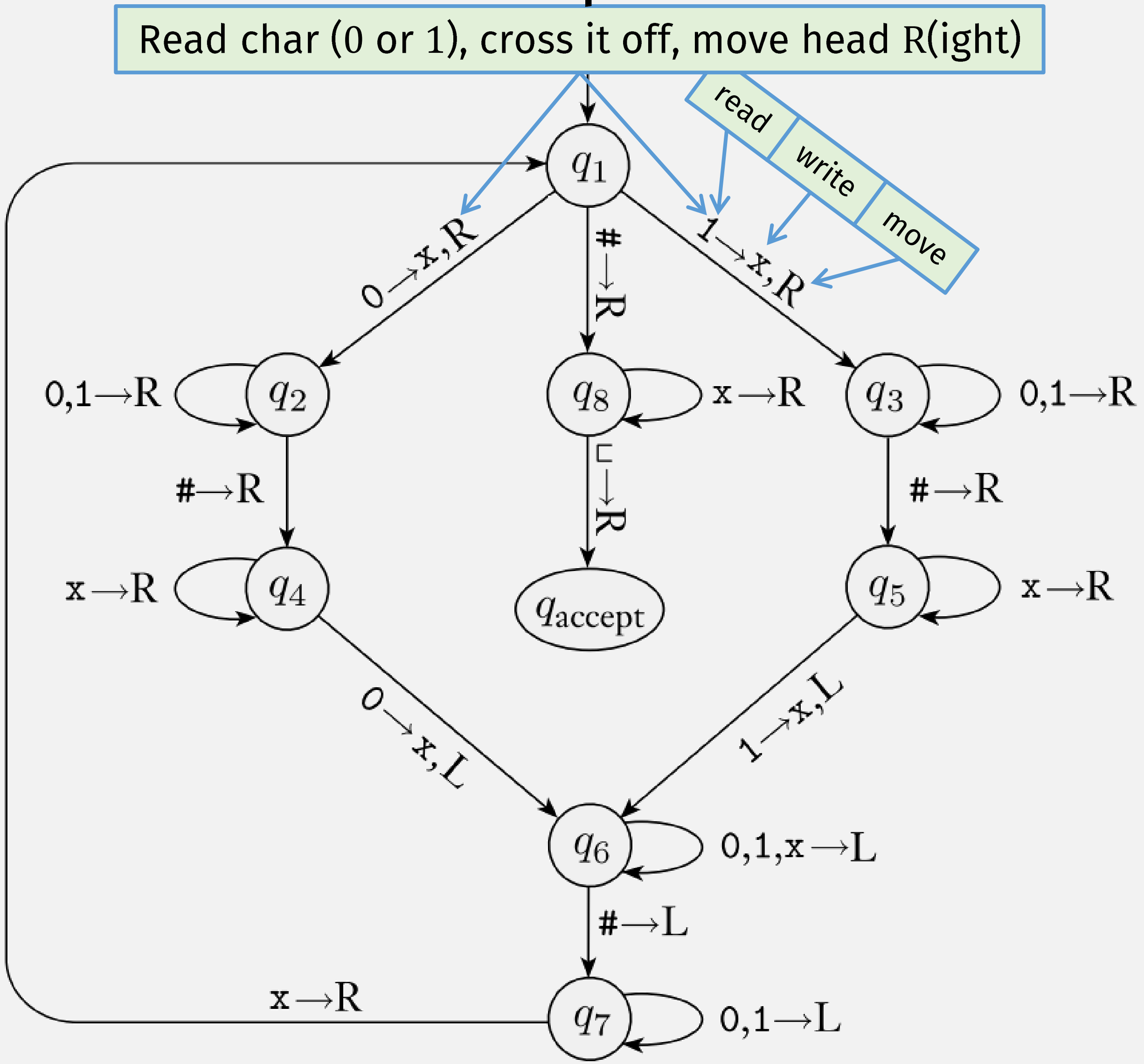
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

