# CS170 Computation Theory

## Lecture 6

September 21, 2023

Megumi Ando

Tufts University | School of Engineering Computer Science

# Review: Formal Definition of TM

# Review: Formal Definition of TM

**Definition (p. 168):** A Turing Machine (TM) is a 7-tuple

# Review: Formal Definition of TM

**Definition (p. 168):** A Turing Machine (TM) is a 7-tuple

$Q = \{q_1, q_2, \ldots, q_n\}$, a finite set of states

# Review: Formal Definition of TM

**Definition (p. 168):** A Turing Machine (TM) is a 7-tuple

$Q = \{q_1, q_2, \ldots, q_n\}$, a finite set of states

$\Sigma$, a finite set of alphabet symbols NOT containing blank symbol "⊔"

# Review: Formal Definition of TM

**Definition (p. 168):** A Turing Machine (TM) is a 7-tuple

$Q = \{q_1, q_2, \ldots, q_n\}$, a finite set of states

$\Sigma$, a finite set of alphabet symbols NOT containing blank symbol "$\sqcup$"

$\Gamma$, a finite set of tape symbols, including all the symbols in $\Sigma$, i.e., $\Sigma \subseteq \Gamma$

# Review: Formal Definition of TM

**Definition (p. 168):** A Turing Machine (TM) is a 7-tuple

$Q = \{q_1, q_2, \ldots, q_n\}$, a finite set of states

$\Sigma$, a finite set of alphabet symbols NOT containing blank symbol "$\sqcup$"

$\Gamma$, a finite set of tape symbols, including all the symbols in $\Sigma$, i.e., $\Sigma \subseteq \Gamma$

$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\text{Left}, \text{Right}\}$, a transition function

# Review: Formal Definition of TM

**Definition (p. 168):** A Turing Machine (TM) is a 7-tuple

$Q = \{q_1, q_2, \ldots, q_n\}$, a finite set of states

$\Sigma$, a finite set of alphabet symbols NOT containing blank symbol "$\sqcup$"

$\Gamma$, a finite set of tape symbols, including all the symbols in $\Sigma$, i.e., $\Sigma \subseteq \Gamma$

$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\text{Left}, \text{Right}\}$, a transition function

$q_1 \in Q$, a start state

# Review: Formal Definition of TM

**Definition (p. 168):** A Turing Machine (TM) is a 7-tuple

$Q = \{q_1, q_2, \ldots, q_n\}$, a finite set of states

$\Sigma$, a finite set of alphabet symbols NOT containing blank symbol "⊔"

$\Gamma$, a finite set of tape symbols, including all the symbols in $\Sigma$, i.e., $\Sigma \subseteq \Gamma$

$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\text{Left}, \text{Right}\}$, a transition function

$q_1 \in Q$, a start state

$q_{\text{accept}} \in Q$, an accept state

# Review: Formal Definition of TM

**Definition (p. 168):** A Turing Machine (TM) is a 7-tuple

$Q = \{q_1, q_2, \ldots, q_n\}$, a finite set of states

$\Sigma$, a finite set of alphabet symbols NOT containing blank symbol "⊔"

$\Gamma$, a finite set of tape symbols, including all the symbols in $\Sigma$, i.e., $\Sigma \subseteq \Gamma$

$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\text{Left}, \text{Right}\}$, a transition function

$q_1 \in Q$, a start state

$q_{\text{accept}} \in Q$, an accept state

$q_{\text{reject}} \in Q$, a reject state

# Review: Formal Definition of TM

**Definition (p. 168):** A Turing Machine (TM) is a 7-tuple

$Q = \{q_1, q_2, \ldots, q_n\}$, a finite set of states

$\Sigma$, a finite set of alphabet symbols NOT containing blank symbol "$\sqcup$"

$\Gamma$, a finite set of tape symbols, including all the symbols in $\Sigma$, i.e., $\Sigma \subseteq \Gamma$

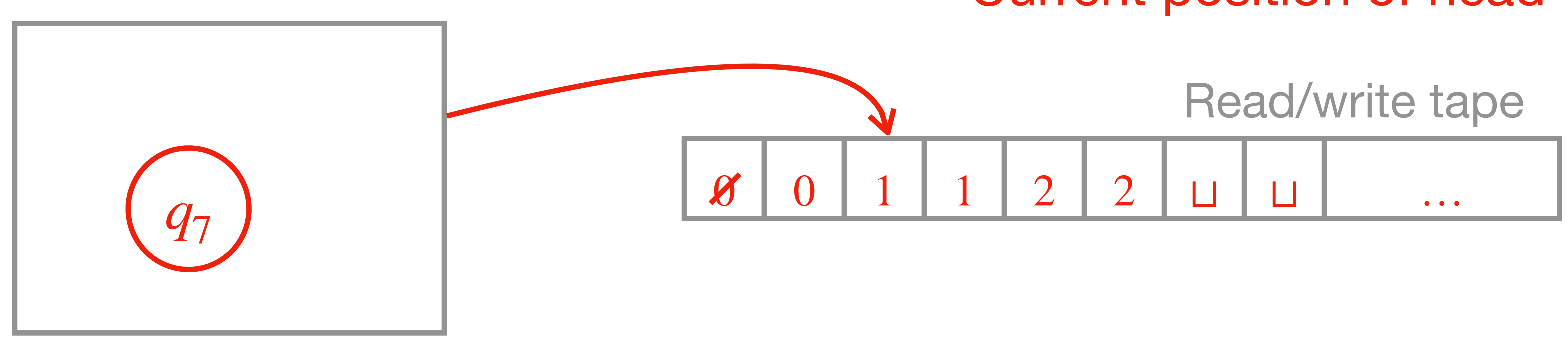
$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\text{Left}, \text{Right}\}$, a transition function

$q_1 \in Q$, a start state

$q_{\text{accept}} \in Q$, an accept state

$q_{\text{reject}} \in Q$, a reject state

Configuration =
- Current state,
- Current tape contents,
- Current position of head




Control

"$\emptyset 0 q_7 1122$"

# Review: Formal Definition of TM

**Definition (p. 168):** A <u>Turing Machine</u> (TM) is a 7-tuple

$Q = \{q_1, q_2, \dots, q_n\}$, a finite set of states

$\Sigma$, a finite set of alphabet symbols $\boxed{\text{NOT containing blank symbol “}\sqcup\text{”}}$

$\boxed{\Gamma\text{, a finite set of tape symbols, including all the symbols in } \Sigma\text{, i.e., } \Sigma \subseteq \Gamma}$

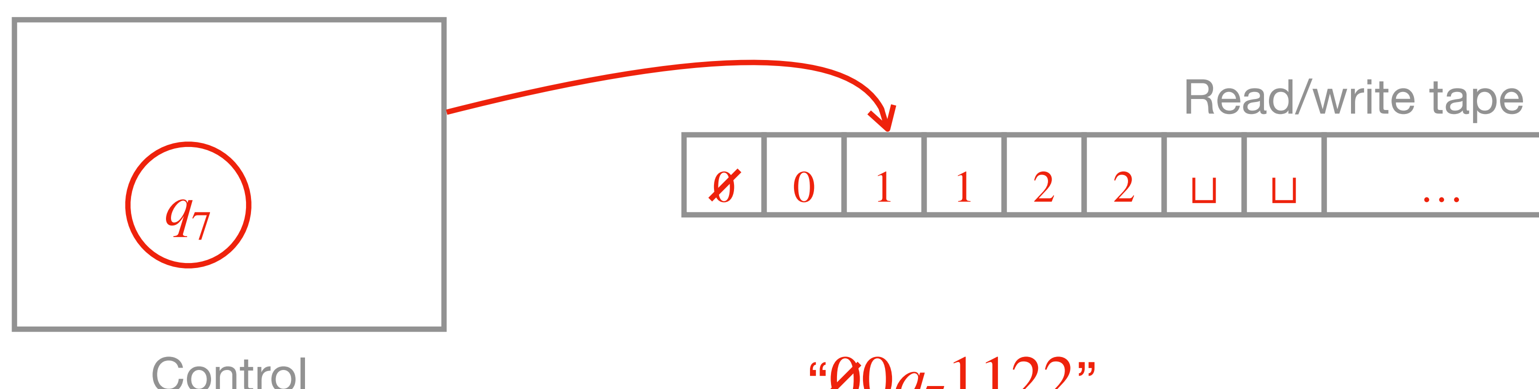
$\boxed{\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\text{Left}, \text{Right}\}\text{, a transition function}}$

$q_1 \in Q$, a start state

E.g., $\delta(q_7, 1) = (q_4, \cancel{1}, \text{R})$

$\boxed{q_{\text{accept}} \in Q\text{, an accept state}}$

$\boxed{q_{\text{reject}} \in Q\text{, a reject state}}$




“$\cancel{0}0q_71122$”

# Review: Formal Definition of TM

**Definition (p. 168):** A <u>Turing Machine</u> (TM) is a 7-tuple

$Q = \{q_1, q_2, \ldots, q_n\}$, a finite set of states

$\Sigma$, a finite set of alphabet symbols NOT containing blank symbol "$\sqcup$"

$\Gamma$, a finite set of tape symbols, including all the symbols in $\Sigma$, i.e., $\Sigma \subseteq \Gamma$

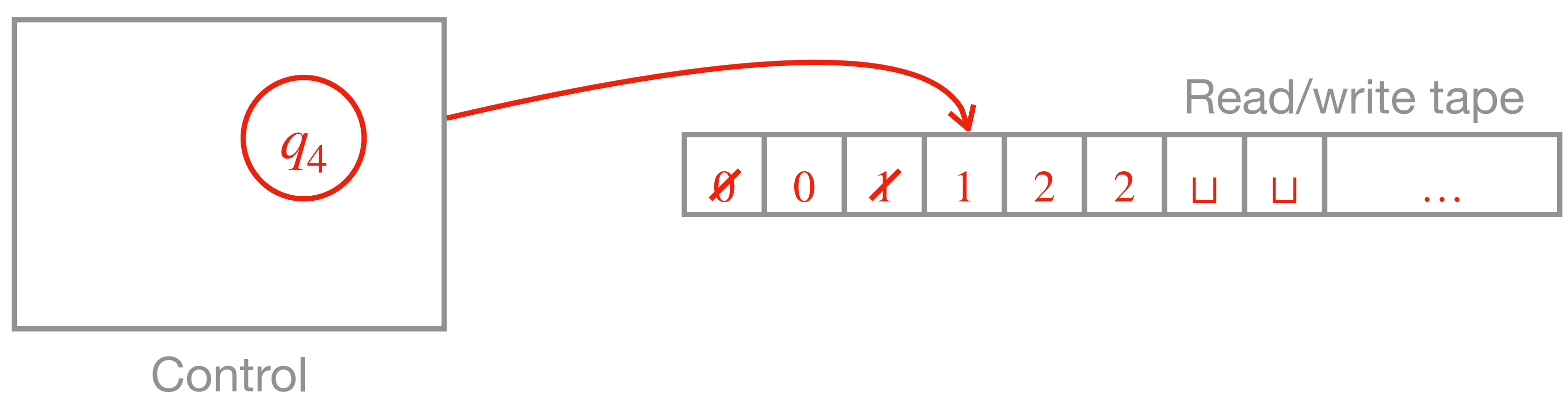
$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\text{Left}, \text{Right}\}$, a transition function

$q_1 \in Q$, a start state

E.g., $\delta(q_7, 1) = (q_4, \cancel{1}, \text{R})$

$q_{\text{accept}} \in Q$, an accept state

$q_{\text{reject}} \in Q$, a reject state



# Review: Formal Definition of TM

**Definition (p. 168):** A Turing Machine (TM) is a 7-tuple

$Q = \{q_1, q_2, \ldots, q_n\}$, a finite set of states

$\Sigma$, a finite set of alphabet symbols NOT containing blank symbol "$\sqcup$"

$\Gamma$, a finite set of tape symbols, including all the symbols in $\Sigma$, i.e., $\Sigma \subseteq \Gamma$

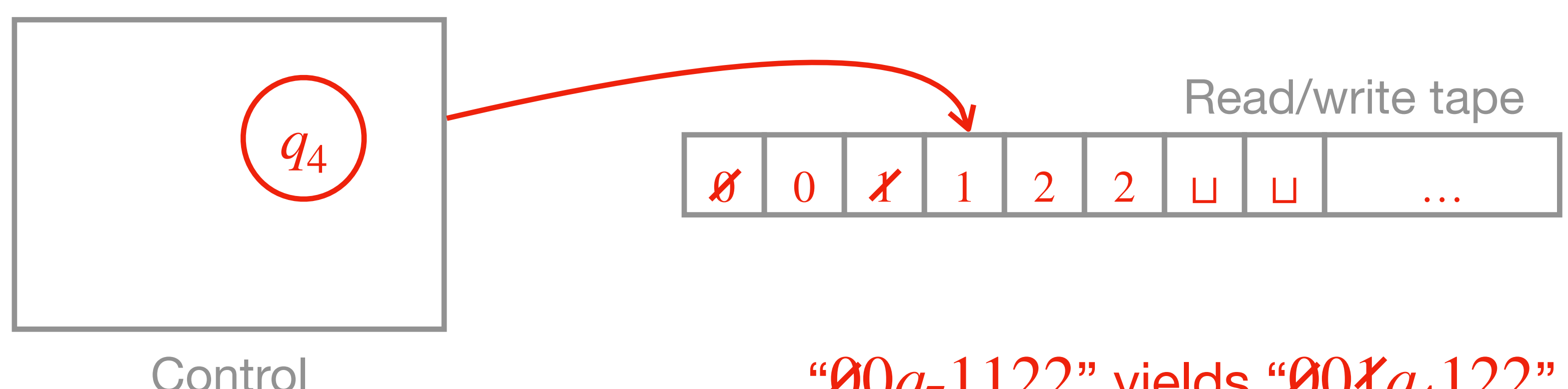
$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\text{Left}, \text{Right}\}$, a transition function

$q_1 \in Q$, a start state

E.g., $\delta(q_7, 1) = (q_4, \cancel{1}, \text{R})$

$q_{\text{accept}} \in Q$, an accept state

$q_{\text{reject}} \in Q$, a reject state




"$\cancel{0}0q_7 1122$" yields "$\cancel{0}0\cancel{1}q_4 122$"

# Review: T-recognizability

# Review: T-recognizability

**Definition (p. 169):** A TM $M$ accepts input $w = w_1 w_2 \ldots w_k$ if exists configurations $C_1, C_2 \ldots, C_\ell$ such that

# Review: T-recognizability

**Definition (p. 169):** A TM $M$ accepts input $w = w_1 w_2 \ldots w_k$ if exists configurations $C_1, C_2 \ldots, C_\ell$ such that

- $C_1$ is the start configuration ($q_1 w_1 w_2 \ldots w_k$)

# Review: T-recognizability

**Definition (p. 169):** A TM $M$ accepts input $w = w_1 w_2 \ldots w_k$ if exists configurations $C_1, C_2 \ldots, C_\ell$ such that

- $C_1$ is the start configuration ($q_1 w_1 w_2 \ldots w_k$)
- Each $C_i$ yields $C_{i+1}$

# Review: T-recognizability

**Definition (p. 169):** A TM $M$ accepts input $w = w_1w_2 \ldots w_k$ if exists configurations $C_1, C_2 \ldots, C_\ell$ such that

- $C_1$ is the start configuration ($q_1w_1w_2\ldots w_k$)
- Each $C_i$ yields $C_{i+1}$
- $C_\ell$ is an accepting configuration (i.e., with $q_{\text{accept}}$)

# Review: T-recognizability

**Definition (p. 169):** A TM $M$ accepts input $w = w_1 w_2 \dots w_k$ if exists configurations $C_1, C_2 \dots, C_\ell$ such that

- $C_1$ is the start configuration ($q_1 w_1 w_2 \dots w_k$)
- Each $C_i$ yields $C_{i+1}$
- $C_\ell$ is an accepting configuration (i.e., with $q_{\text{accept}}$)

**Definition (p. 170):** Let $M$ be a TM. Let $A = \{w \mid M \text{ accepts } w\}$. Then, $A$ is the language recognized by $M$, i.e., $A = L(M)$.

# Review: T-recognizability

**Definition (p. 169):** A TM $M$ <u>accepts</u> input $w = w_1 w_2 \ldots w_k$ if exists configurations $C_1, C_2 \ldots, C_\ell$ such that

- $C_1$ is the start configuration ($q_1 w_1 w_2 \ldots w_k$)
- Each $C_i$ yields $C_{i+1}$
- $C_\ell$ is an accepting configuration (i.e., with $q_{\text{accept}}$)

**Definition (p. 170):** Let $M$ be a TM. Let $A = \{w \mid M \text{ accepts } w\}$. Then, <u>$A$ is the language recognized by $M$</u>, i.e., <u>$A = L(M)$</u>.

