Lecture 19: Mapping Reducibility and Time Complexity

CSC 320: Foundations of Computer Science

Quinton Yong

quintonyong@uvic.ca

University of Victoria

Computable Functions

Definition: A function $\boldsymbol{f}: \boldsymbol{\Sigma}^* \to \boldsymbol{\Sigma}^*$ is a **computable function** if a TM $\boldsymbol{M}$ exists such that on input $\boldsymbol{w}$, $\boldsymbol{M}$ halts with just $\boldsymbol{f(w)}$ on its tape

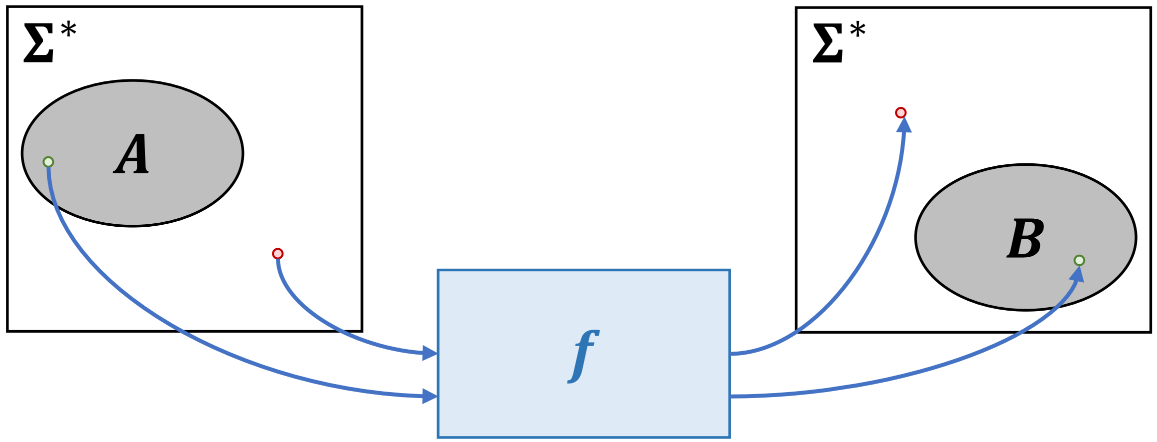
i.e. A **function is computable** if we can use a TM (decider) to compute it

$$\boldsymbol{w} \longrightarrow \boxed{\quad \boldsymbol{f} \quad} \longrightarrow \boldsymbol{f(w)}$$

Mapping Reducibility

Definition: Language $\boldsymbol{A}$ is **mapping reducible** to language $\boldsymbol{B}$, denoted $\boldsymbol{A} \leq_{\boldsymbol{m}} \boldsymbol{B}$, if there is a **computable function** $\boldsymbol{f}: \boldsymbol{\Sigma}^* \rightarrow \boldsymbol{\Sigma}^*$ such that for every $\boldsymbol{w} \in \boldsymbol{\Sigma}^*$,

$$\boldsymbol{w} \in \boldsymbol{A} \text{ if and only if } \boldsymbol{f}(\boldsymbol{w}) \in \boldsymbol{B}$$



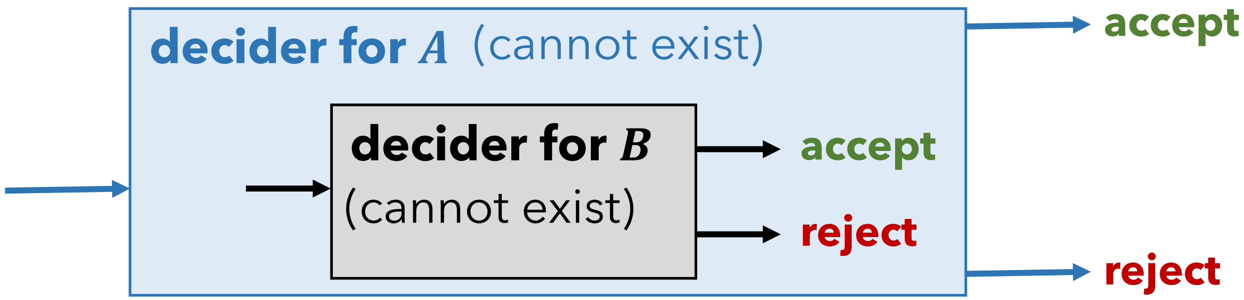
$\boldsymbol{f}$ is called a **mapping reduction** from $\boldsymbol{A}$ to $\boldsymbol{B}$

i.e. There's a way to convert inputs of $\boldsymbol{A}$ to equivalent inputs of $\boldsymbol{B}$
($\boldsymbol{w} \in \boldsymbol{A} \Leftrightarrow \boldsymbol{f}(\boldsymbol{w}) \in \boldsymbol{B}$)