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



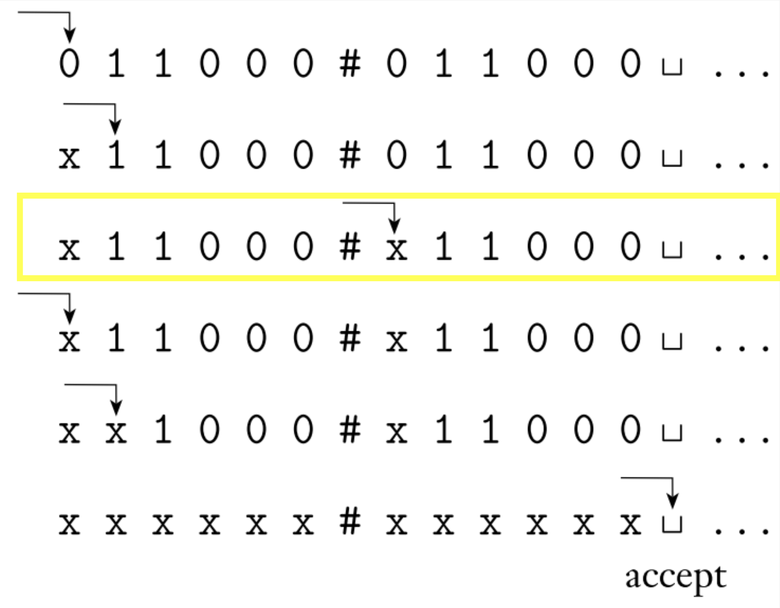


A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** ⊔,
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 $q_{reject} \neq q_{accept}$.