**Definition (p. 170):** A language is <u>T-recognizable</u> if there is a TM that recognizes it.

# Review: T-decidability

The TM $M$ has 3 possible outcomes for input $w$:

- Accept (by entering $q_{\text{accept}}$)
- Reject by entering $q_{\text{reject}}$
- Reject by looping (running forever)

**Definition (p. 170):** A TM $M$ is a decider if it halts on all inputs.

**Definition (p. 170):** A language $A$ is T-decidable (or just decidable) if $A = L(M)$ for some TM decider $M$.

# Today’s Topics

- ~~Pumping Lemma for CFLs~~
- ~~Turing Machines (TMs)~~
- ~~T-recognizability and T-decidability~~

- Robustness of TMs
- Church-Turing Thesis

# Robustness of TM Model

# Robustness of TM Model

- Captures our intuition of computation

# Robustness of TM Model

- Captures our intuition of computation
- Mathematically precise

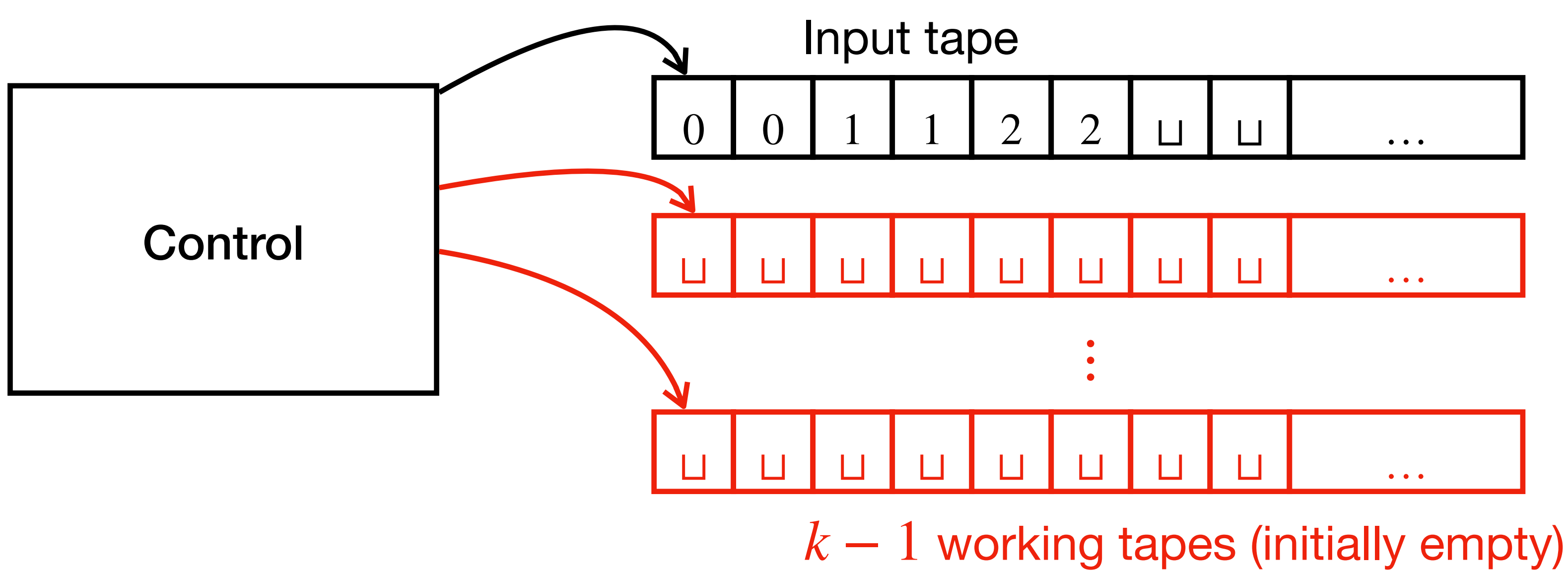
# Robustness of TM Model

- Captures our intuition of computation
- Mathematically precise
- Simple

# Robustness of TM Model

- Captures our intuition of computation
- Mathematically precise
- Simple
- <u>Robust</u>: equivalent to other TM variants (e.g., TMs with multiple tapes, Nondeterministic TMs)

# Multi-tape TMs




**Definition (p. 176):** A multi-tape Turing Machine is a TM with $k$ tapes. The transition function $\delta : Q \times \Gamma^k \mapsto Q \times \Gamma^k \times \{\mathsf{L}, \mathsf{R}\}^k$ allows reading, writing, and moving the heads on some or all of the tapes.

**Theorem:** The language $A$ is T-recognizable iff some multi-tape TM recognizes it.

# Proof of Equivalence of Multi-tape TMs and TMs

We must prove both directions:

- $A$ is T-recognizable $\Longrightarrow$ multi-tape TM recognizes $A$.
- Multi-tape TM recognizes $A \Longrightarrow A$ is T-recognizable.

# Proof of Equivalence of Multi-tape TMs and TMs

We must prove both directions:

- $A$ is T-recognizable $\Longrightarrow$ multi-tape TM recognizes $A$.

  Vacuously true.

- Multi-tape TM recognizes $A \Longrightarrow A$ is T-recognizable.

# Proof of Equivalence of Multi-tape TMs and TMs

We must prove both directions:

- $A$ is T-recognizable $\Longrightarrow$ multi-tape TM recognizes $A$.

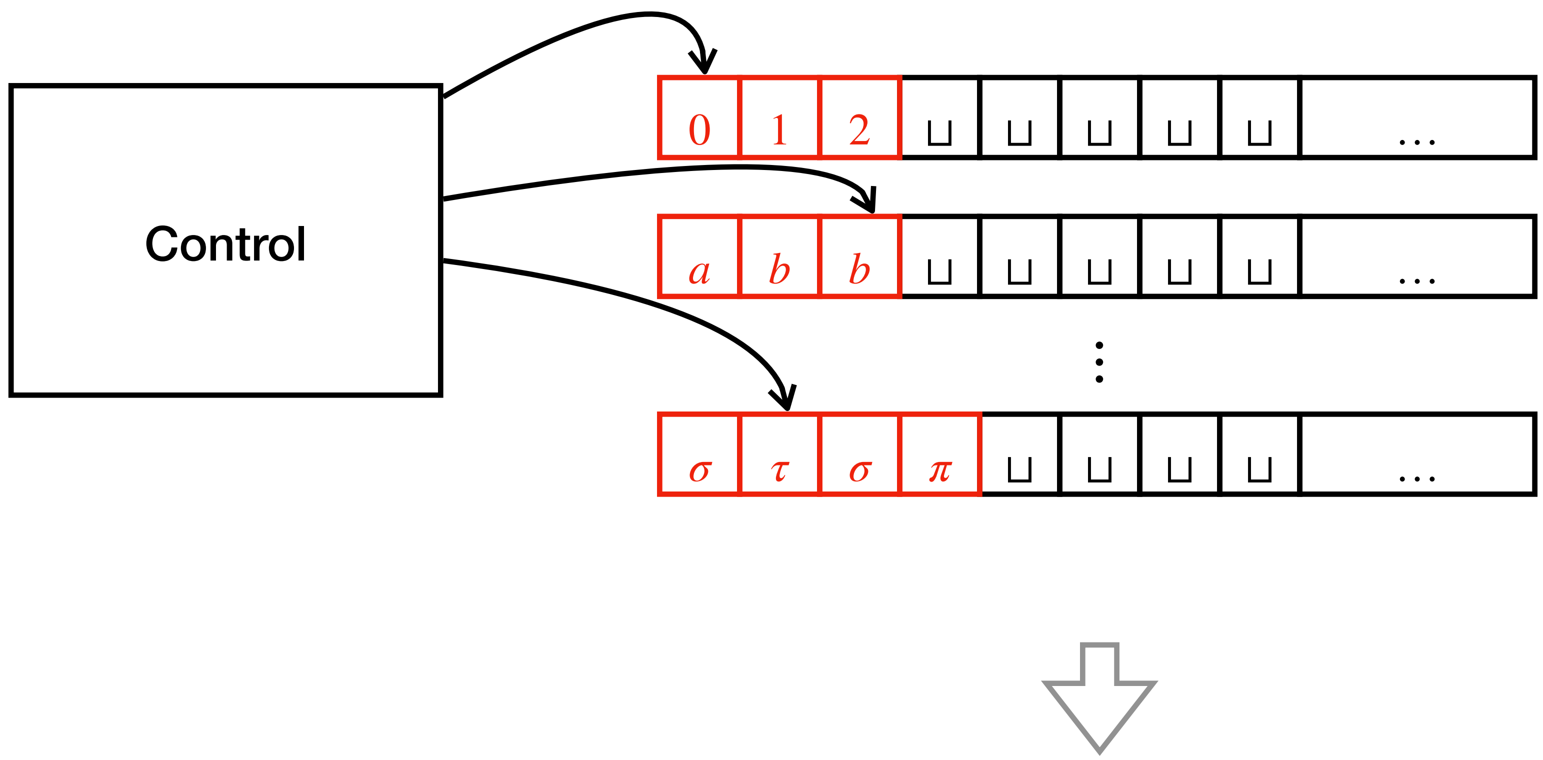
  Vacuously true.

- Multi-tape TM recognizes $A \Longrightarrow A$ is T-recognizable.

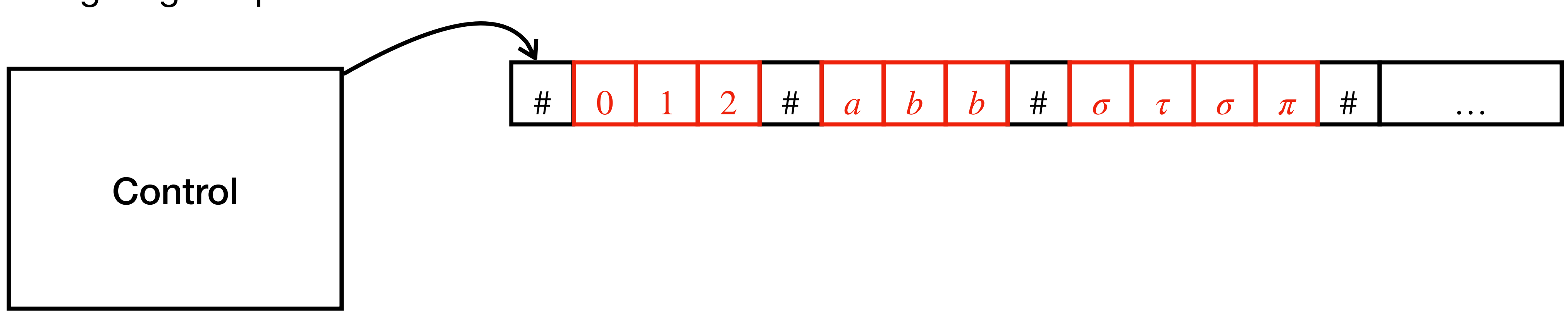
  Need to show.

# Proof that Multi-tape TM $\Longrightarrow$ Single-tape TM

Multi-tape TM $M$:

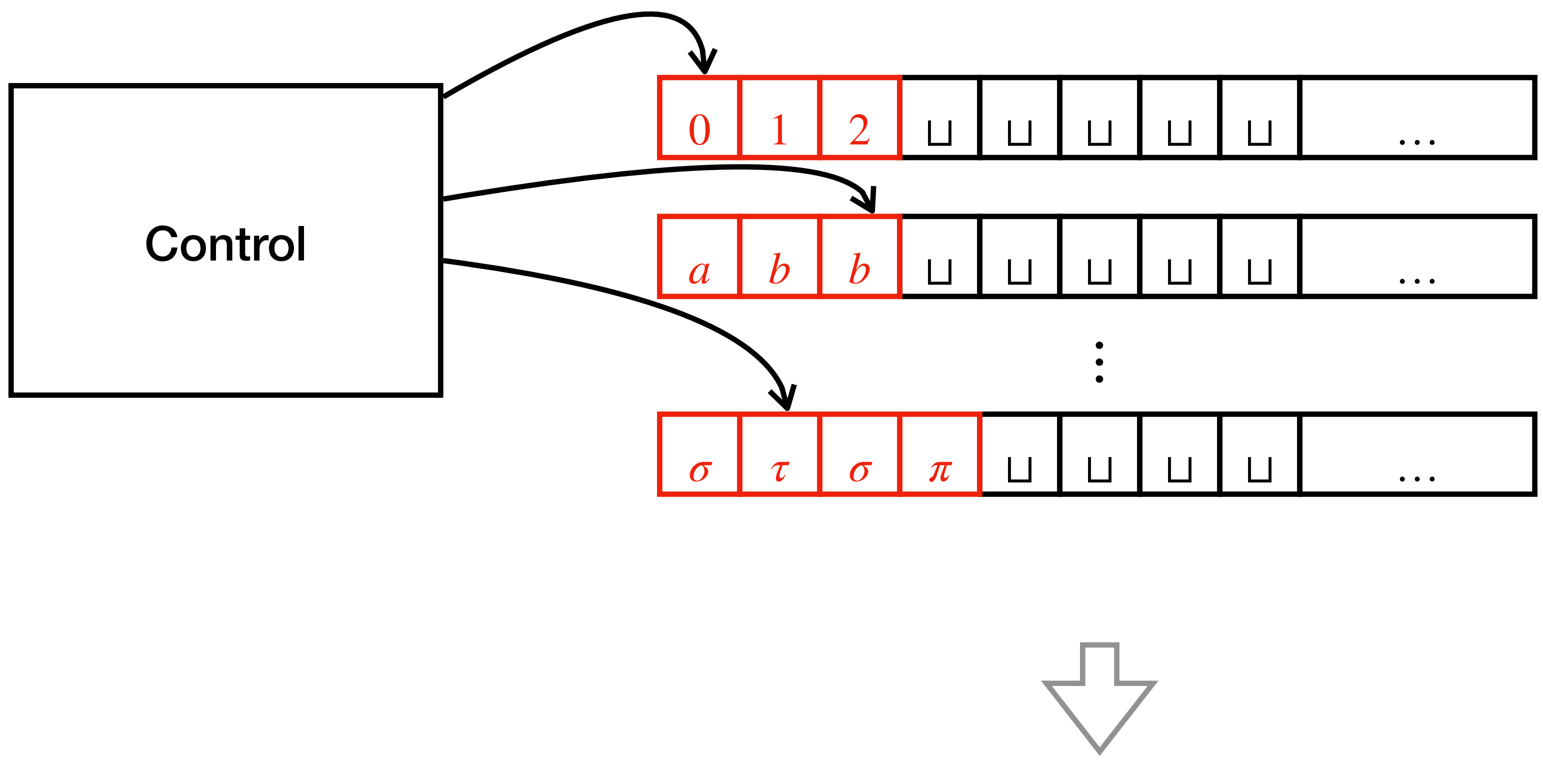



"Simulate" $M$ using single-tape TM $M'$:

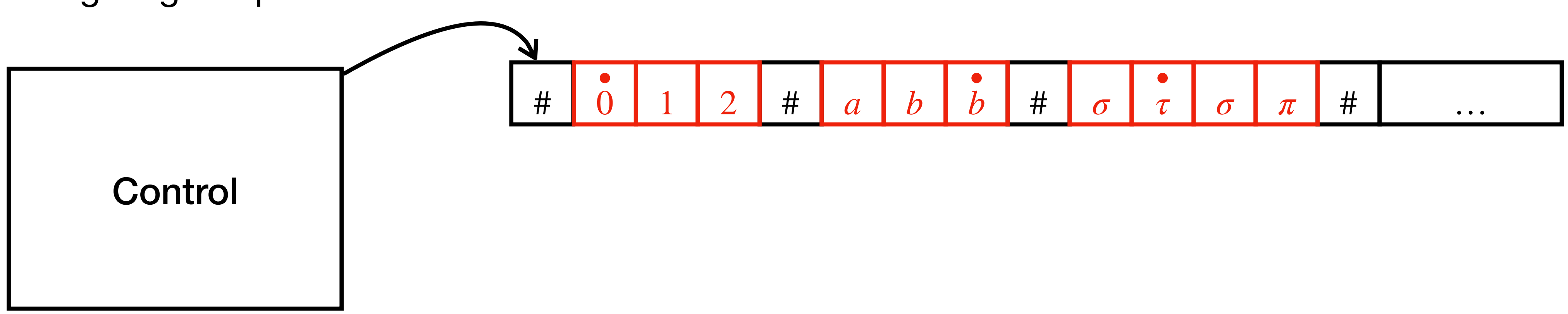


# Proof that Multi-tape TM $\implies$ Single-tape TM

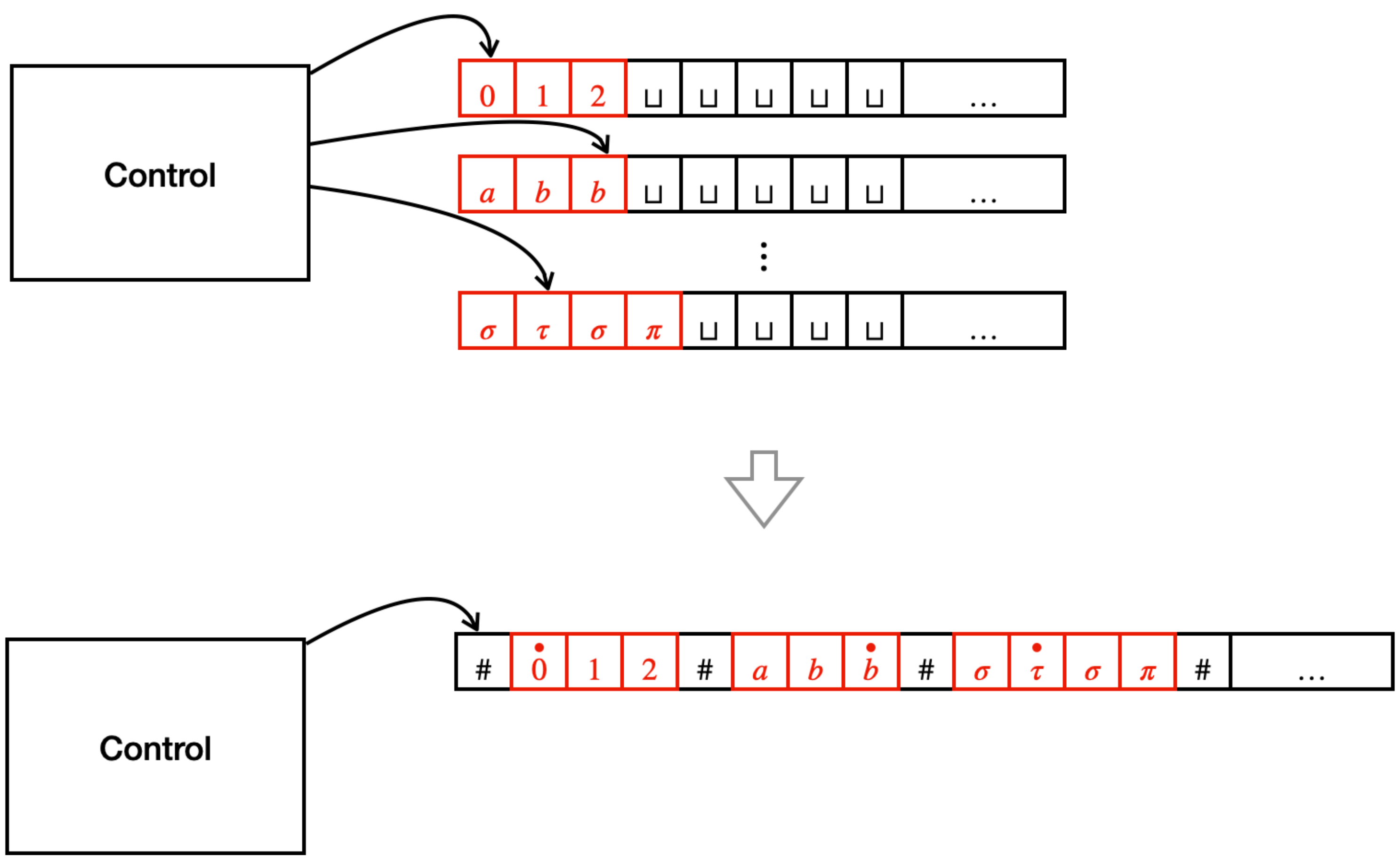
Multi-tape TM $M$:




“Simulate” $M$ using single-tape TM $M'$:

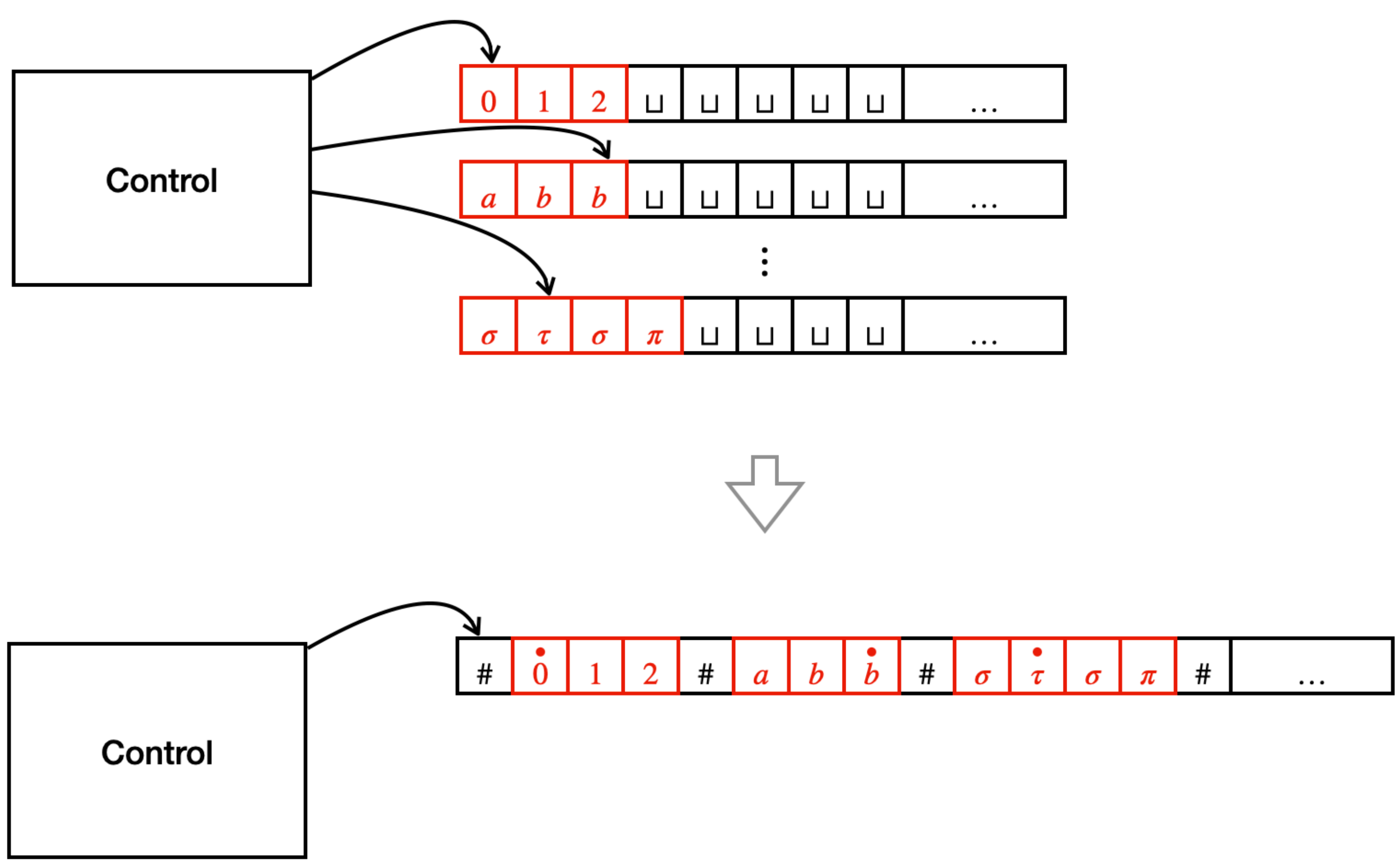


# Description of $M'$

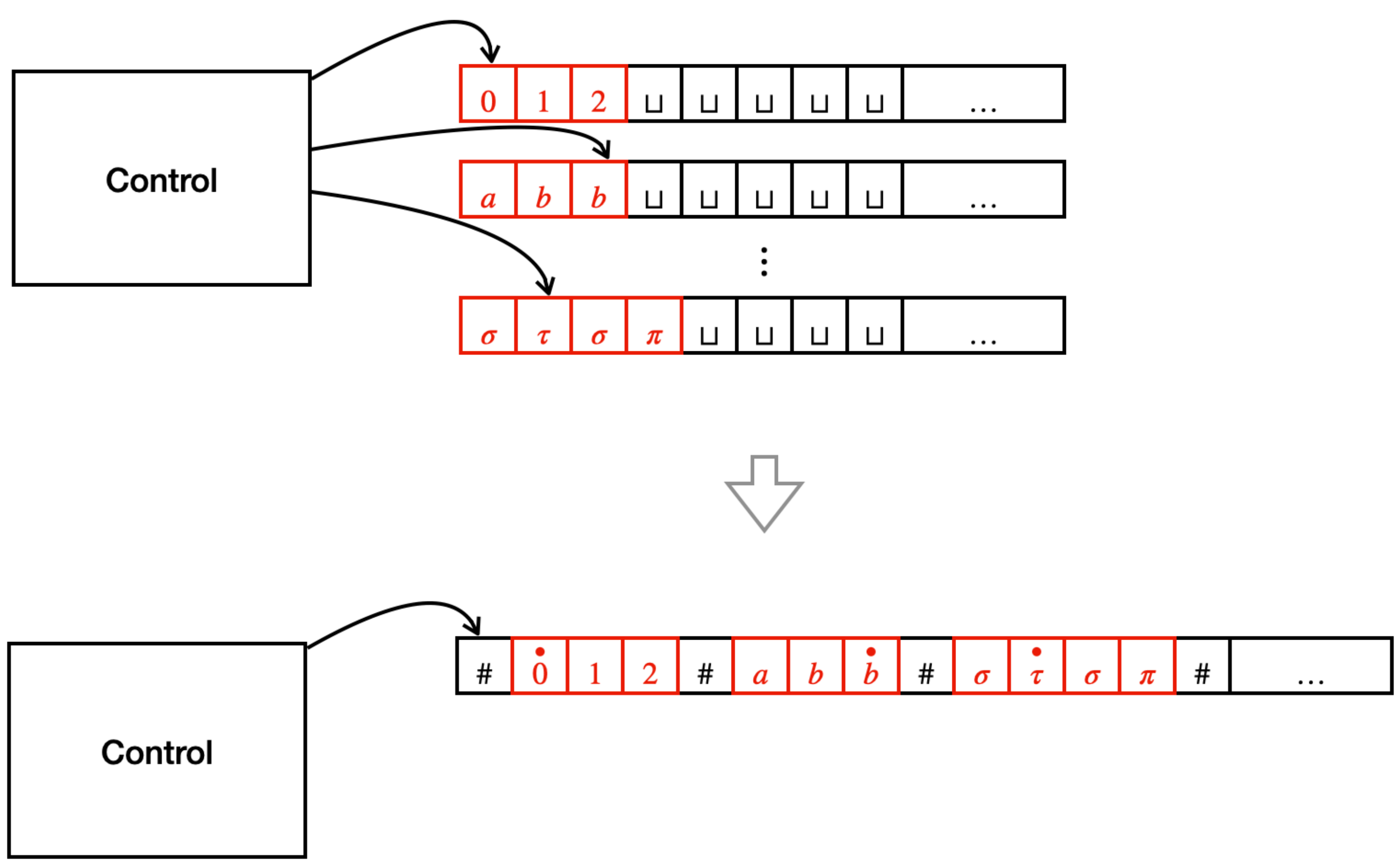


# Description of $M'$

$M'$ = “On input $w = w_1 w_2 \ldots w_n$:



# Description of $M'$




$M' =$ "On input $w = w_1 w_2 \ldots w_n$:

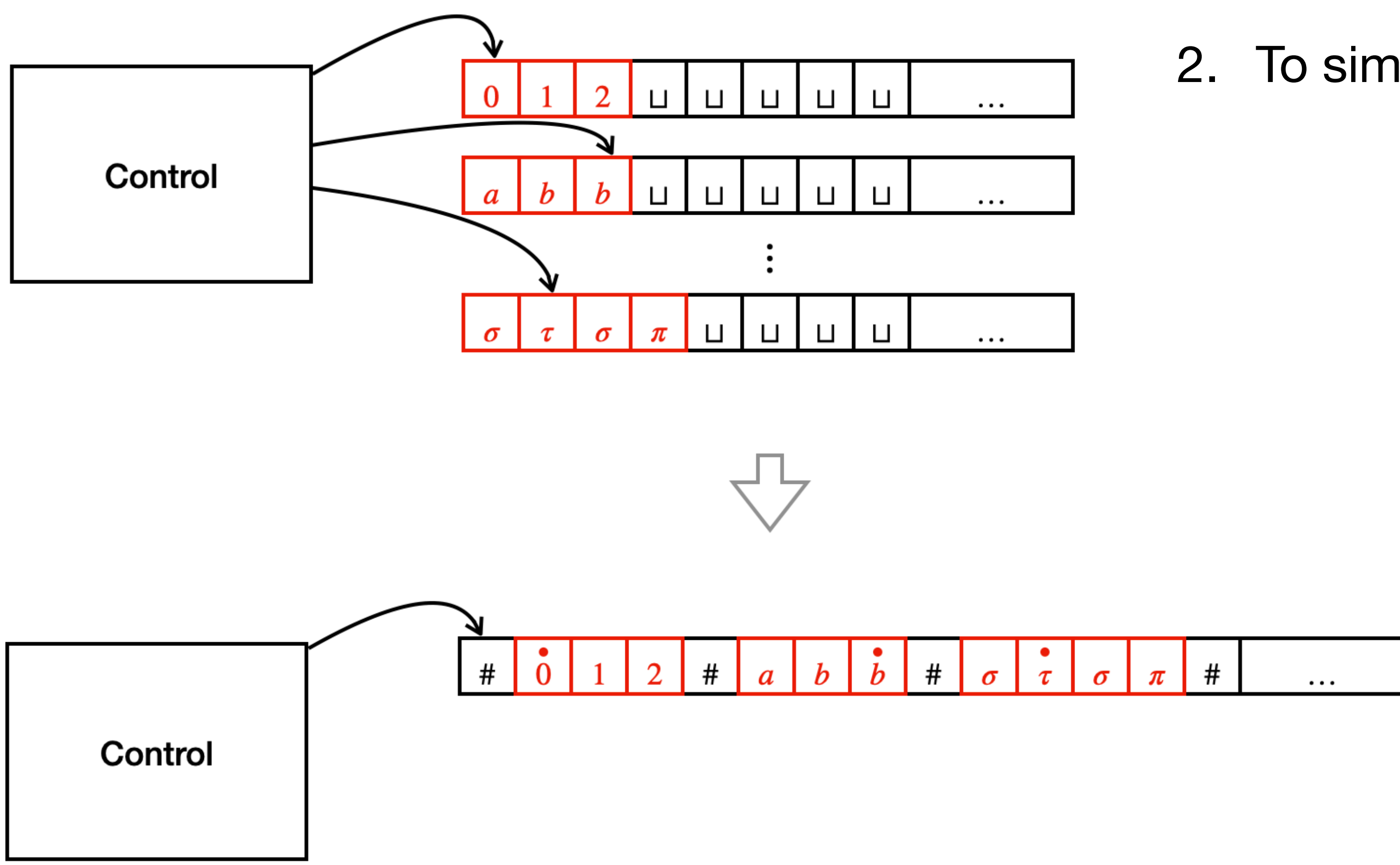
1. Initiate tape: $\# \dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$