Reductions vs Mapping Reductions

Reductions $A \leq B$ (also known as Turing reductions):

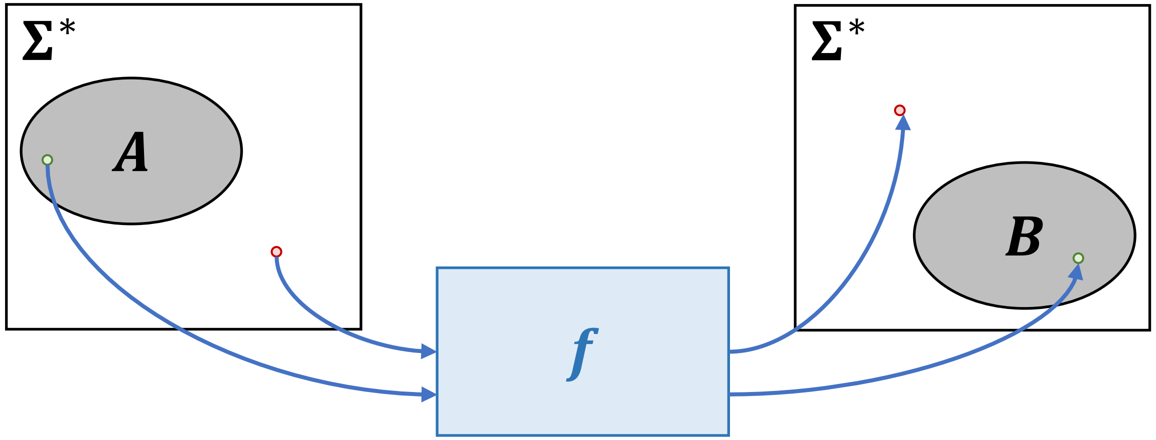
- Show how to create a TM which recognizes (or decides) A if we had a TM which recognizes (or decides) B
- We can show reductions such as $A_{TM} \leq Halt_{TM}$ and $A_{TM} \leq E_{TM}$
- In this course, we will use these reductions to prove that languages are undecidable



Reductions vs Mapping Reductions

Mapping Reductions $A \leq_m B$:

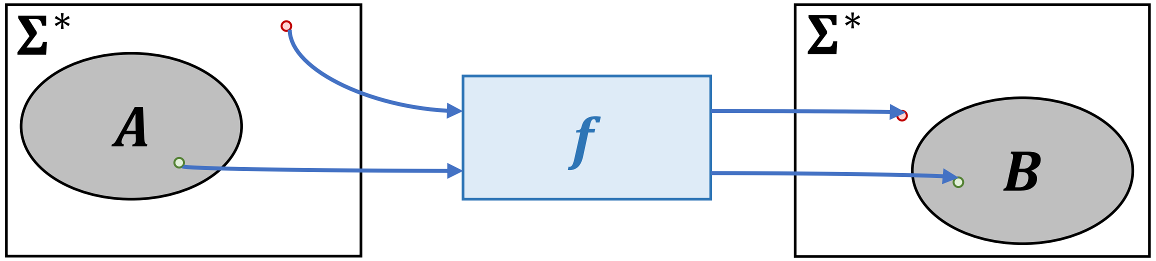
- Show how to convert inputs for $\boldsymbol{A}$ into inputs for $\boldsymbol{B}$ such that for each input $\boldsymbol{w}$, $\boldsymbol{w} \in \boldsymbol{A}$ if and only if $\boldsymbol{f}(\boldsymbol{w}) \in \boldsymbol{B}$
- We can show mapping reductions such as $\boldsymbol{A_{TM} \leq_m Halt_{TM}}$
- There **does not exist** a mapping reduction $\boldsymbol{A_{TM} \leq_m E_{TM}}$
- We can use mapping reductions for decidability, but we will use them primarily for **time complexity** in this course

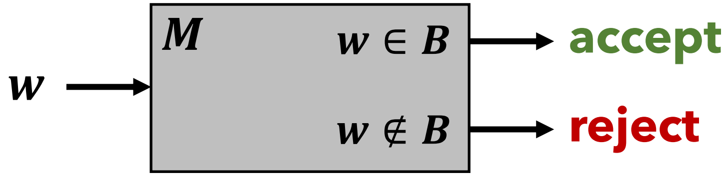


Mapping Reductions for Decidability

Let A and B be languages over Σ.