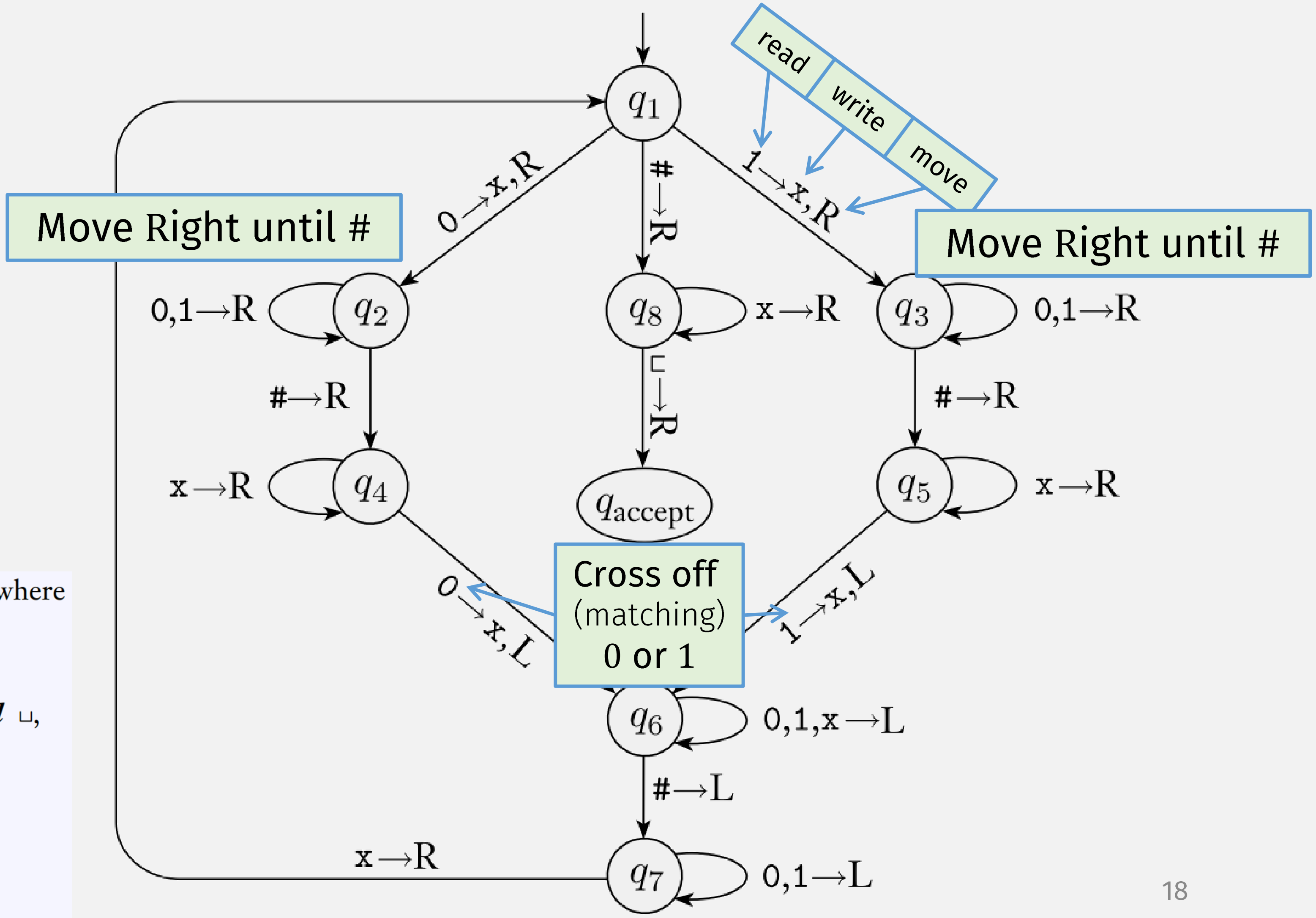
$B = \{w\#w \mid w \in \{0,1\}^*\}$

Formal Turing Machine Example



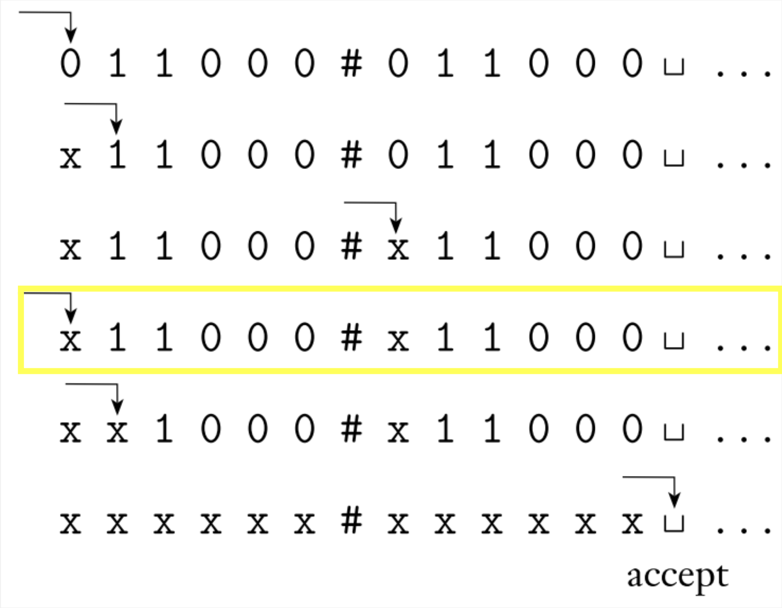
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