# Description of $M'$

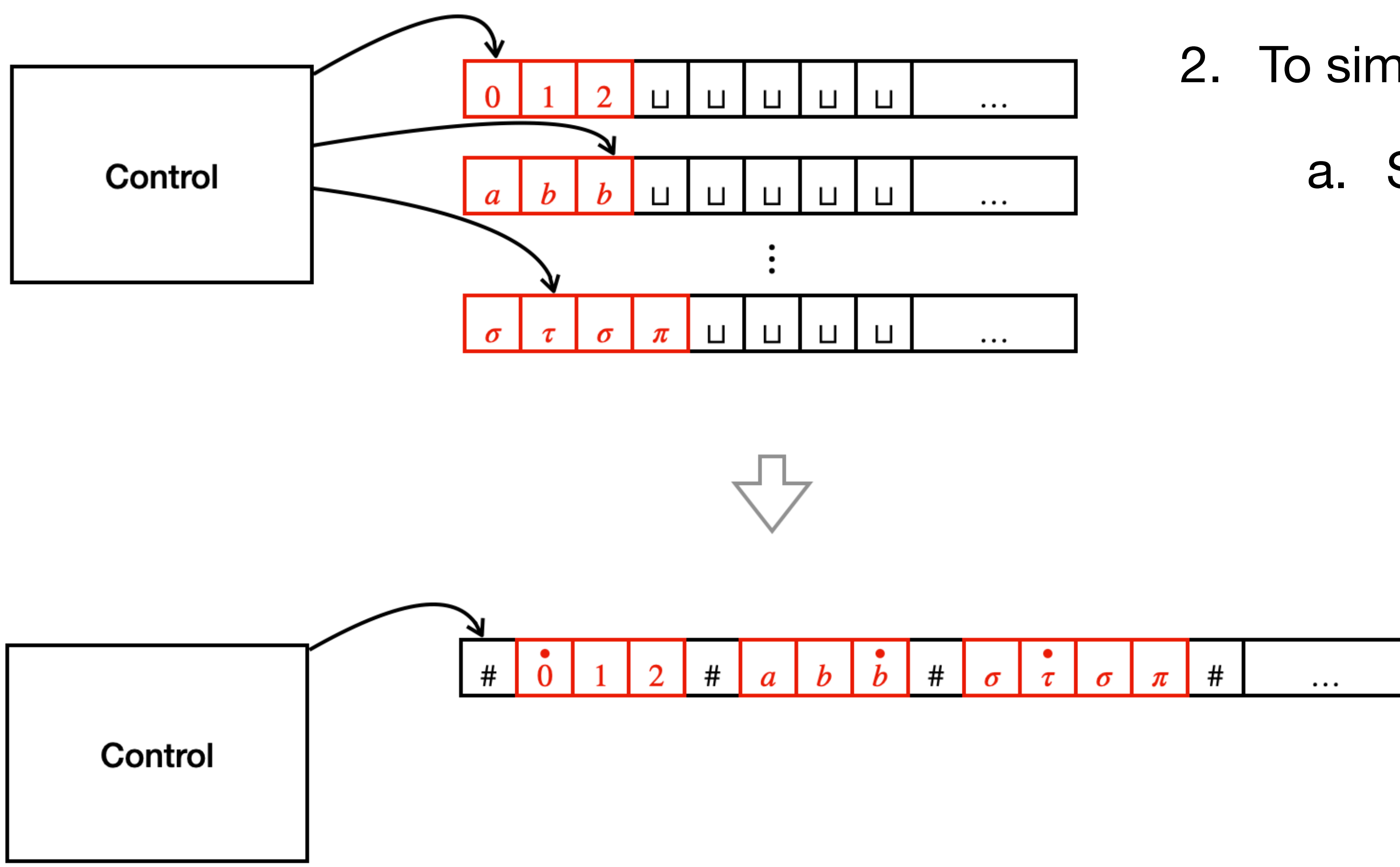
$M' =$ "On input $w = w_1 w_2 \ldots w_n$:

1. Initiate tape: $\# \dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$

2. To simulate a step in $M$:



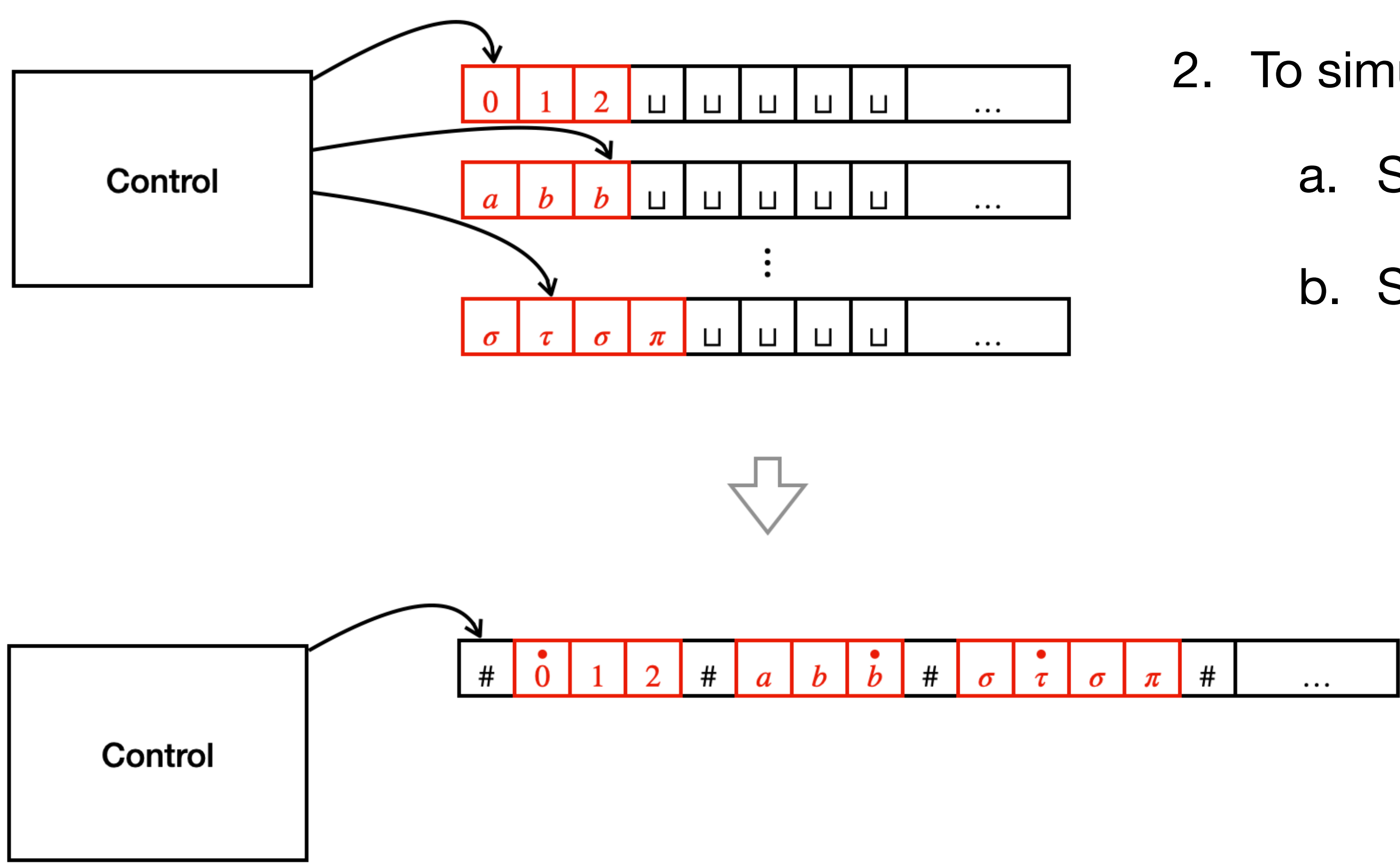
# Description of $M'$




$M' =$ "On input $w = w_1 w_2 \ldots w_n$:

1. Initiate tape: $\#\dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$
2. To simulate a step in $M$:
   a. Scan entire tape to find dotted symbols.

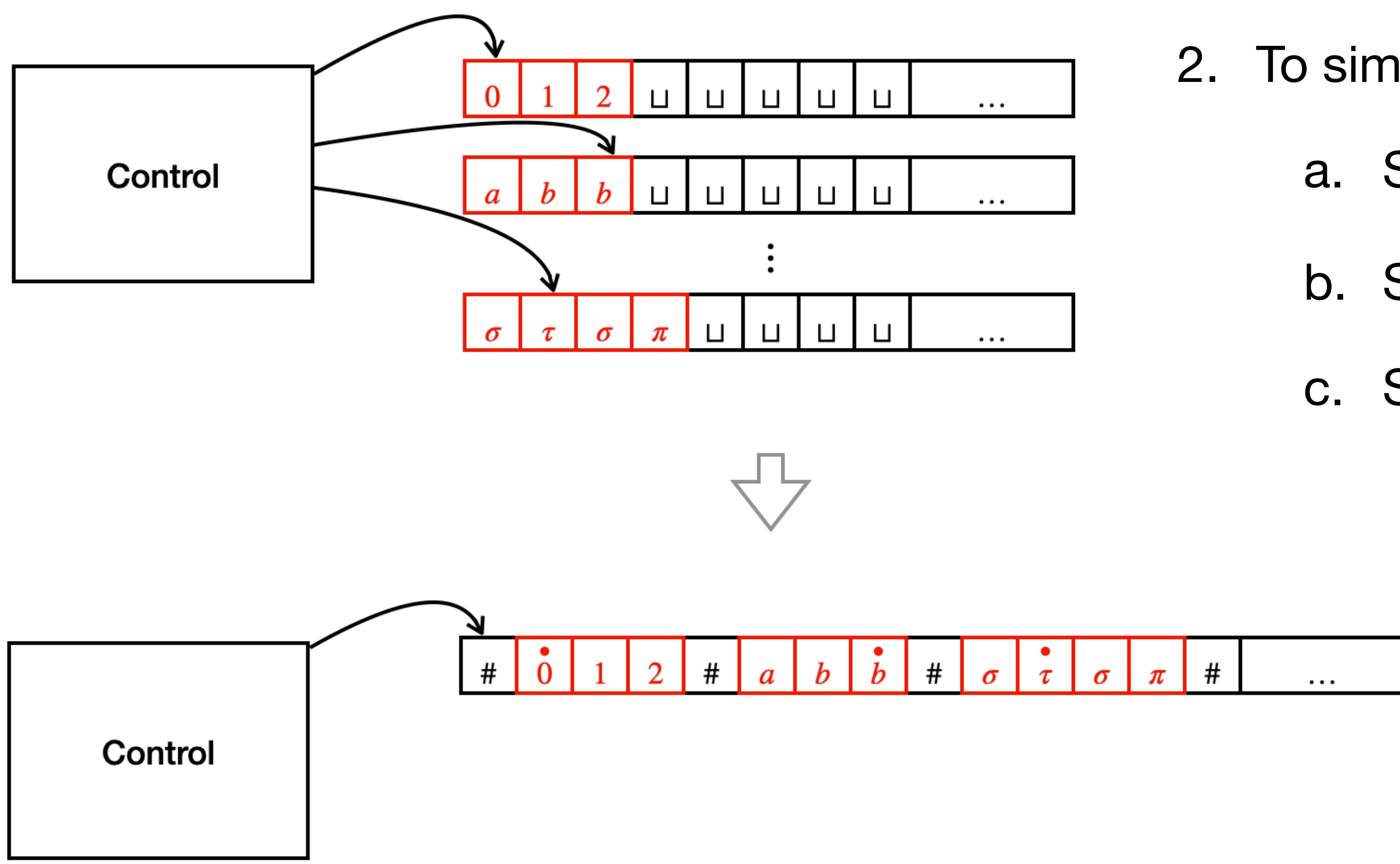
# Description of $M'$




$M'$ = "On input $w = w_1 w_2 \ldots w_n$:

1. Initiate tape: $\# \dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$
2. To simulate a step in $M$:
   a. Scan entire tape to find dotted symbols.
   b. Scan again to update according to $M$.

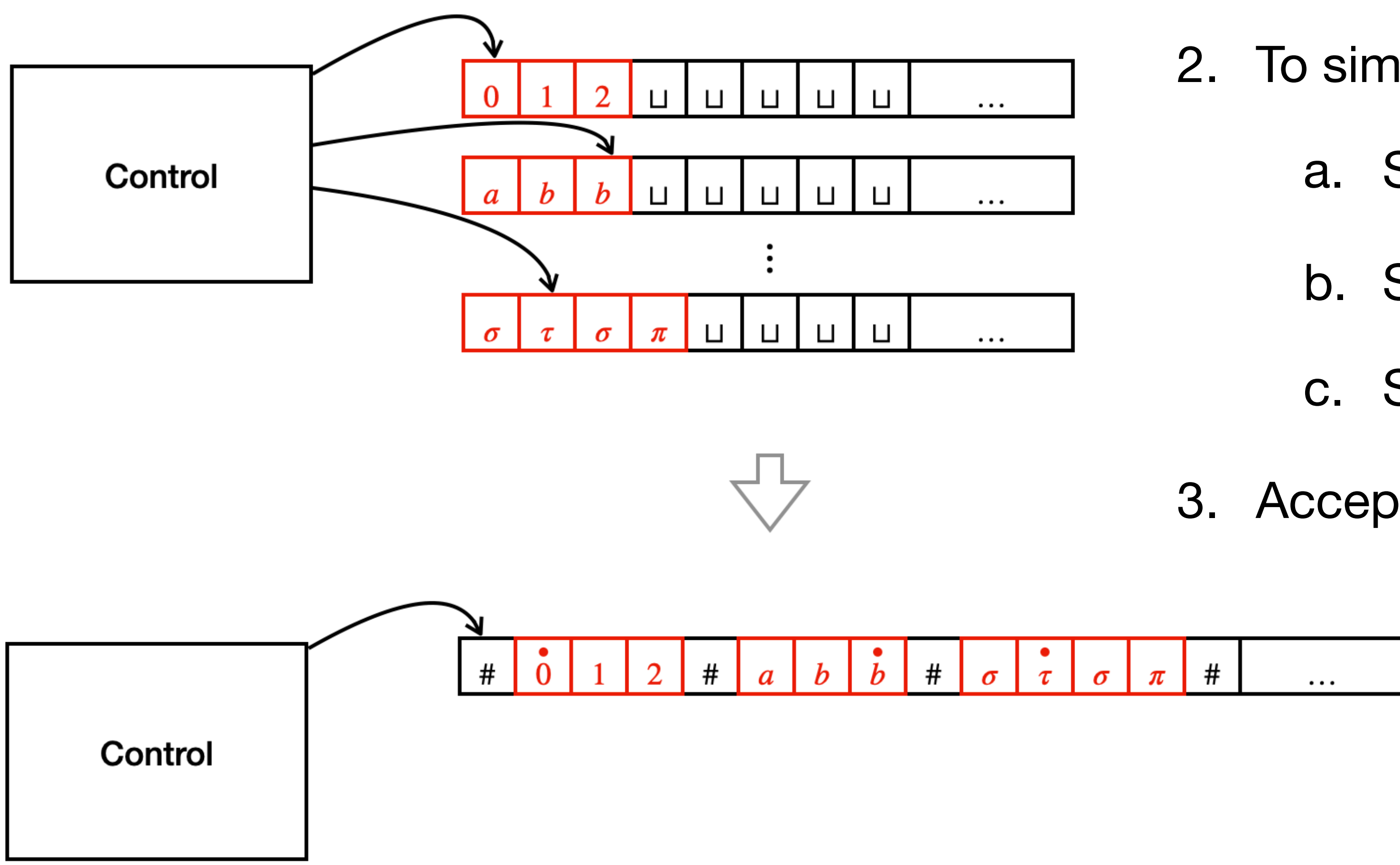
# Description of $M'$




$M' =$ "On input $w = w_1 w_2 \ldots w_n$:

1. Initiate tape: $\#\dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$
2. To simulate a step in $M$:
   a. Scan entire tape to find dotted symbols.
   b. Scan again to update according to $M$.
   c. Shift to add room as needed.