Theorem: If $A \leq_m B$ and B **is decidable**, then A is decidable

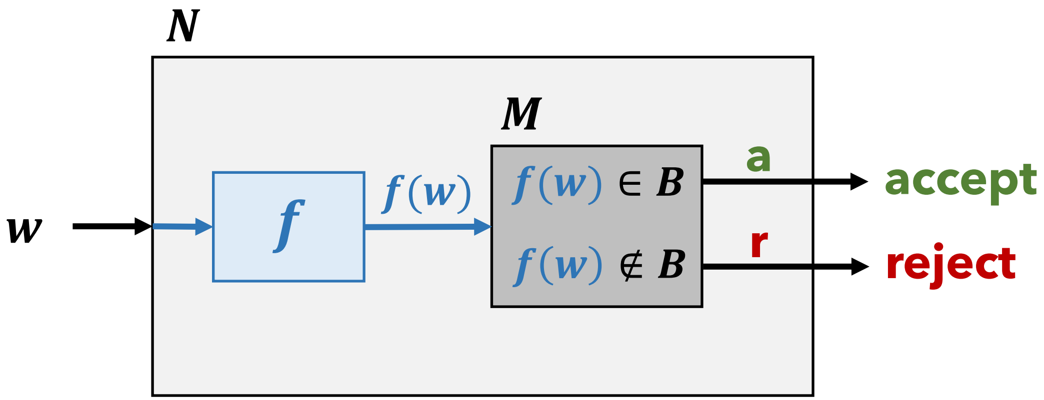




Proof: Given **mapping reduction** f **from** A **to** B and **decider** M **for** B, build a decider N for A as follows:

N = "On input w:

- Compute $f(w)$
- Run M on input $f(w)$
- Output whatever M outputs"



A_{TM} is mapping reducible to $Halt_{TM}$

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$$

$$Halt_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$$

- $A_{TM} \leq_m Halt_{TM}$ if
  - there is a **computable function** f which takes $\langle M, w\rangle$ and outputs $f(\langle M, w\rangle)$
  - $\langle M, w\rangle \in A_{TM}$ if and only if $f(\langle M, w\rangle) \in Halt_{TM}$

- To show $A_{TM} \leq_m Halt_{TM}$, we design a **TM** F that computes f

A_{TM} is mapping reducible to $Halt_{TM}$

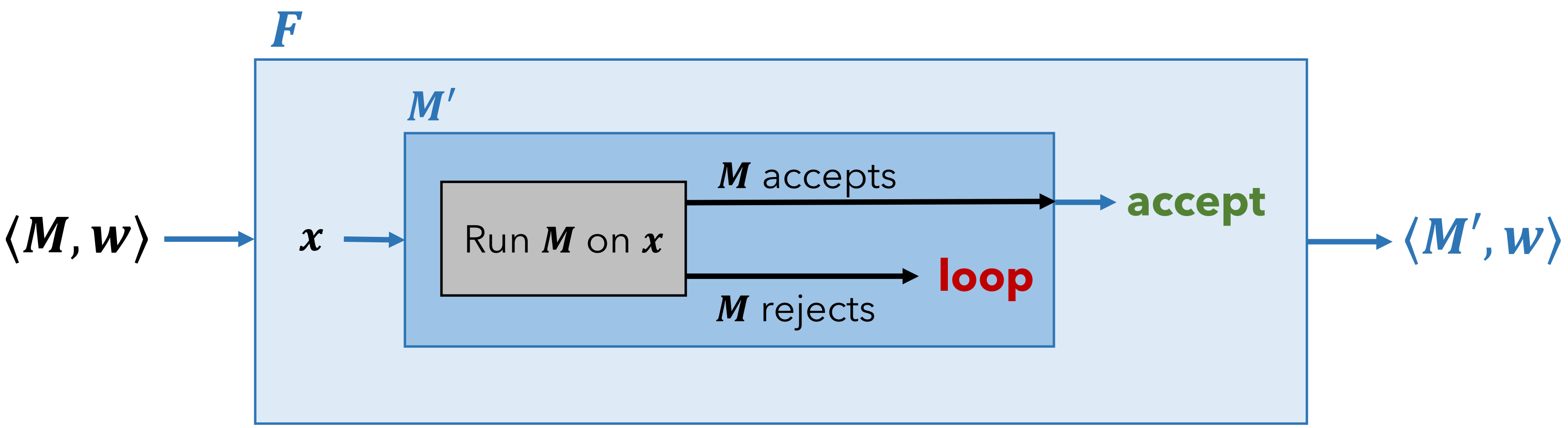
F = "On input $\langle M, w \rangle$

- Construct description of TM M' as follows:

  M' = "On input x, where x is any string

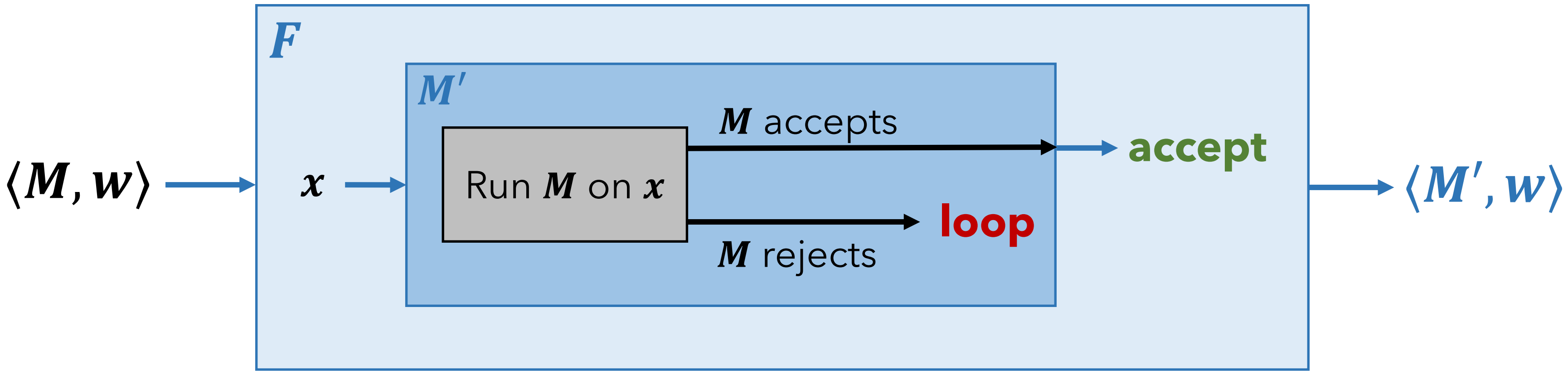
  - Run M on x
  - If M accepts, then **accept**
  - If M rejects, then **enter a loop**"

- Output $\langle M', w \rangle$ "



A_{TM} is mapping reducible to $Halt_{TM}$

- TM F computes f
- Is f a correct mapping reduction from A_{TM} to $Halt_{TM}$?
  - $\langle M, w\rangle \in A_{TM}$ iff $\langle M', w\rangle \in Halt_{TM}$

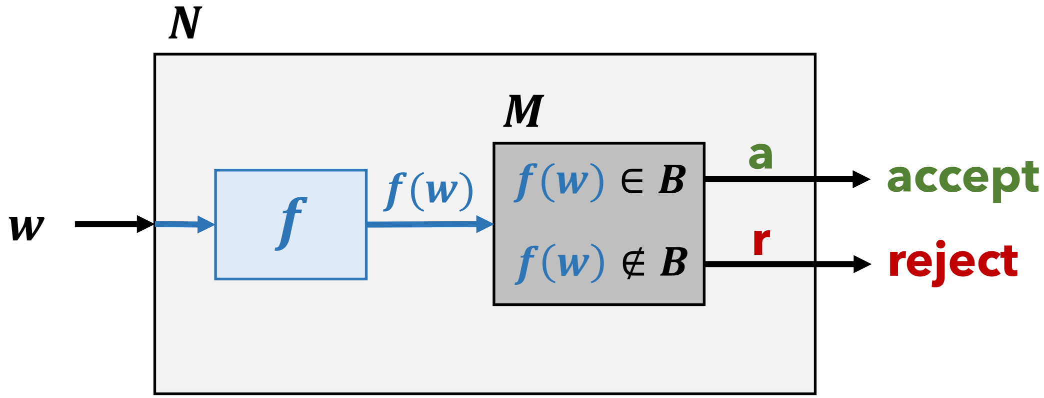


- $\langle M, w\rangle \in A_{TM}$ (M accepts w) $\Rightarrow$ $\langle M', w\rangle \in Halt_{TM}$ (M' halts and accepts w)
- $\langle M, w\rangle \notin A_{TM}$ (M rejects or loops w) $\Rightarrow$ $\langle M', w\rangle \notin Halt_{TM}$ (M' loops on w)
- Therefore, f is a mapping reduction from A_{TM} to $Halt_{TM}$