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



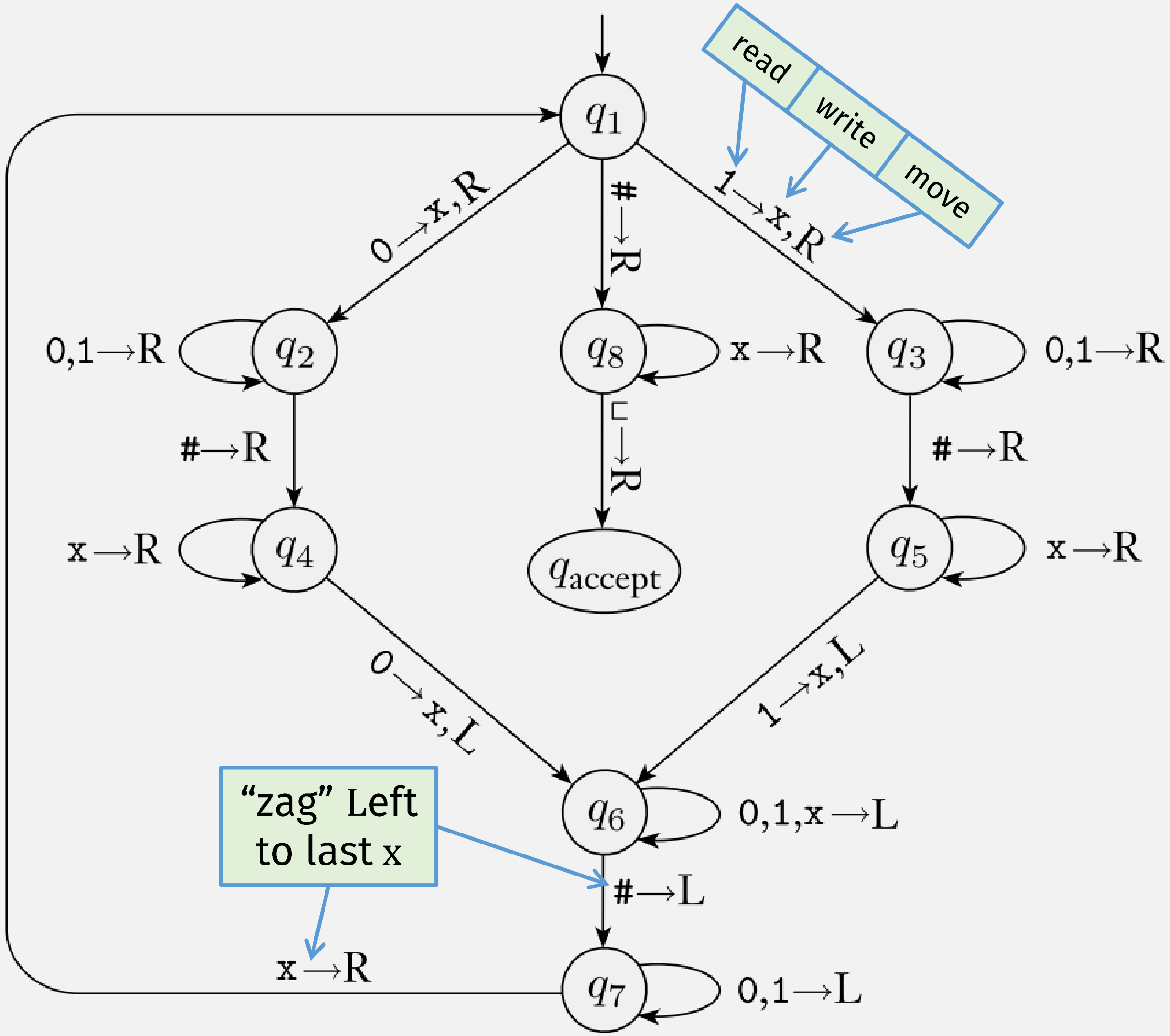
$$B = \{w\#w \mid w \in \{0,1\}^*\}$$

Formal Turing Machine Example



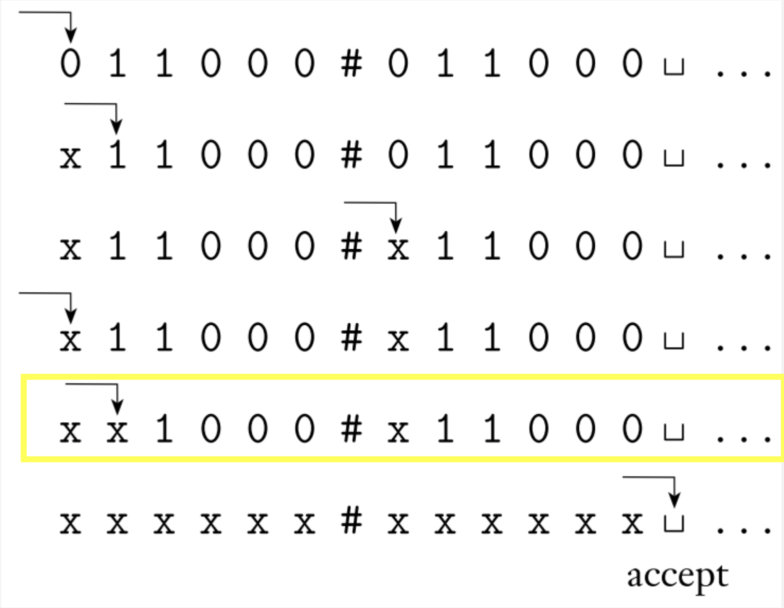
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

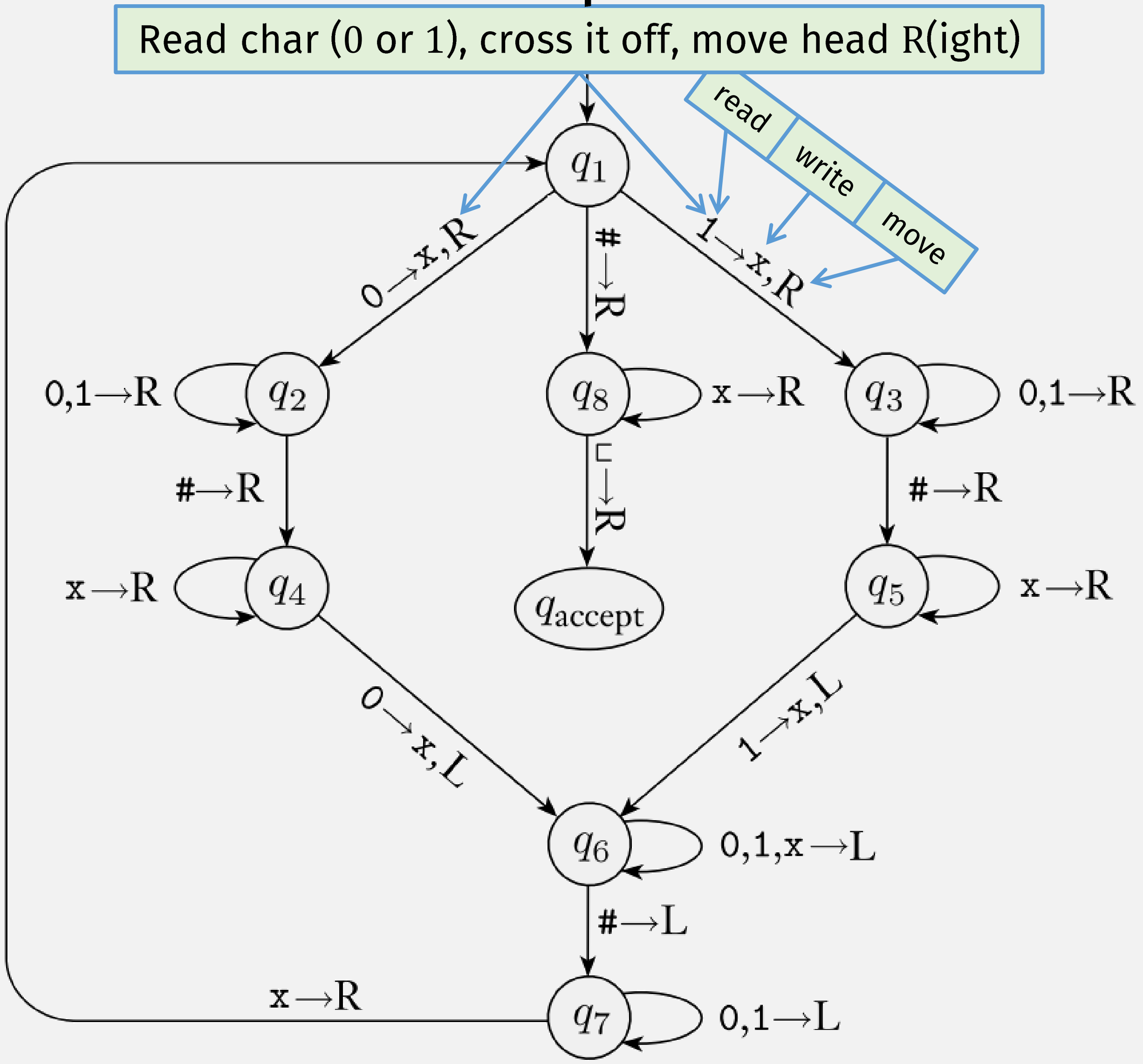
1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** ␣,
3. Γ is the tape alphabet, where ␣ $\in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta \colon Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ is the transition function,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