# Description of $M'$



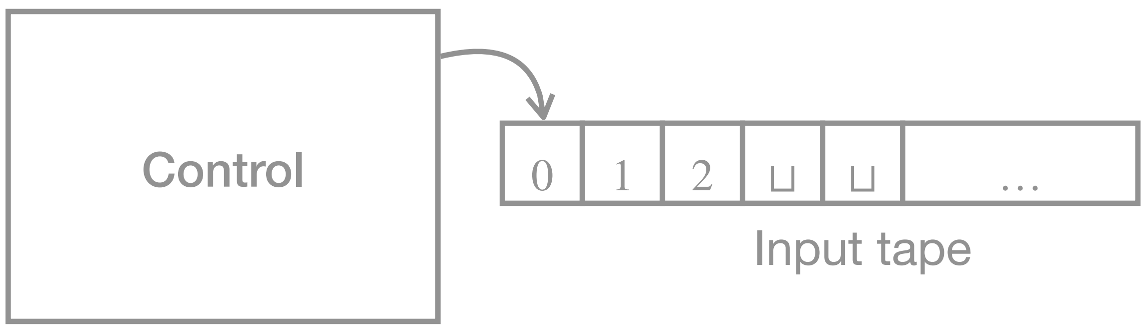

$M'$ = "On input $w = w_1 w_2 \ldots w_n$:

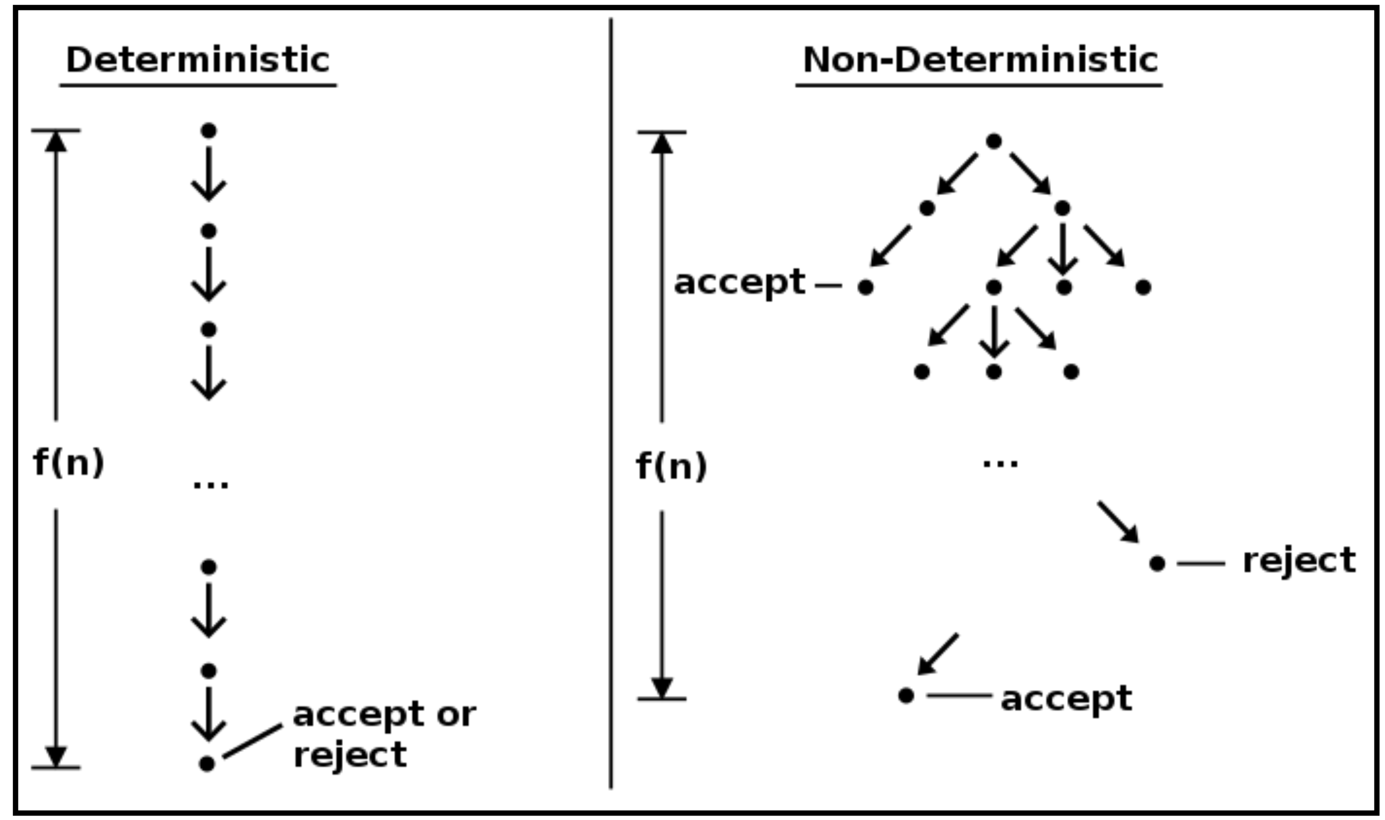
1. Initiate tape: $\#\dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$
2. To simulate a step in $M$:
   a. Scan entire tape to find dotted symbols.
   b. Scan again to update according to $M$.
   c. Shift to add room as needed.
3. Accept if $M$ does."



**Q.E.D.**

# Nondeterministic TMs




**Definition (p. 178):** A <u>Nondeterministic Turing Machine</u> is like a Deterministic TM, except for the transition function $\delta : Q \times \Gamma \mapsto \mathscr{P}(Q \times \Gamma \times \{\mathsf{L}, \mathsf{R}\})$.

**Theorem:** The language $A$ is T-recognizable iff some Nondeterministic TM recognizes it.

Figure comparing of deterministic and nondeterministic computation taken from https://en.wikipedia.org/wiki/Nondeterministic_Turing_machine

# Proof of Equivalence of Nondeterministic TMs and TMs

We must prove both directions:

- $A$ is T-recognizable $\Longrightarrow$ Nondeterministic TM recognizes $A$.
- Nondeterministic TM recognizes $A \Longrightarrow A$ is T-recognizable.

# Proof of Equivalence of Nondeterministic TMs and TMs

We must prove both directions:

- $A$ is T-recognizable $\implies$ Nondeterministic TM recognizes $A$.

  Vacuously true.

- Nondeterministic TM recognizes $A \implies A$ is T-recognizable.

# Proof of Equivalence of Nondeterministic TMs and TMs

We must prove both directions:

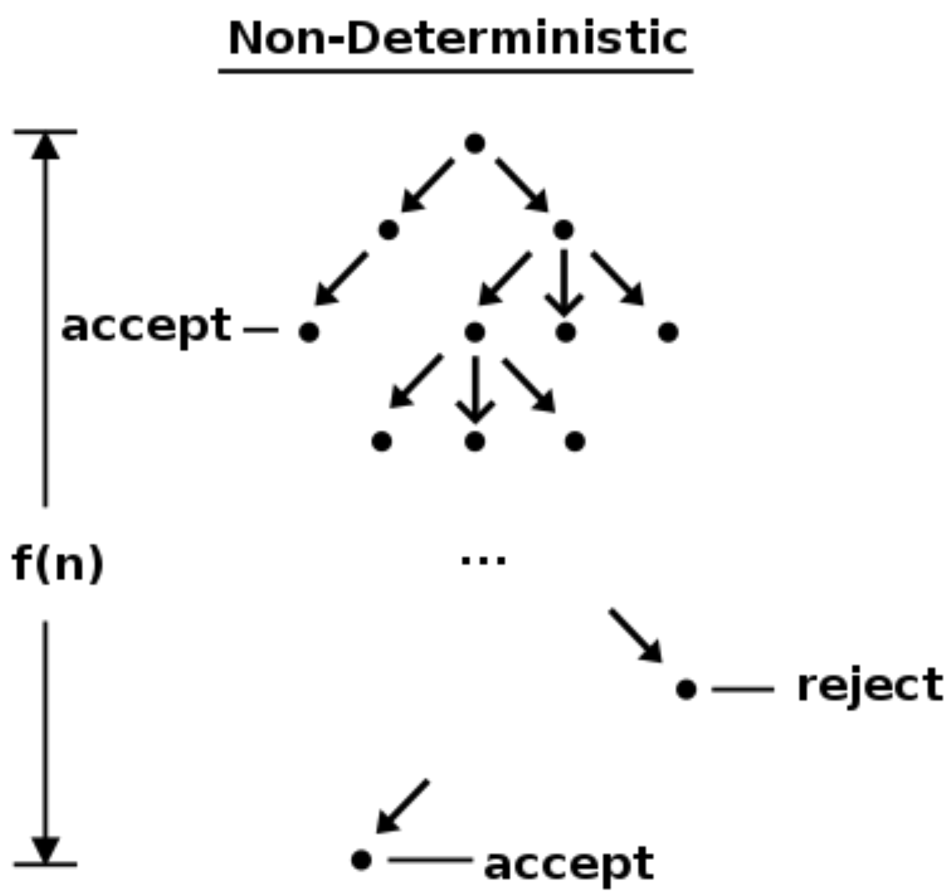
- $A$ is T-recognizable $\implies$ Nondeterministic TM recognizes $A$.

  Vacuously true.

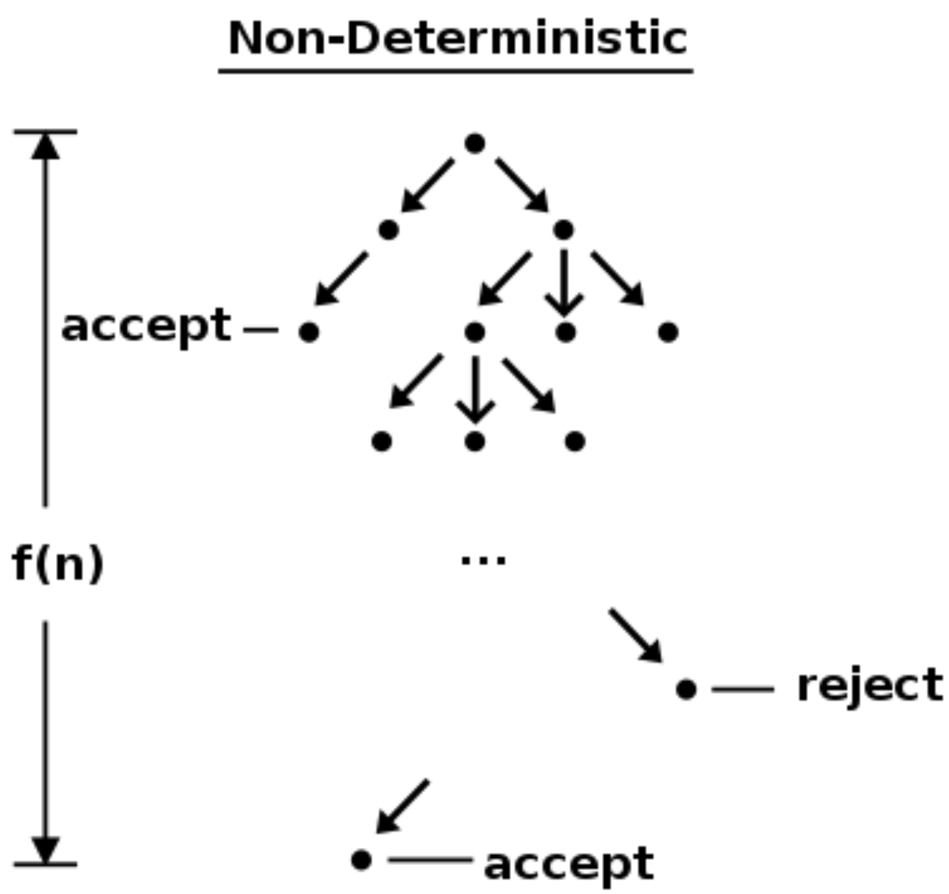
- Nondeterministic TM recognizes $A \implies A$ is T-recognizable.

  Need to show.

# Proof that Nondeterministic TM $\Longrightarrow$ Deterministic TM

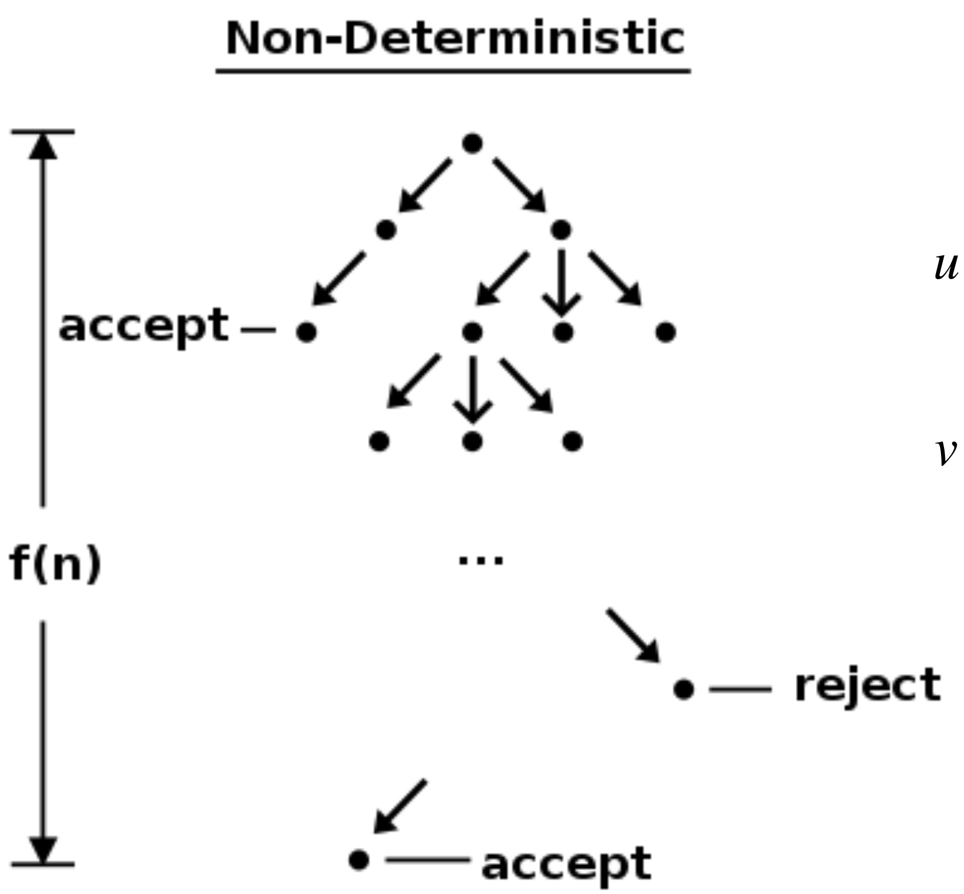


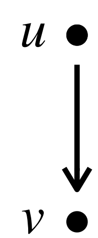
# Proof that Nondeterministic TM $\Longrightarrow$ Deterministic TM




- Each vertex represents a configuration

# Proof that Nondeterministic TM $\Longrightarrow$ Deterministic TM

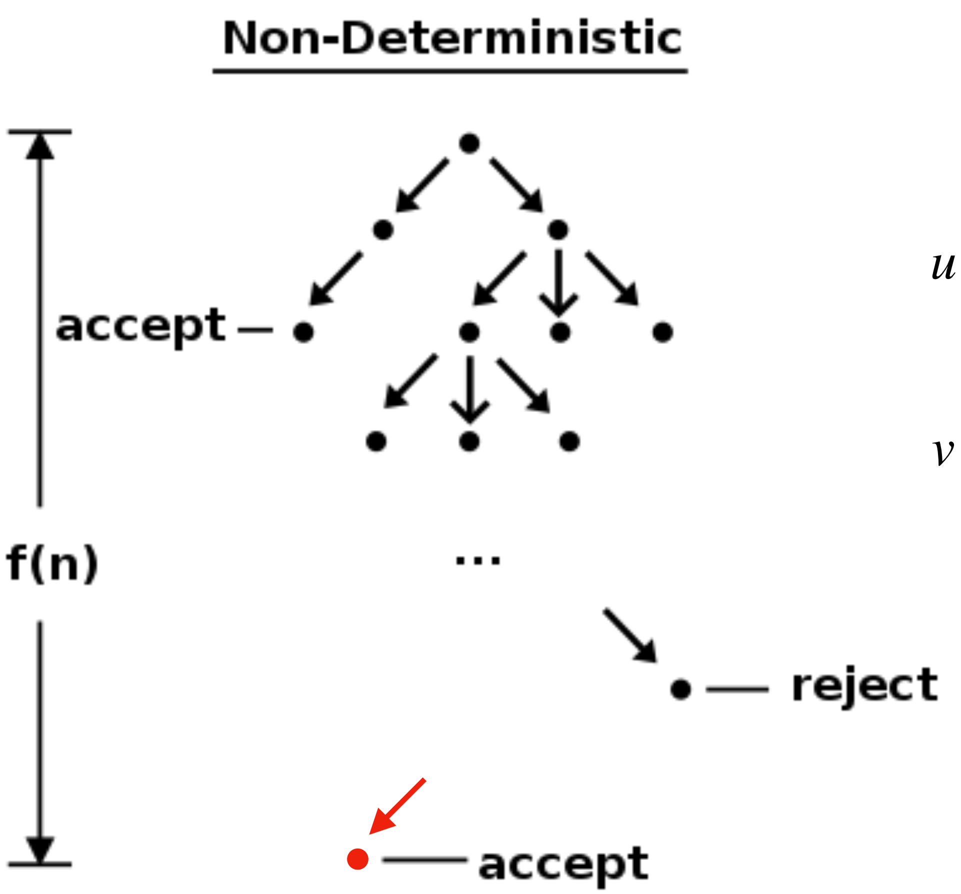


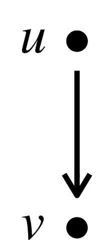




Each vertex represents a configuration

An edge between $u$ and $v$ represents “$u$ yields $v$”