$Halt_{TM}$ is Undecidable

- We can use **mapping reductions** to show that languages are **undecidable** by using the previous theorem

Theorem: If $A \leq_m B$ and B **is decidable**, then A is decidable



- We have shown $A_{TM} \leq_m Halt_{TM}$
- If $Halt_{TM}$ **was decidable**, then A_{TM} **would be decidable** by the theorem
- Therefore, $Halt_{TM}$ **is undecidable** since A_{TM} is undecidable

Time Complexity

- From now on, we will be only considering **decidable problems**
- Since problems are decidable, we will be analyzing **running time / time complexity** of Turing machines

Running Time / Time Complexity

Definition: Let $\boldsymbol{M}$ be a deterministic single-tape decider. The **running time / time complexity** of $\boldsymbol{M}$ is the function $\boldsymbol{f}: \mathbb{N} \to \mathbb{N}$ where $\boldsymbol{f}(\boldsymbol{n})$ is the maximum number of steps that $\boldsymbol{M}$ uses on any input of length $\boldsymbol{n}$

- If $\boldsymbol{f}(\boldsymbol{n})$ is the running time of $\boldsymbol{M}$, then we say:
  - $\boldsymbol{M}$ **runs in time** $\boldsymbol{f}(\boldsymbol{n})$
  - $\boldsymbol{M}$ is an $\boldsymbol{f}(\boldsymbol{n})$-**time Turing machine**

Time Complexity Class

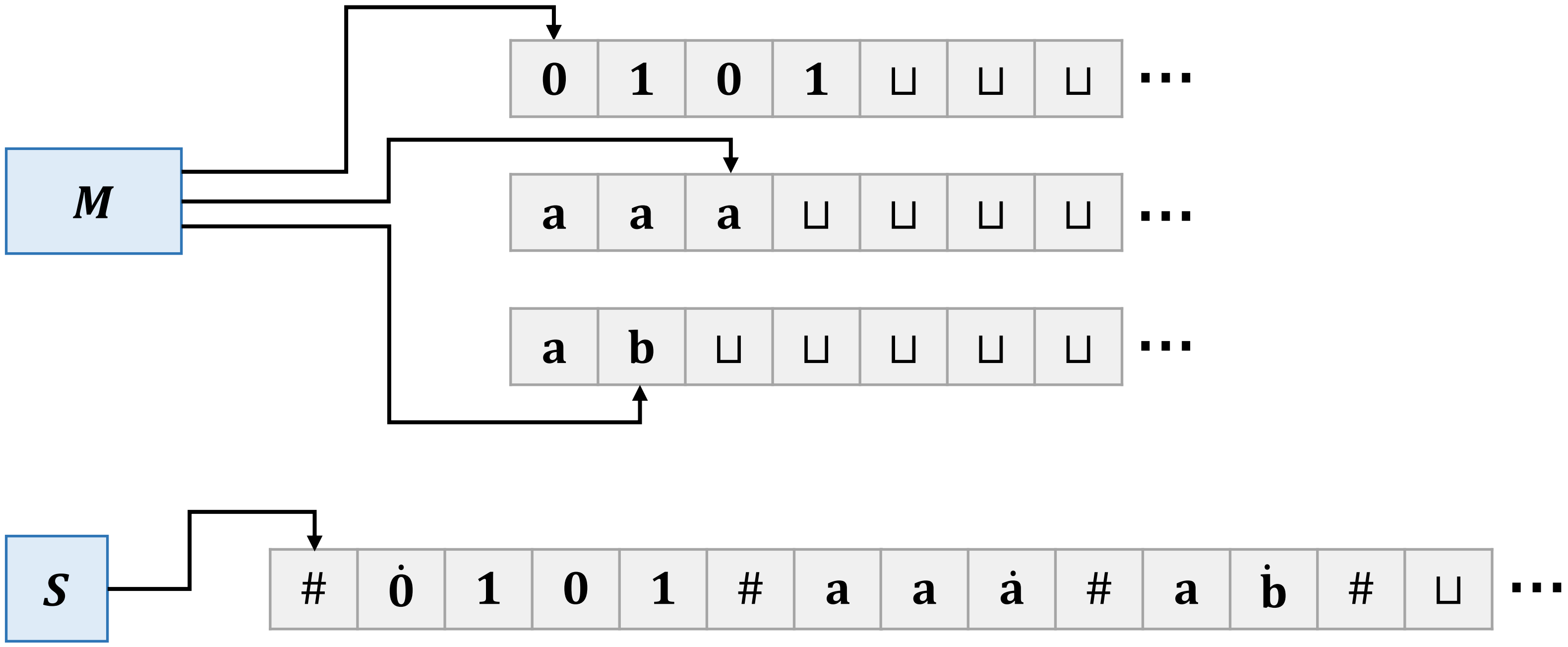
Let $\boldsymbol{t}: \mathbb{N} \rightarrow \mathbb{R}$ be a function