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

Formal Turing Machine Example



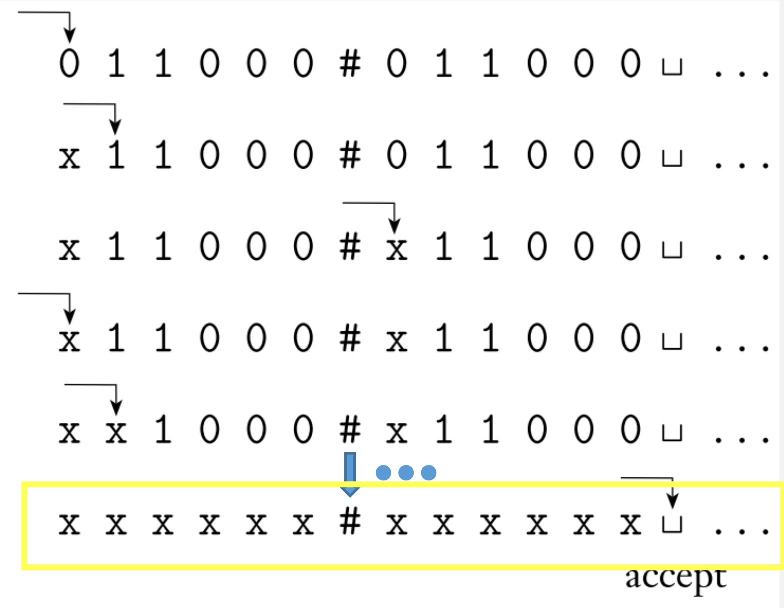


A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

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

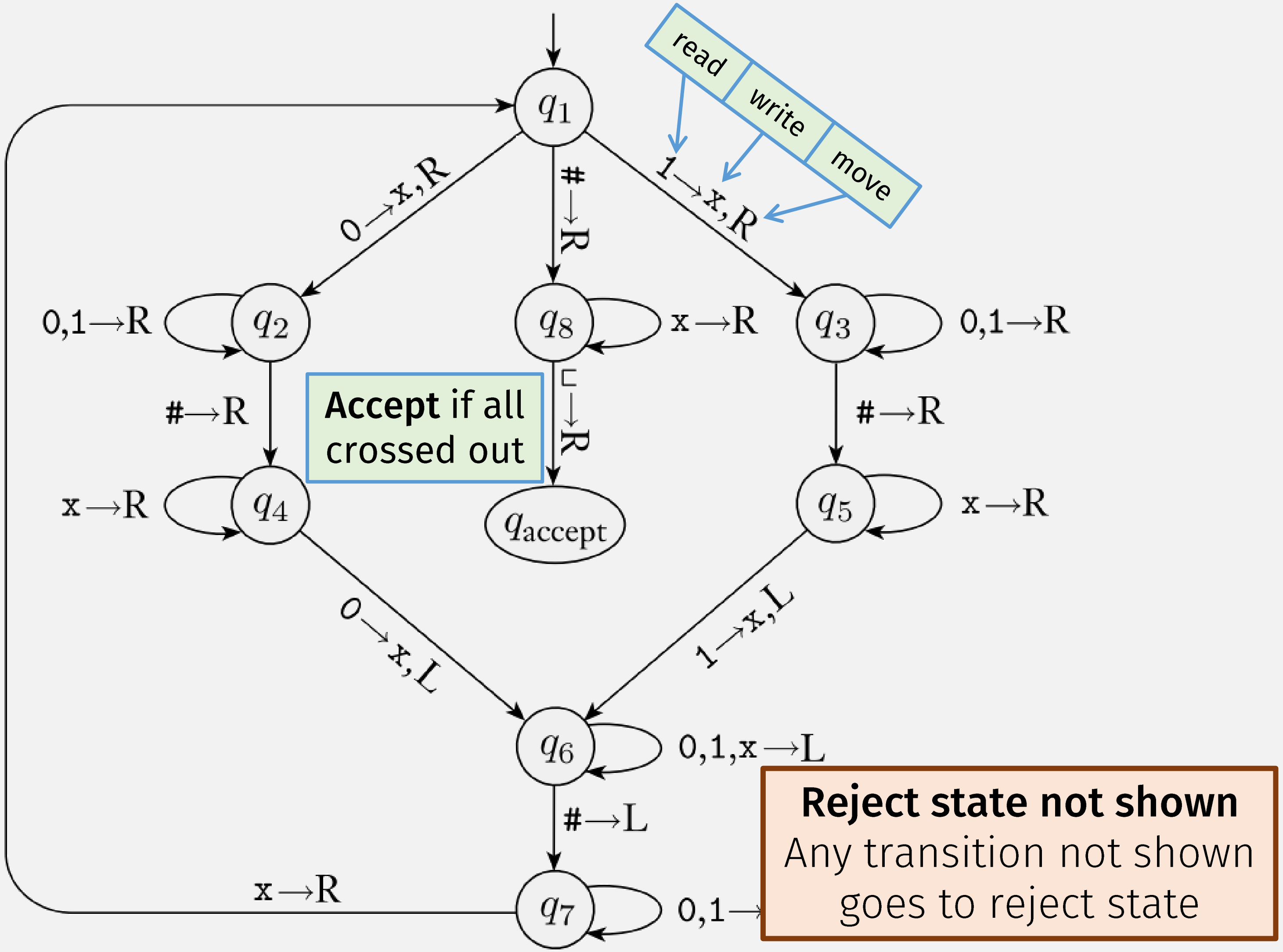
$B = \{w\#w \mid w \in \{0,1\}^*\}$

Formal Turing Machine Example



A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, 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 $q_{\text{reject}} \neq q_{\text{accept}}$.



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 ... # is found, *reject*. Cross off symbols as they ... track of which symbols correspond.

2. When all symbols to ... crossed off, check for any remaining ... 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
  - must be equivalent to a formal tuple

  Analogy:
  - informal TM ~ function definition in “high level” language
  - formal TM ~ function definition in bytecode or assembly

- Input
  - Must be named (like a function parameter), e.g., w
  - Assume string of chars from the alphabet (for now)

- An informal “step” represents a finite # 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