# Proof that Nondeterministic TM $\Longrightarrow$ Deterministic TM



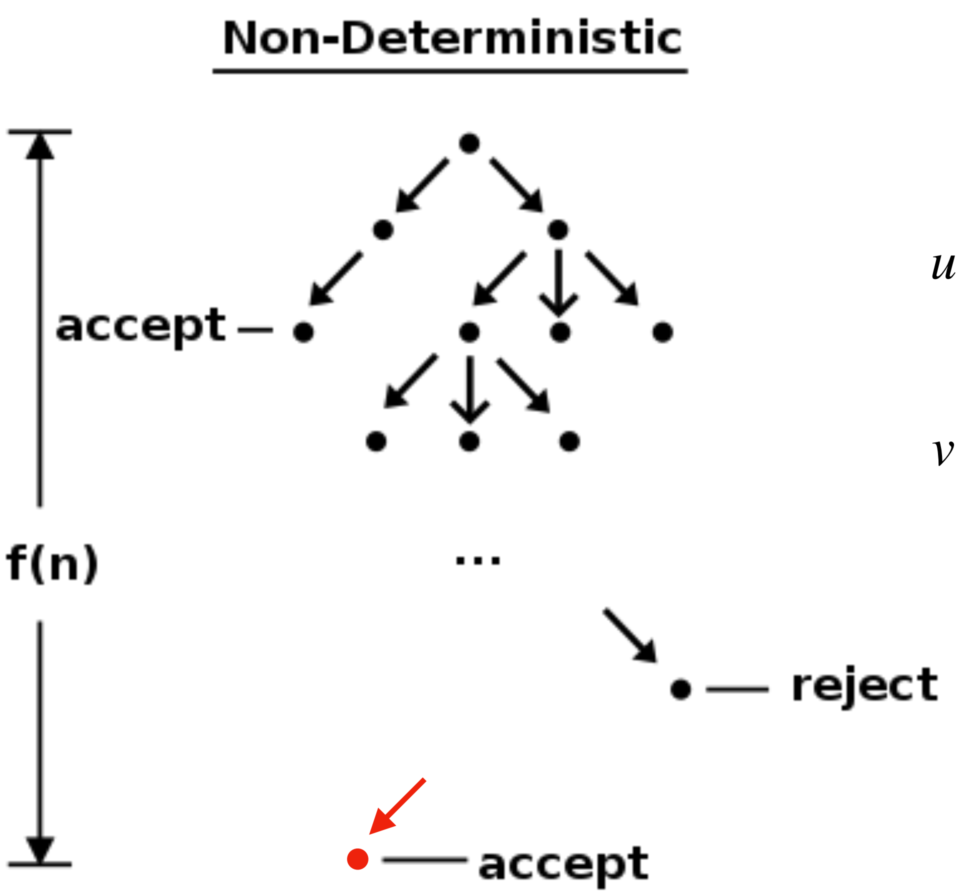



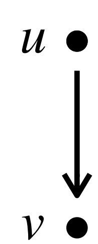

Each vertex represents a configuration

An edge between $u$ and $v$ represents "$u$ yields $v$"

NTM accepts input if there exists "accepting branch"

# Proof that Nondeterministic TM $\Longrightarrow$ Deterministic TM




Each vertex represents a configuration

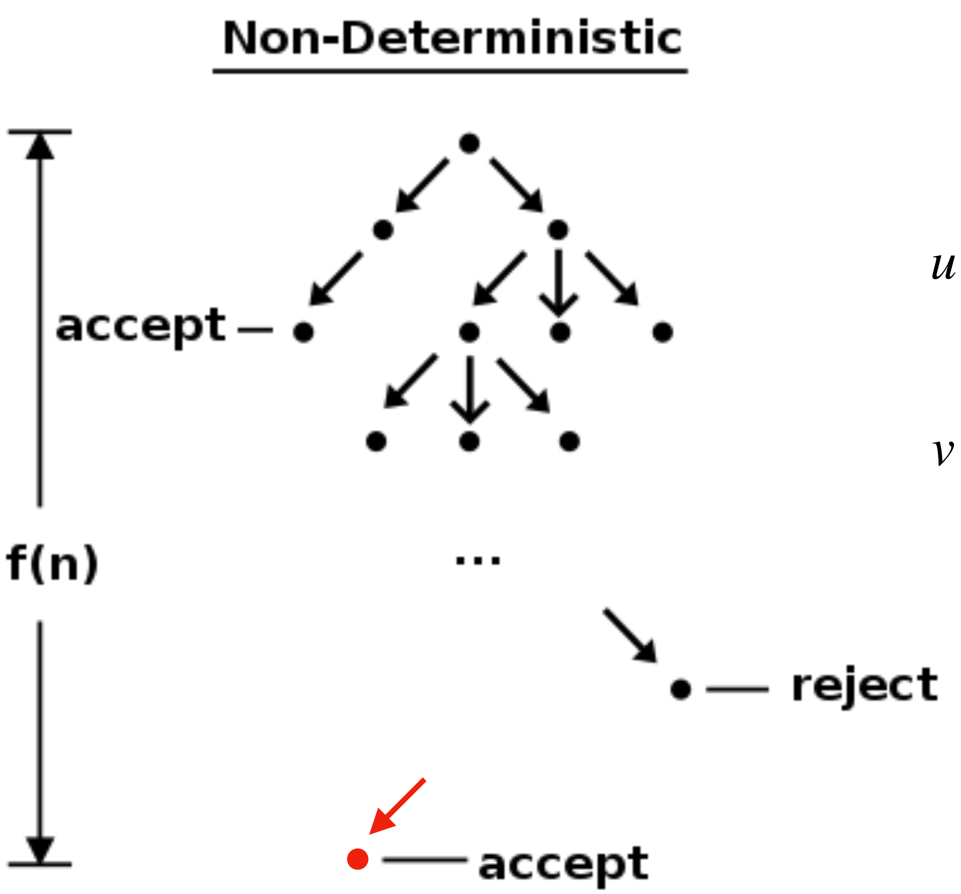
An edge between $u$ and $v$ represents "$u$ yields $v$"

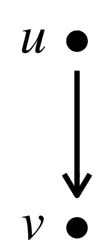
NTM accepts input if there exists "accepting branch"

**Attempt #1** at simulating NTM $N$.

Look for an accepting configuration via a depth-first search on the computation tree. (Doesn't work. Why?)

# Proof that Nondeterministic TM $\Longrightarrow$ Deterministic TM




Each vertex represents a configuration

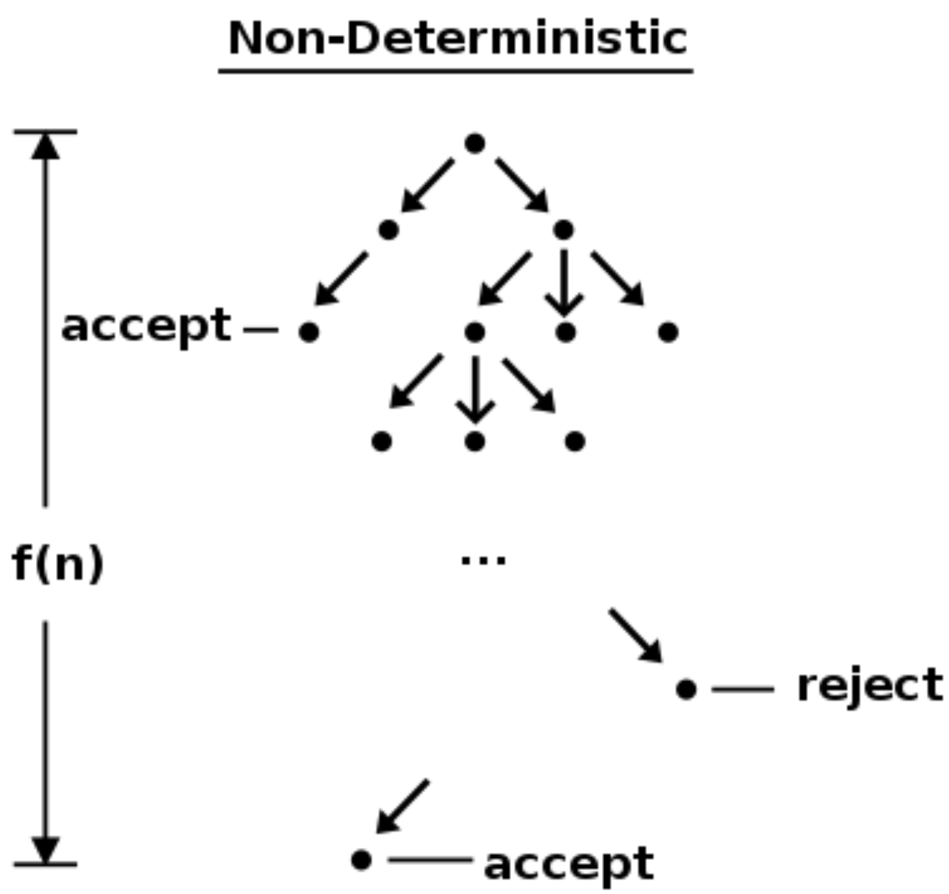
An edge between $u$ and $v$ represents "$u$ yields $v$"

NTM accepts input if there exists "accepting branch"

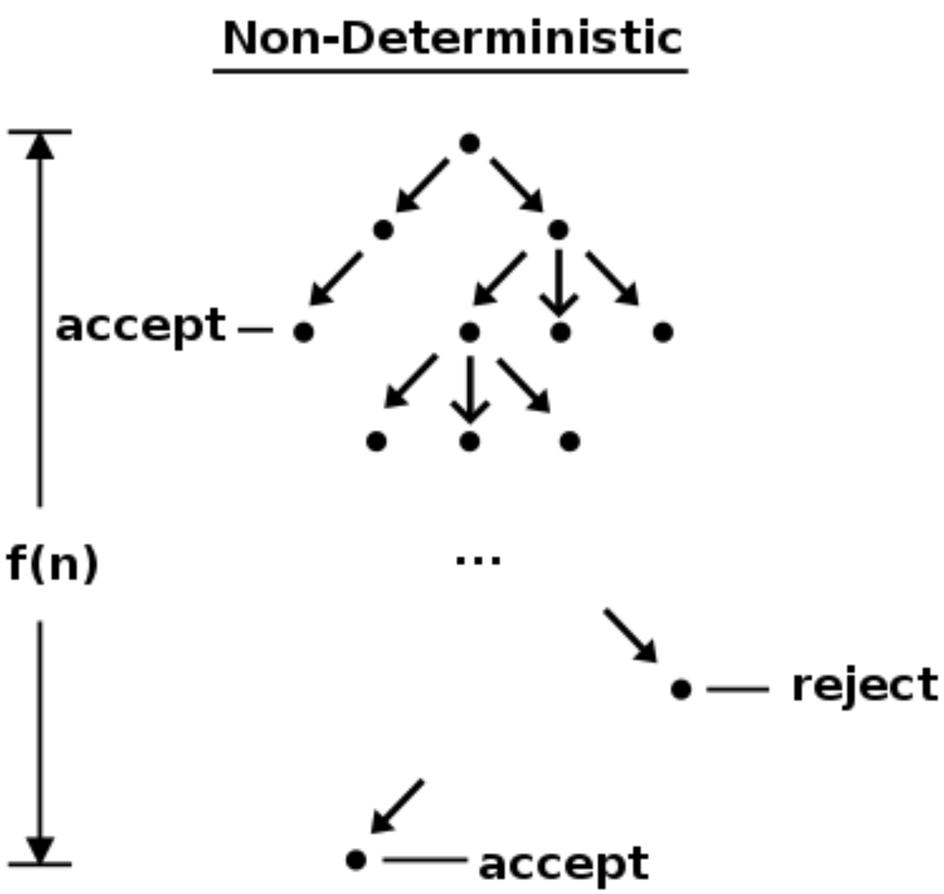
**Proof Idea:**

Look for an accepting configuration via a breadth-first search on the computation tree.

# Proof that Nondeterministic TM $\Longrightarrow$ Deterministic TM

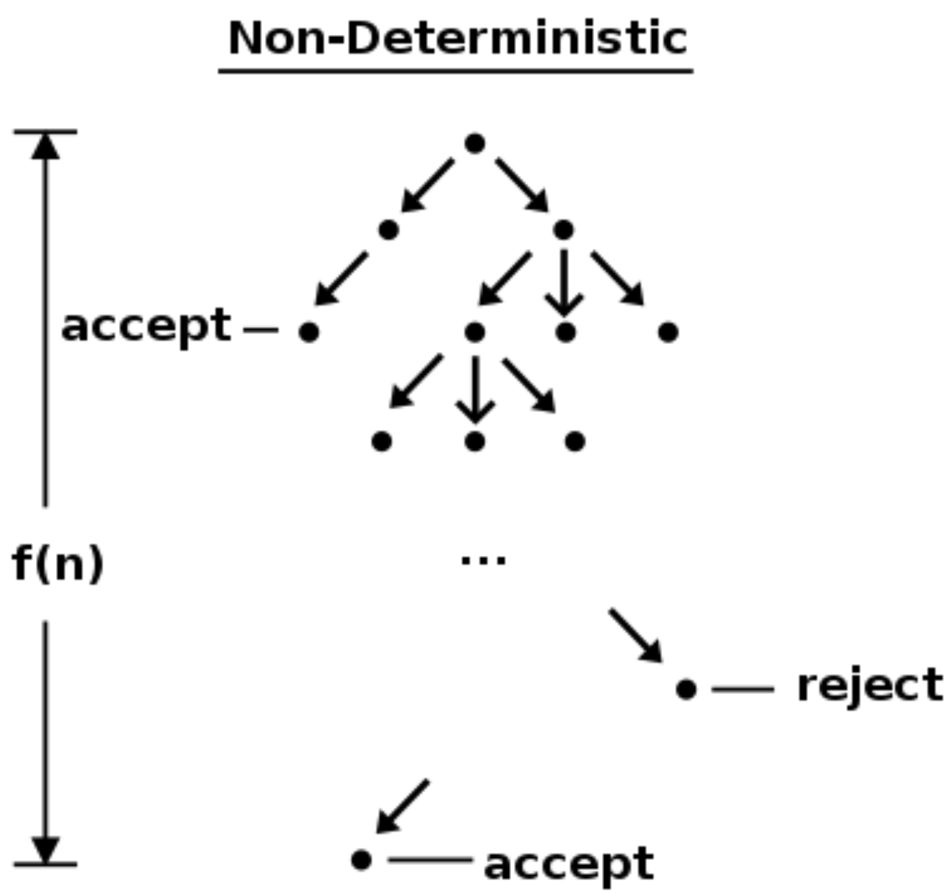


# Proof that Nondeterministic TM $\Longrightarrow$ Deterministic TM




Let $N$ be any NTM.

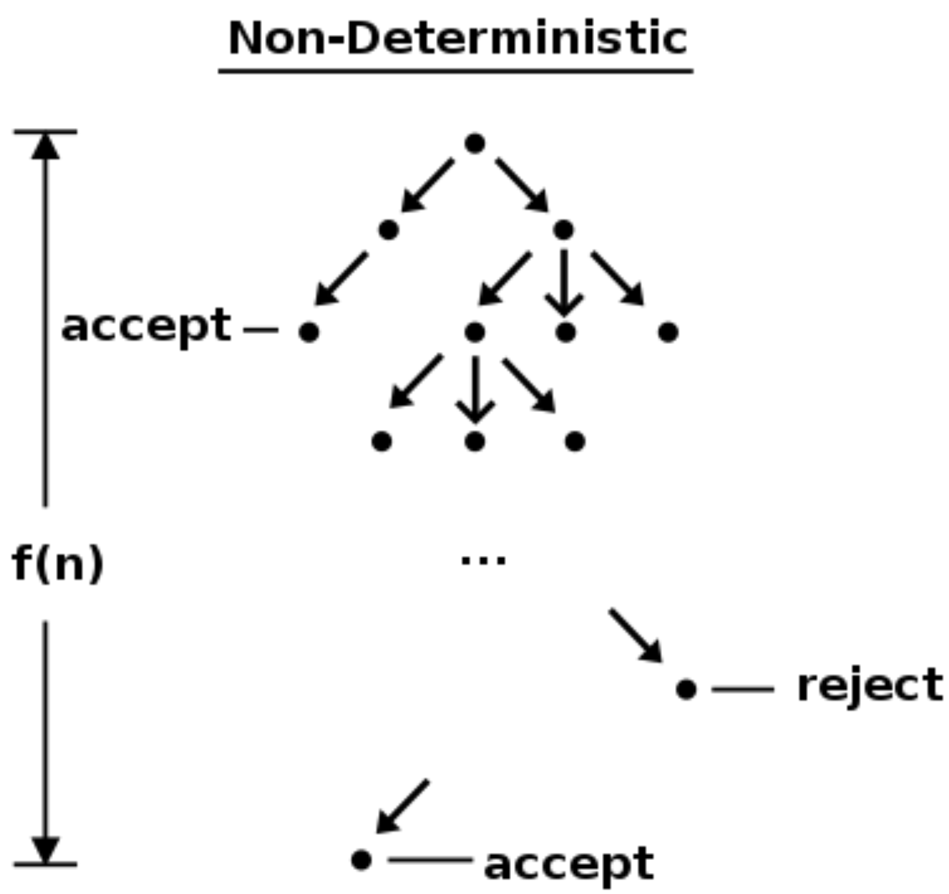
# Proof that Nondeterministic TM $\Longrightarrow$ Deterministic TM




Let $N$ be any NTM.