The **time complexity class** $\boldsymbol{TIME(t(n))}$ is the collection of all languages **decidable** by an $\boldsymbol{O(t(n))}$**-time Turing machine**

Example: $\boldsymbol{L} = \{\langle \boldsymbol{G} \rangle \mid \boldsymbol{G} \text{ is a simple undirected connected graph}\}$

- We can use a traversal algorithm to decide if a graph connected
- For graph with $\boldsymbol{n}$ **vertices** and $\boldsymbol{m \leq n^2}$ **edges**, a TM can decide if it is connected or not in $\boldsymbol{O(n^2)}$ time
- $\boldsymbol{L \in TIME(n^2)}$

Multitape TM to Single-tape TM



Multitape TM Time Complexity

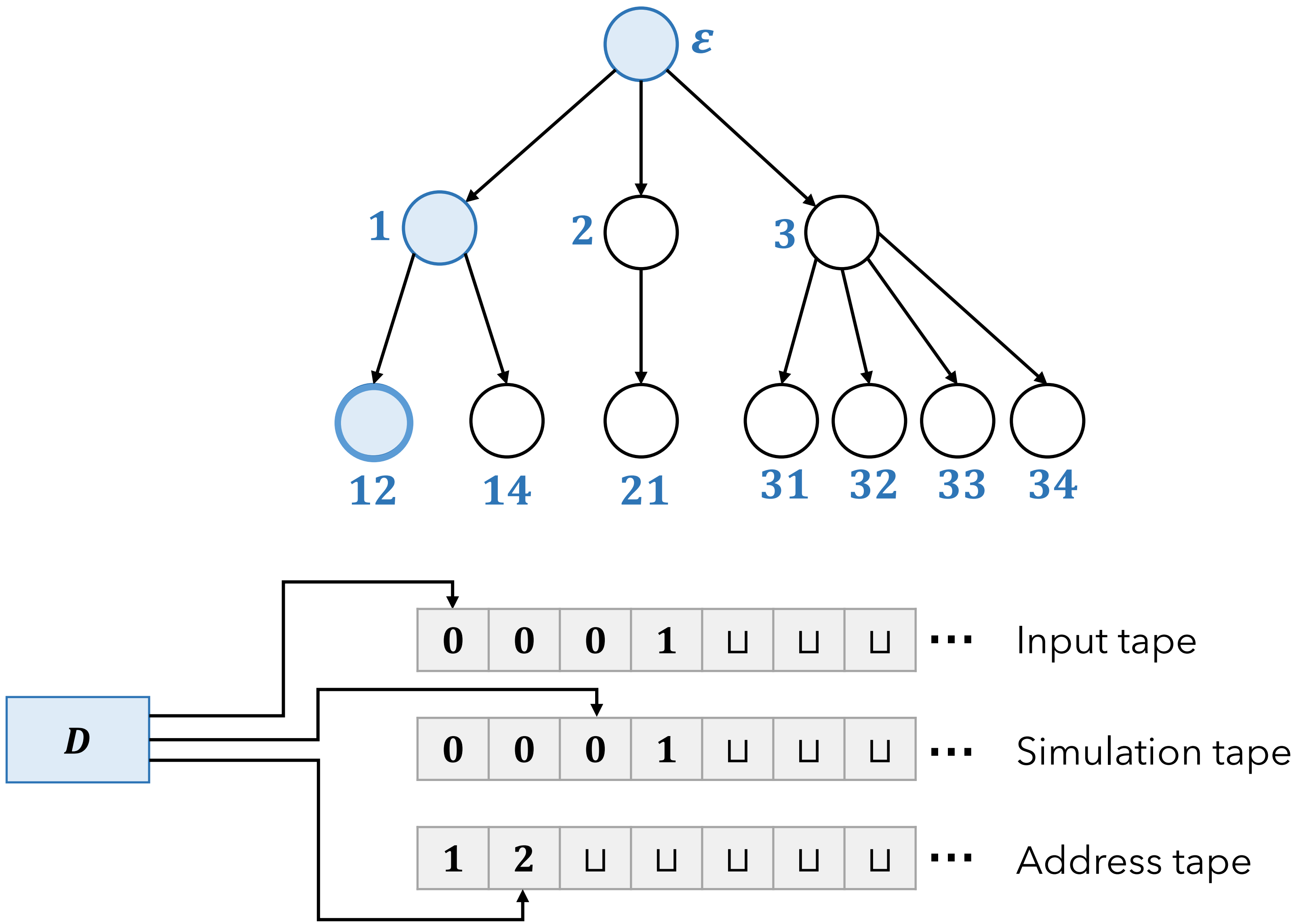
Recall, for every **deterministic single-tape TM**, there exists an equivalent **multitape TM** (and vice versa)