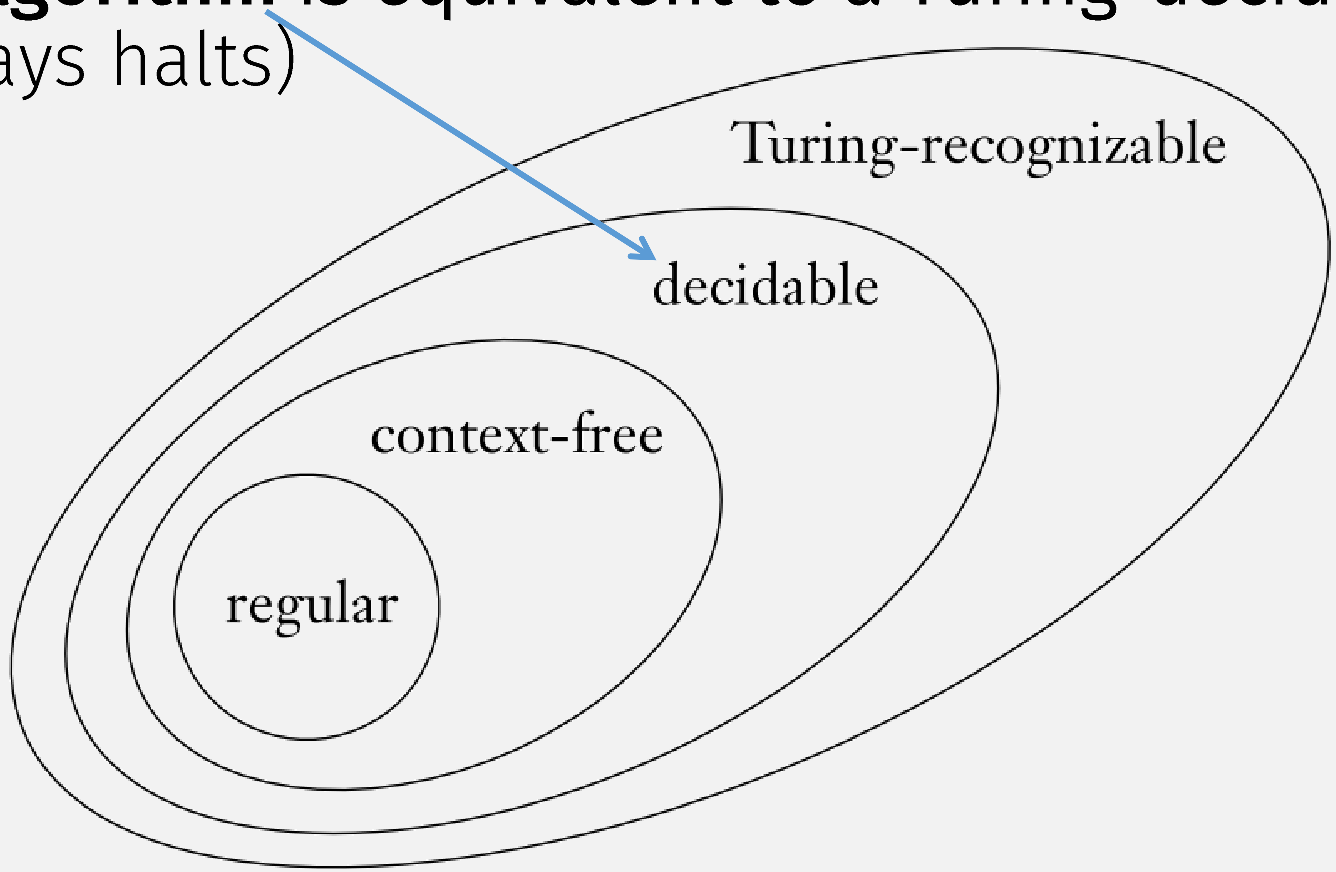
- A Turing Machine can <u>run forever</u>
  - E.g., the head can move back and forth in a loop
- Thus, there are <u>two classes of Turing Machines</u>:
  - A **recognizer** is a Turing Machine that may run forever (all possible TMs)
  - A **decider** is a Turing Machine that always halts.

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

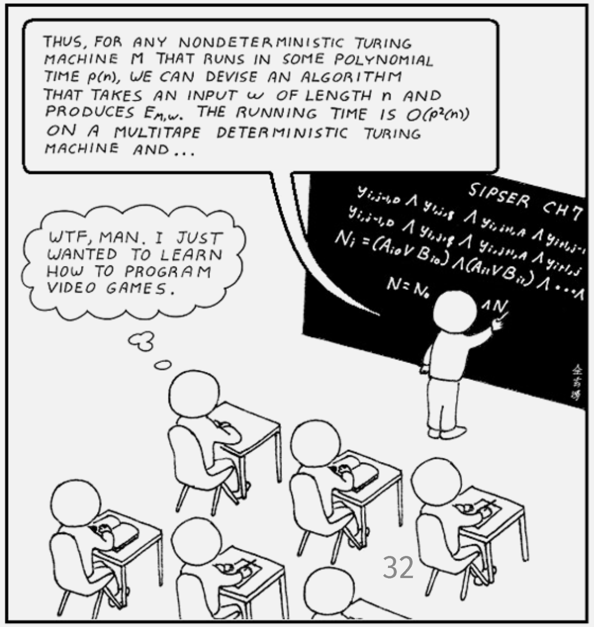
Call a language ***Turing-decidable*** or simply ***decidable*** if some Turing machine decides it.

Formal Definition of an "Algorithm"

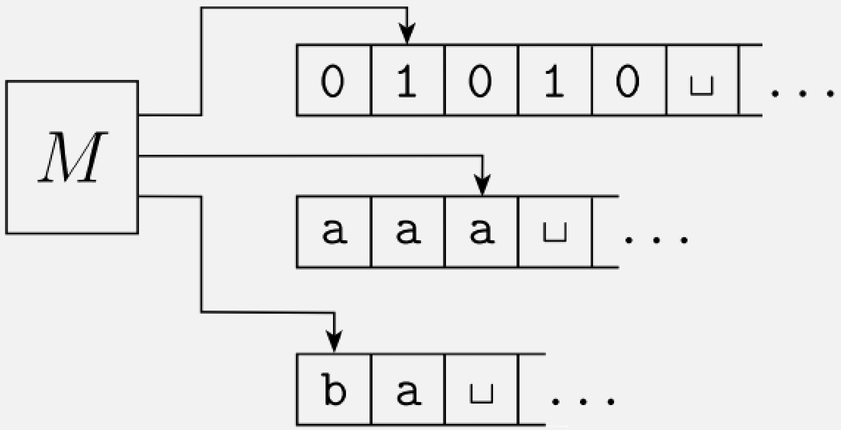
- An **algorithm** is equivalent to a Turing-decidable Language (always halts)