Define DTM $N' =$ "On input $w = w_1 w_2 \ldots w_n$:

# Proof that Nondeterministic TM $\Longrightarrow$ Deterministic TM

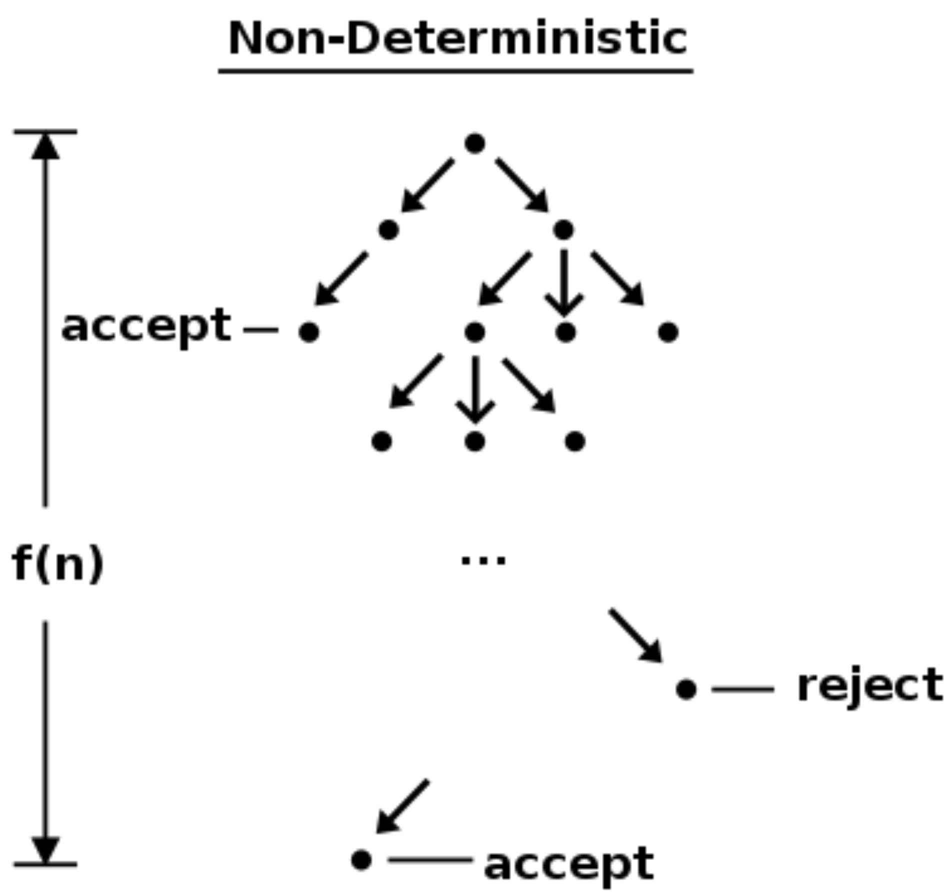



Let $N$ be any NTM.

Define DTM $N' =$ "On input $w = w_1 w_2 \ldots w_n$:

1. Initiate tape: $\# q_{\text{start}} w_1 w_2 \ldots w_n \# \sqcup \sqcup \ldots$

# Proof that Nondeterministic TM $\Longrightarrow$ Deterministic TM

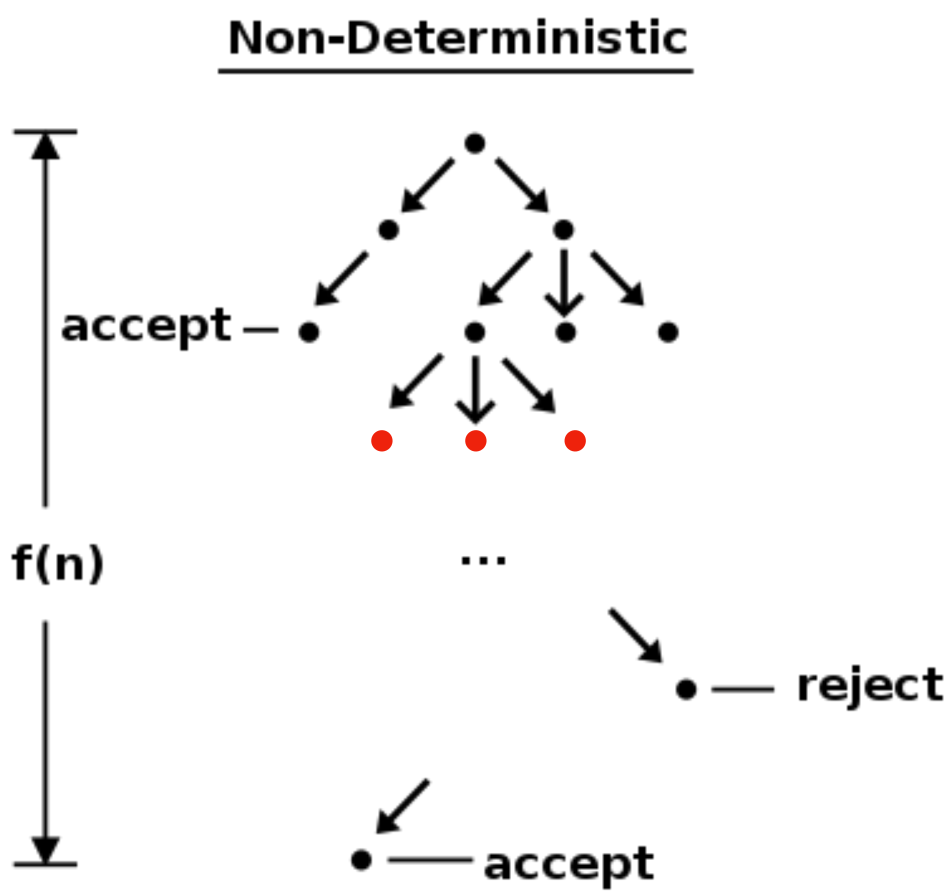



Let $N$ be any NTM.

Define DTM $N' =$ "On input $w = w_1 w_2 \ldots w_n$:

1. Initiate tape: $\# q_{\text{start}} w_1 w_2 \ldots w_n \# \sqcup \sqcup \ldots$
2. To simulate next depth in $N$'s computation tree:

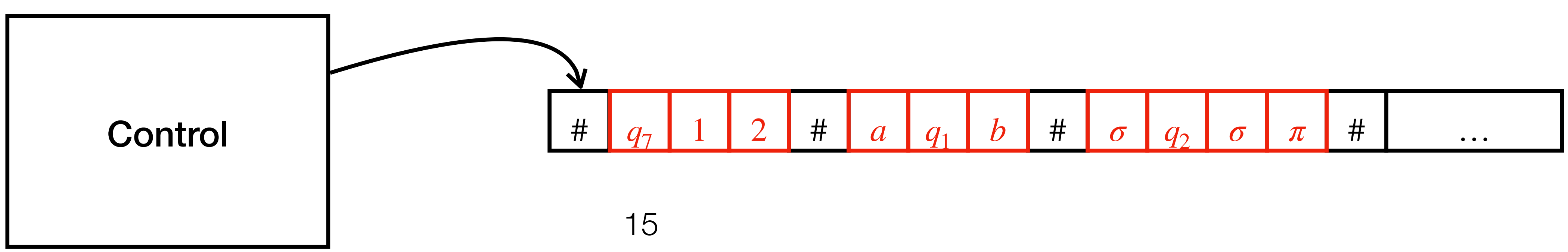
# Proof that Nondeterministic TM $\implies$ Deterministic TM



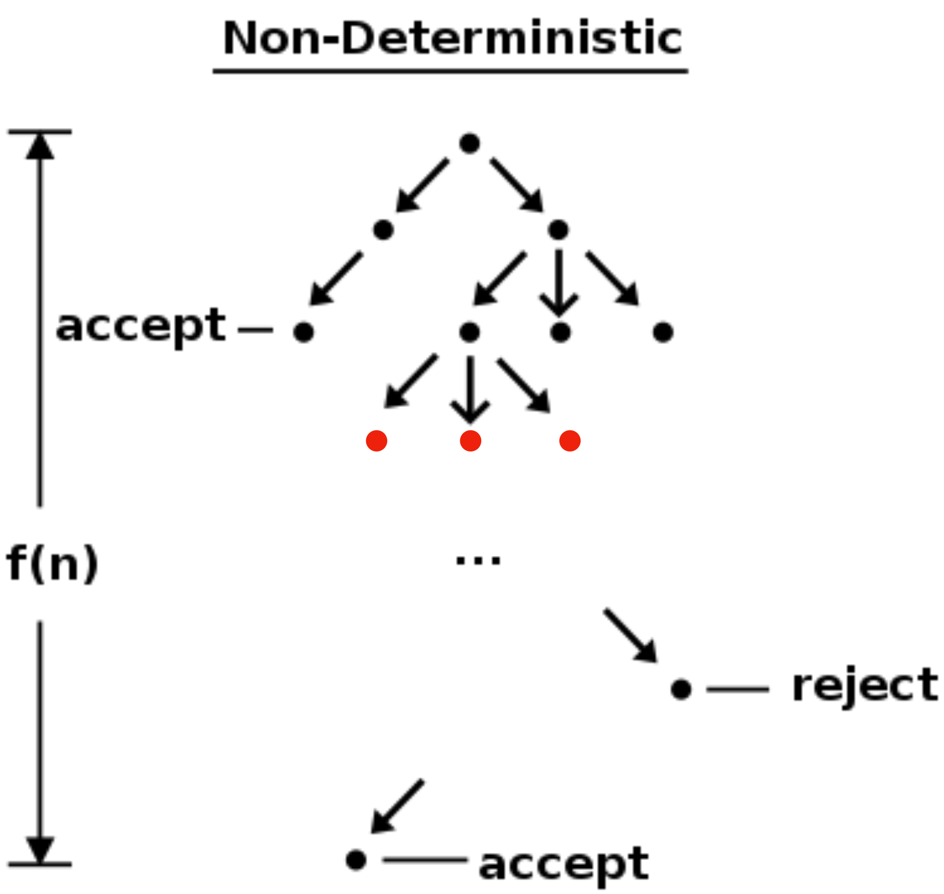

Let $N$ be any NTM.

Define DTM $N' =$ "On input $w = w_1 w_2 \ldots w_n$:

1. Initiate tape: $\# q_{\text{start}} w_1 w_2 \ldots w_n \# \sqcup \sqcup \ldots$
2. To simulate next depth in $N$'s computation tree:
   a. Scan entire tape, updating each "thread" (delimited by #'s) according to $N$.



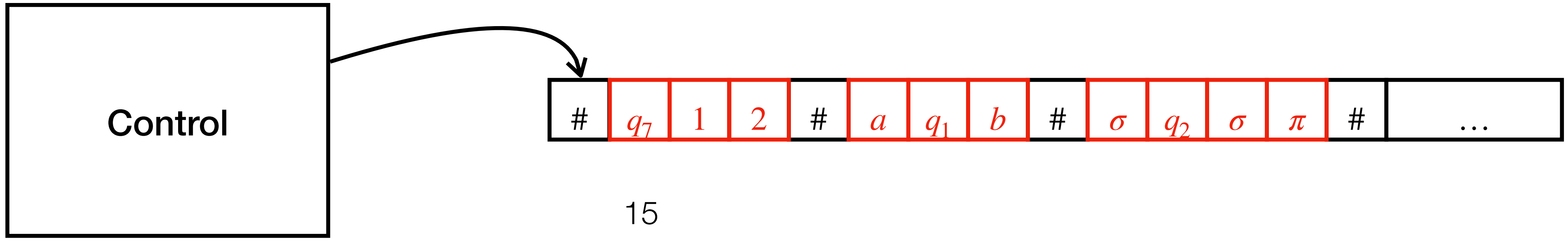
# Proof that Nondeterministic TM $\implies$ Deterministic TM



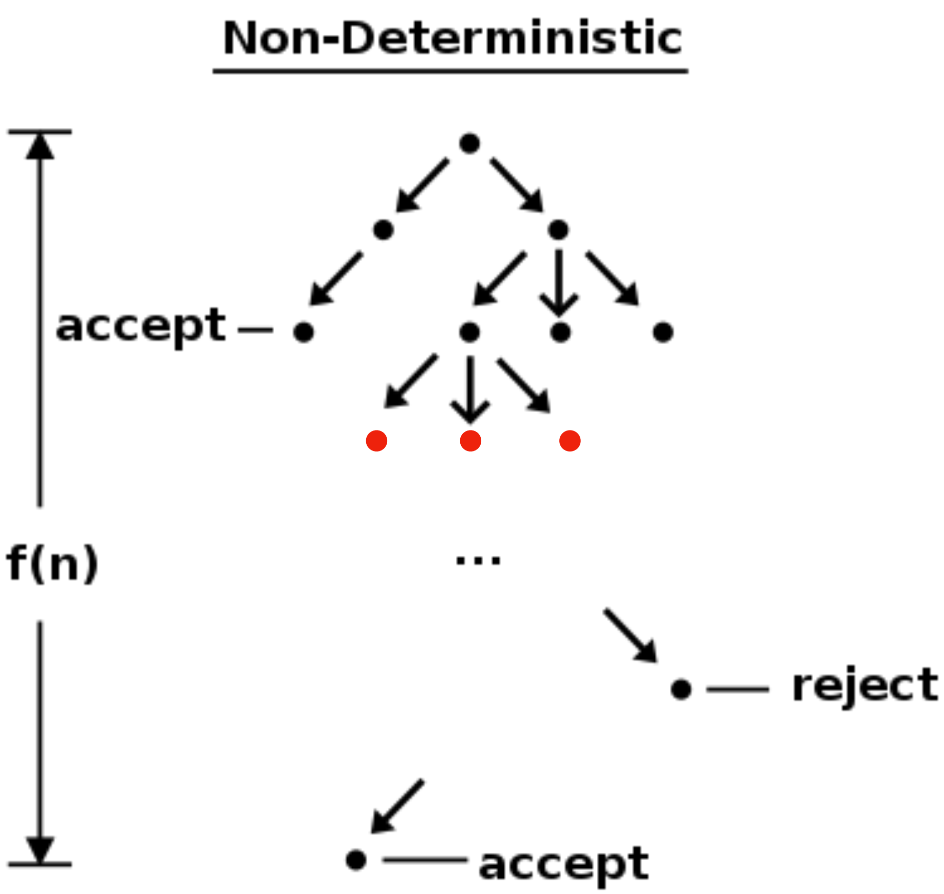

Let $N$ be any NTM.

Define DTM $N' =$ "On input $w = w_1 w_2 \ldots w_n$:

1. Initiate tape: $\# q_{\text{start}} w_1 w_2 \ldots w_n \# \sqcup \sqcup \ldots$
2. To simulate next depth in $N$'s computation tree:
   a. Scan entire tape, updating each "thread" (delimited by #'s) according to $N$.
   b. If a thread forks, copy thread.



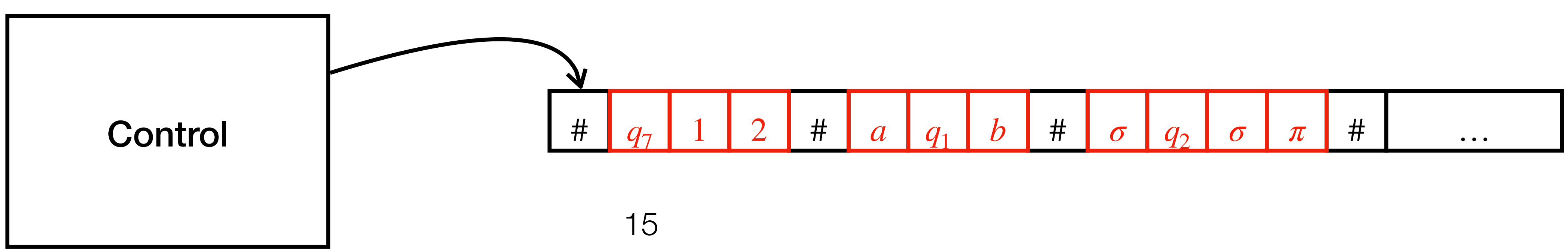
# Proof that Nondeterministic TM $\Longrightarrow$ Deterministic TM




Let $N$ be any NTM.