Theorem: Let $t(n)$ be a function, $t(n) \geq n$. Every $t(n)$**-time multitape TM** has an equivalent $O(t^2(n))$**-time single-tape TM**

Proof: Recall conversion from multitape TM with k tapes to single-tape TM

- The contents of the multitape TM is at most $t(n)$
- In the worst case, the equivalent single-tape TM requires $k \cdot t(n)$ time to simulate a step of the multitape TM
- Multitape TM takes $t(n)$ steps
- Therefore, this takes $O(t^2(n))$ time

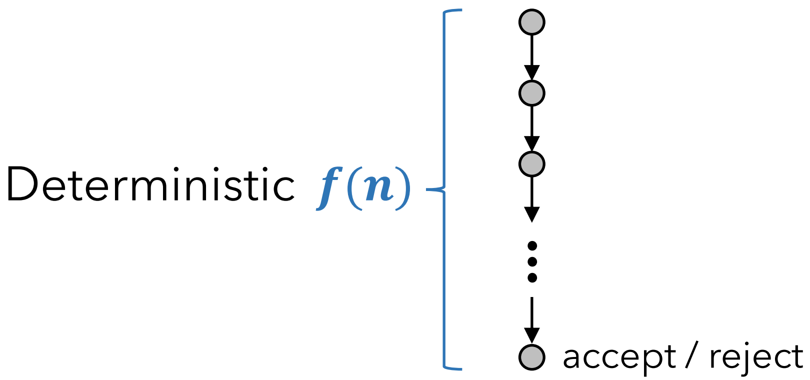
NTM to TM

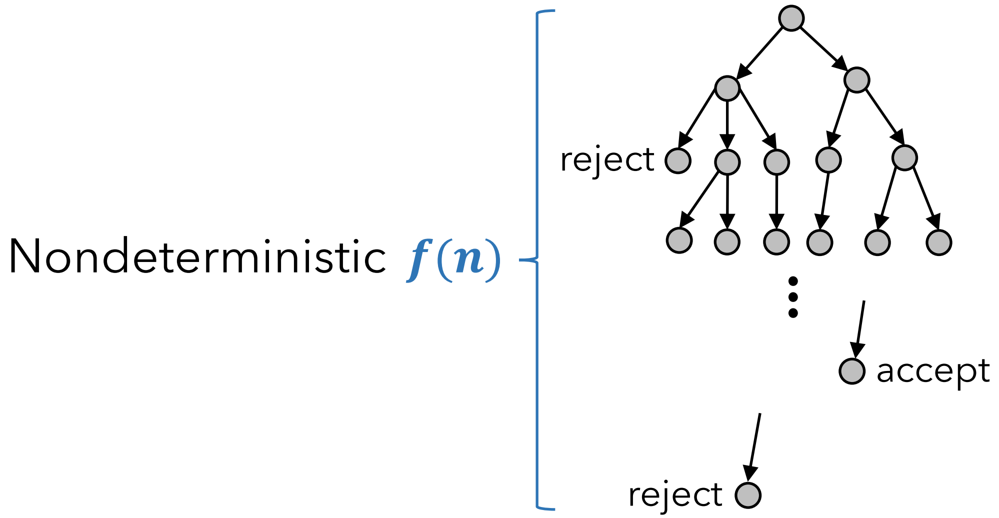

ε
1
2
3
12
14
21
31
32
33
34
D
0 0 0 1 ⊔ ⊔ ⊔ ⋯
Input tape
0 0 0 1 ⊔ ⊔ ⊔ ⋯
Simulation tape
1 2 ⊔ ⊔ ⊔ ⊔ ⊔ ⋯
Address tape