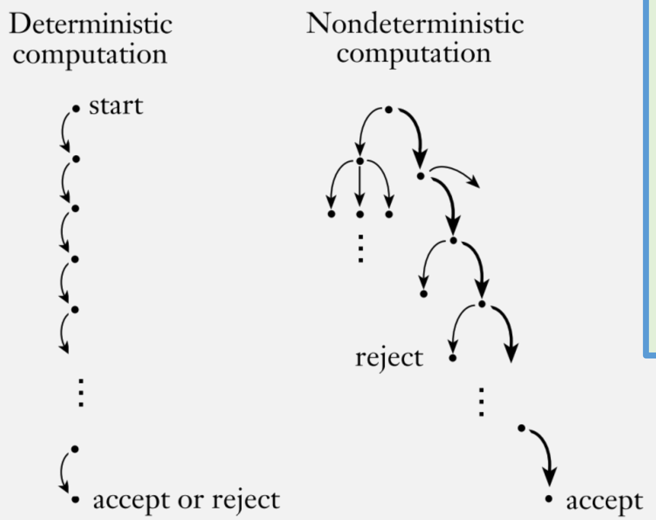
Turing Machine Variations



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?)

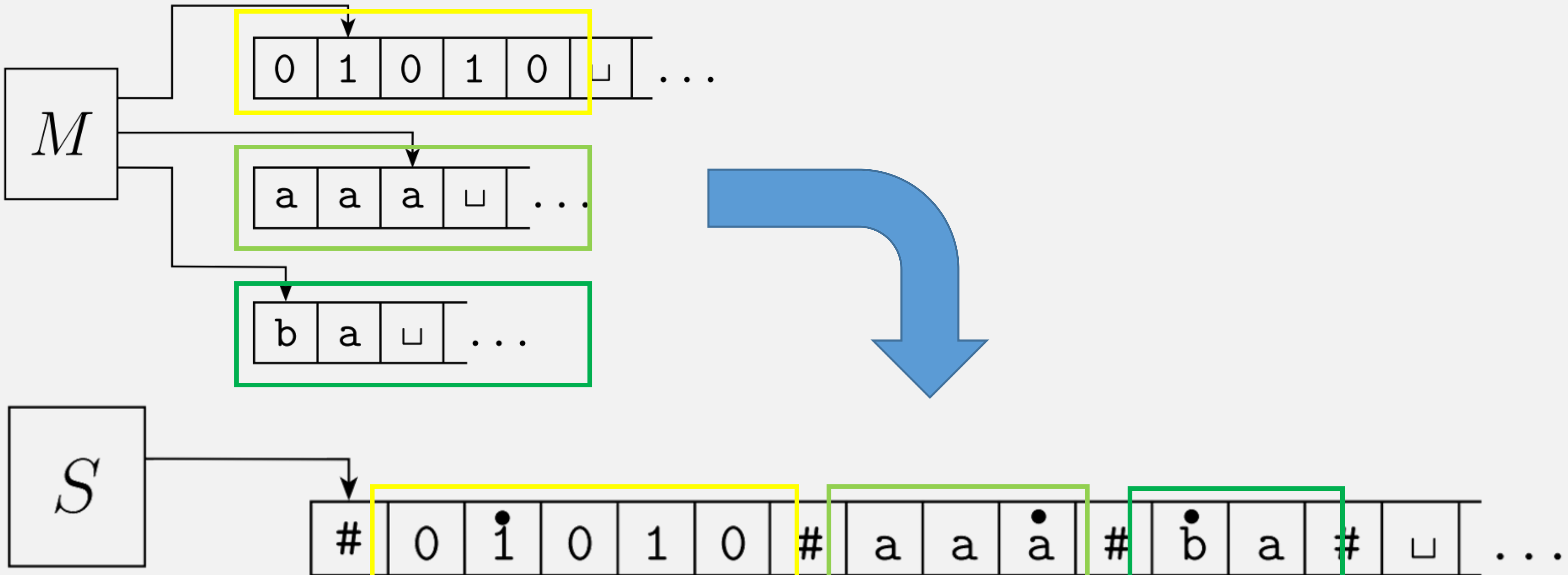
⇐ If a multi-tape TM recognizes a language,
then a single-tape TM recognizes the language

- 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 10/27

On gradescope