Define DTM $N' =$ "On input $w = w_1 w_2 \ldots w_n$:

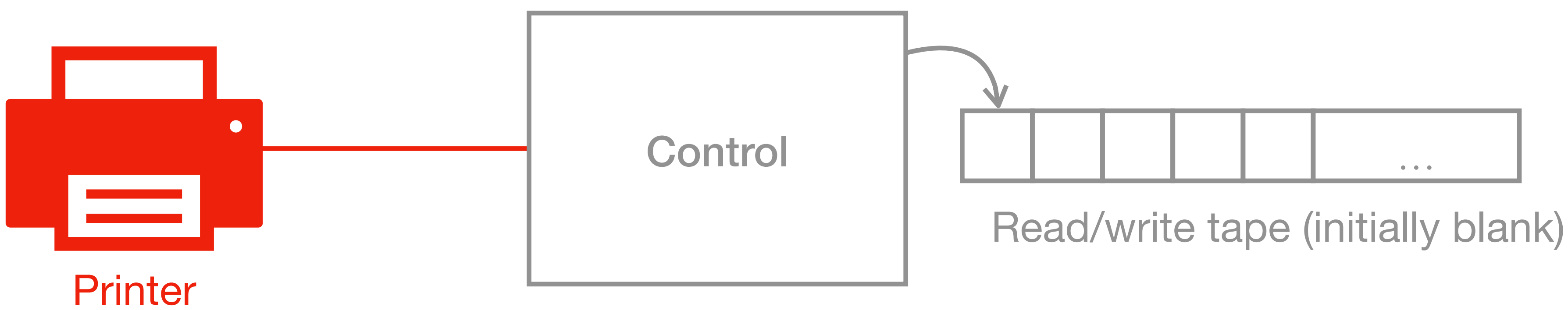
1. Initiate tape: $\# q_{\text{start}} w_1 w_2 \ldots w_n \# \sqcup \sqcup \ldots$
2. To simulate next depth in $N$'s computation tree:
   a. Scan entire tape, updating each "thread" (delimited by #'s) according to $N$.
   b. If a thread forks, copy thread.
   c. **If a thread accepts, accept**." **Q.E.D.**



# Check In (Break)

**True or false:** A language is decidable iff some nondeterministic TM decides it.

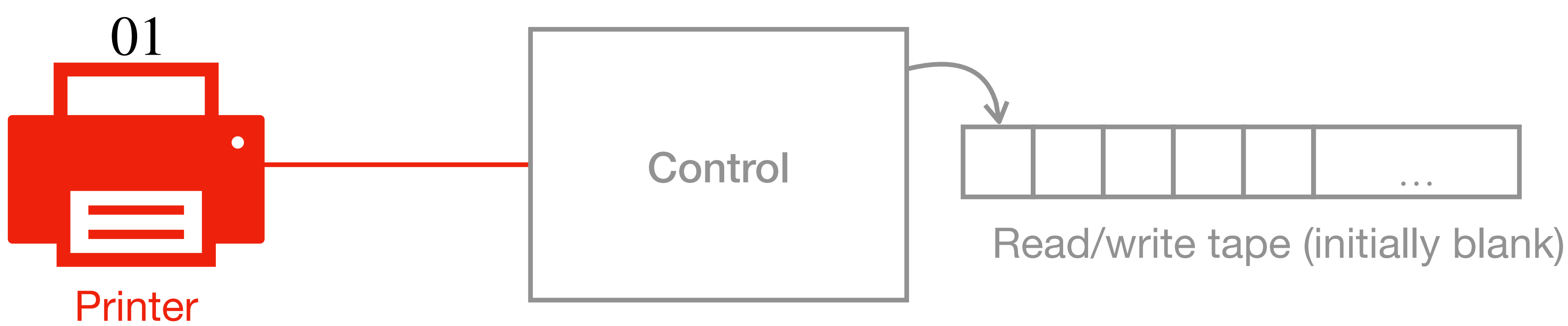
# Enumerators




**Definition (p. 180):** An enumerator $E$ is like a Deterministic TM, except it has a "printer." It starts with a blank tape and eventually prints out all the strings $w_1, w_2, \ldots$ in the language $L(E)$ it generates.

**Theorem:** The language $A$ is T-recognizable iff $A = L(E)$ for some enumerator $E$.

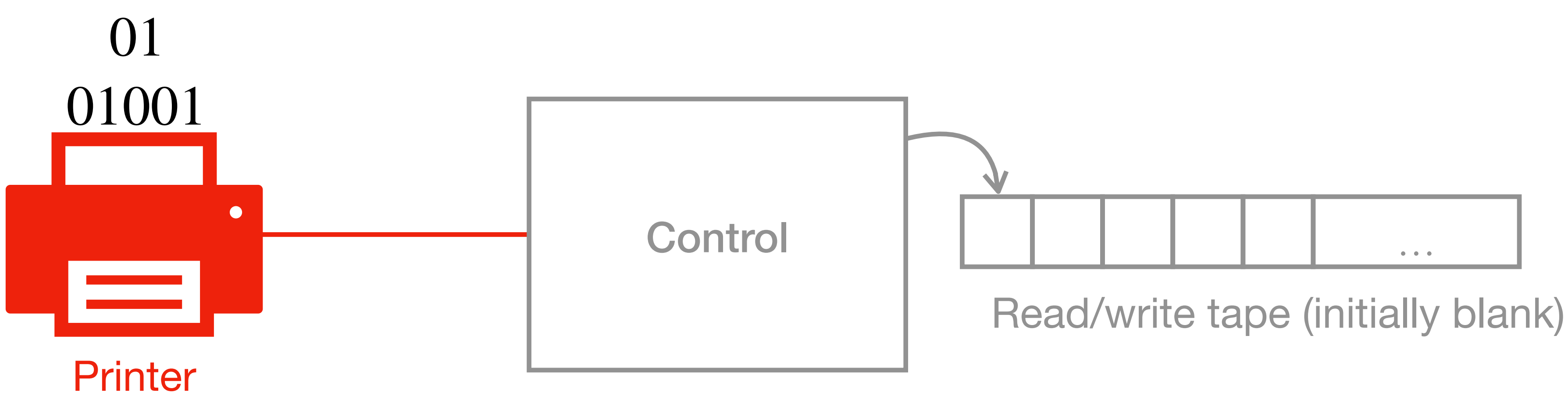
# Enumerators




**Definition (p. 180):** An <u>enumerator</u> $E$ is like a Deterministic TM, except it has a "printer." It starts with a blank tape and eventually prints out all the strings $w_1, w_2, \ldots$ in the language $L(E)$ it generates.

**Theorem:** The language $A$ is T-recognizable iff $A = L(E)$ for some enumerator $E$.

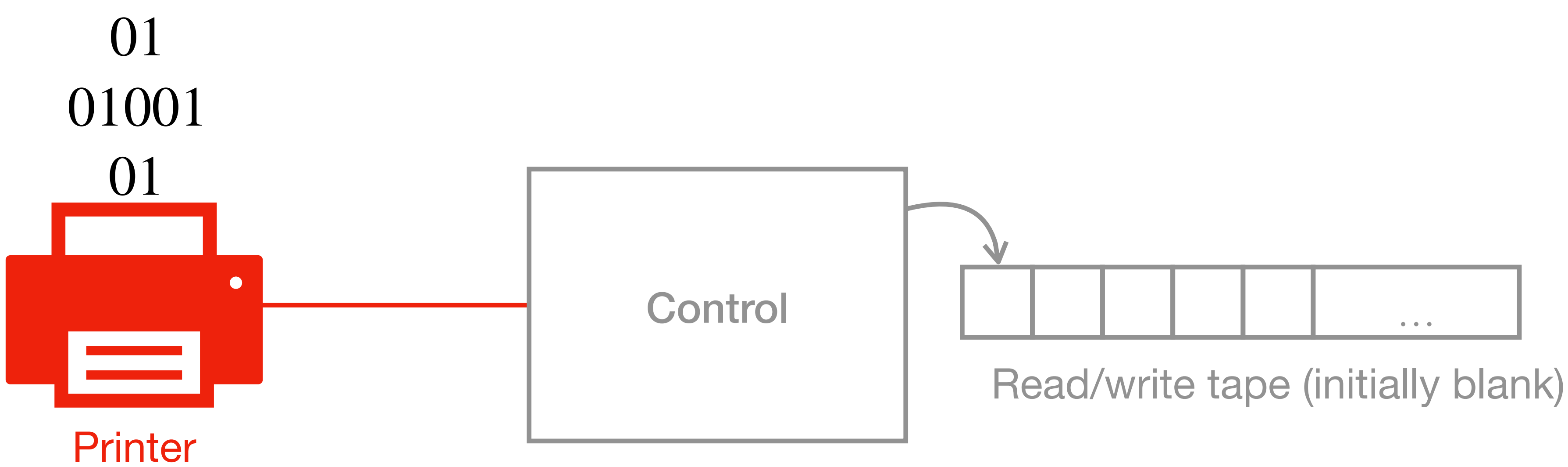
# Enumerators




**Definition (p. 180):** An <u>enumerator</u> $E$ is like a Deterministic TM, except it has a "printer." It starts with a blank tape and eventually prints out all the strings $w_1, w_2, \ldots$ in the language $L(E)$ it generates.

**Theorem:** The language $A$ is T-recognizable iff $A = L(E)$ for some enumerator $E$.

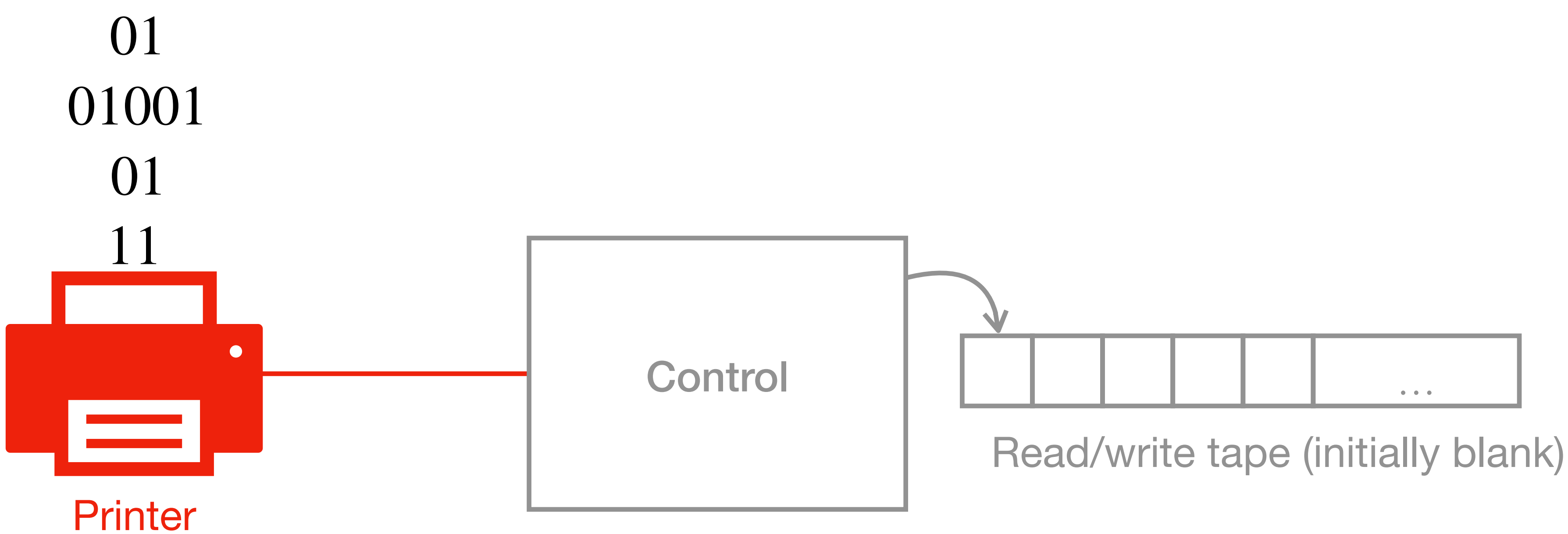
# Enumerators




**Definition (p. 180):** An enumerator $E$ is like a Deterministic TM, except it has a "printer." It starts with a blank tape and eventually prints out all the strings $w_1, w_2, \ldots$ in the language $L(E)$ it generates.

**Theorem:** The language $A$ is T-recognizable iff $A = L(E)$ for some enumerator $E$.

# Enumerators




**Definition (p. 180):** An <u>enumerator</u> $E$ is like a Deterministic TM, except it has a "printer." It starts with a blank tape and eventually prints out all the strings $w_1, w_2, \ldots$ in the language $L(E)$ it generates.

**Theorem:** The language $A$ is T-recognizable iff $A = L(E)$ for some enumerator $E$.

# Proof of Equivalence of Enumerators and TMs

# Proof of Equivalence of Enumerators and TMs

Enumerator generates $A \implies A$ is T-recognizable.

# Proof of Equivalence of Enumerators and TMs

Enumerator generates $A \implies A$ is T-recognizable.

$A$ is T-recognizable $\implies$ Enumerator generates $A$.

# Proof of Equivalence of Enumerators and TMs

Enumerator generates $A \Longrightarrow A$ is T-recognizable.

**Proof.** Let $E$ be enumerator that generates $A$. Construct TM as follows:

$A$ is T-recognizable $\Longrightarrow$ Enumerator generates $A$.

# Proof of Equivalence of Enumerators and TMs

Enumerator generates $A \Longrightarrow A$ is T-recognizable.

$A$ is T-recognizable $\Longrightarrow$ Enumerator generates $A$.

**Proof.** Let $E$ be enumerator that generates $A$. Construct TM as follows:

$M =$ “On input $w$,

# Proof of Equivalence of Enumerators and TMs

Enumerator generates $A \Longrightarrow A$ is T-recognizable.

$A$ is T-recognizable $\Longrightarrow$ Enumerator generates $A$.

**Proof.** Let $E$ be enumerator that generates $A$. Construct TM as follows:

$M =$ "On input $w$,

1. Run $E$.

# Proof of Equivalence of Enumerators and TMs

Enumerator generates $A \implies A$ is T-recognizable.

$A$ is T-recognizable $\implies$ Enumerator generates $A$.

**Proof.** Let $E$ be enumerator that generates $A$. Construct TM as follows:

$M =$ "On input $w$,

1. Run $E$.
2. Whenever $E$ prints out a string $w'$, check whether $w' = w$. If it is, accept."

# Proof of Equivalence of Enumerators and TMs

Enumerator generates $A \implies A$ is T-recognizable.

**Proof.** Let $E$ be enumerator that generates $A$. Construct TM as follows:

$M =$ "On input $w$,

1. Run $E$.
2. Whenever $E$ prints out a string $w'$, check whether $w' = w$. If it is, accept."

$A$ is T-recognizable $\implies$ Enumerator generates $A$.

**Proof.** Let $M$ be TM that recognizes $A$. Construct enumerator as follows:

# Proof of Equivalence of Enumerators and TMs

Enumerator generates $A \Longrightarrow A$ is T-recognizable.

**Proof.** Let $E$ be enumerator that generates $A$. Construct TM as follows:

$M =$ "On input $w$,

1. Run $E$.
2. Whenever $E$ prints out a string $w'$, check whether $w' = w$. If it is, accept."

$A$ is T-recognizable $\Longrightarrow$ Enumerator generates $A$.

**Proof.** Let $M$ be TM that recognizes $A$. Construct enumerator as follows:

$E$ = "Repeat for $i = 1,2,\ldots$

# Proof of Equivalence of Enumerators and TMs

Enumerator generates $A \implies A$ is T-recognizable.

**Proof.** Let $E$ be enumerator that generates $A$. Construct TM as follows:

$M =$ "On input $w$,

1. Run $E$.
2. Whenever $E$ prints out a string $w'$, check whether $w' = w$. If it is, accept."

$A$ is T-recognizable $\implies$ Enumerator generates $A$.

**Proof.** Let $M$ be TM that recognizes $A$. Construct enumerator as follows:

$E$ = "Repeat for $i = 1,2,\ldots$

1. Run $M$ for $i$ steps on each of the "first" $i$ strings in $\Sigma^*$: $s_1, s_2, \ldots, s_i$.

# Proof of Equivalence of Enumerators and TMs

Enumerator generates $A \implies A$ is T-recognizable.

**Proof.** Let $E$ be enumerator that generates $A$. Construct TM as follows:

$M =$ "On input $w$,

1. Run $E$.
2. Whenever $E$ prints out a string $w'$, check whether $w' = w$. If it is, accept."

$A$ is T-recognizable $\implies$ Enumerator generates $A$.

**Proof.** Let $M$ be TM that recognizes $A$. Construct enumerator as follows:

$E$ = "Repeat for $i = 1,2,\ldots$

1. Run $M$ for $i$ steps on each of the "first" $i$ strings in $\Sigma^*$: $s_1, s_2, \ldots, s_i$.
   a. For $s \in \{s_1, s_2, \ldots, s_i\}$ s.t. $M$ accepts $s$, print $s$."

**Q.E.D.**

# Church-Turing Thesis

- Captures our intuition of computation
- Mathematically precise
- Simple
- Robust: equivalent to other TM variants (e.g., TMs with multiple tapes, Nondeterministic TMs)

# Church-Turing Thesis

- Captures our intuition of computation
- Mathematically precise
- Simple
- Robust: equivalent to other TM variants (e.g., TMs with multiple tapes, Nondeterministic TMs)
- Church-Turing Thesis: any (classical) real-world computation can be modeled as a TM

# Summary of Today's Lecture

- Robustness of TMs
- Church-Turing Thesis

# Acknowledgements


- These slides are based on lecture notes on Theory of Computation from other universities, namely Michael Sipser (MIT), Lorenzo De Stefani (Brown).

- **Errata:** If you let us know of any errors in the slides, we'll fix them and acknowledge you here!