Nondeterministic TM Time Complexity

Recall that for every **deterministic single-tape TM**, there exists an equivalent **nondeterministic single-tape TM** (and vice versa)

- Let $\boldsymbol{N}$ be a non-deterministic single tape decider
- The **running time / time complexity** of $\boldsymbol{N}$ is the function $\boldsymbol{f(n)}$ which is the maximum number of steps that N uses on **any one branch** of its computation on any input of length $\boldsymbol{n}$



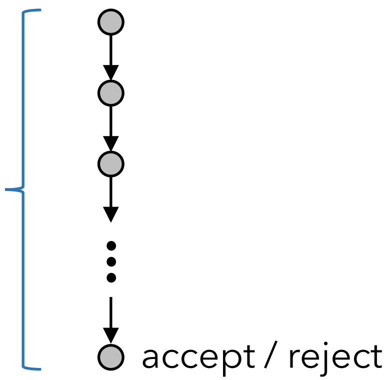


Nondeterministic TM Time Complexity

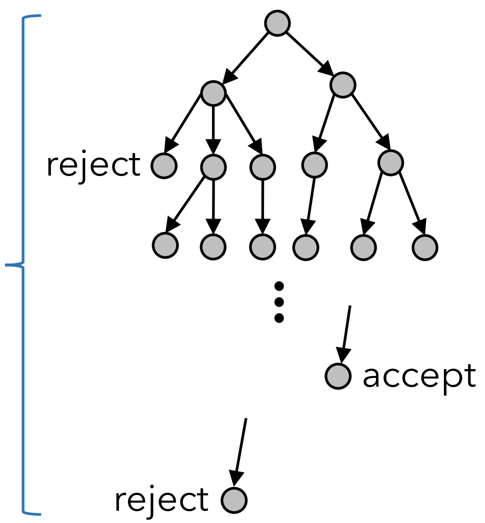
- The running time of **deterministic deciders** is the model for running time when running algorithms on classical computers

- The running time of **nondeterministic deciders** is not intended to correspond to a real-world computing device (purely theoretical)

Deterministic $f(n)$



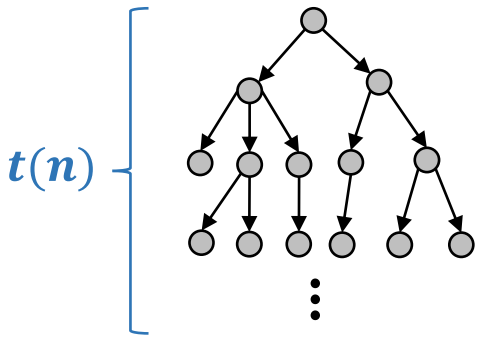
Nondeterministic $f(n)$



Nondeterministic TM Time Complexity

Theorem: Let $t(n)$ be a function, $t(n) \geq n$. Every **$t(n)$-time nondeterministic single-tape TM** has an equivalent **$2^{O(t(n))}$-time deterministic single-tape TM**

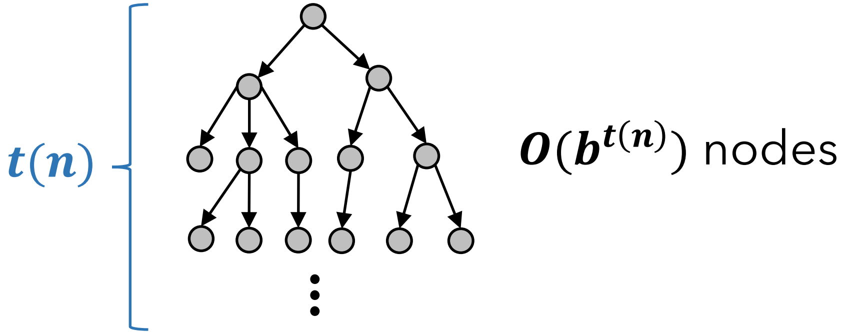
Proof: Recall conversion from nondeterministic single-tape TM to a 3-tape deterministic TM



- Let b be the maximum number of children of any node (possible choices in NTM)
- Total number of leaves $\leq b^{t(n)}$
- Number of nodes in tree $\leq 2 \cdot \#\ \textbf{leaf nodes} = 2 \cdot b^{t(n)}$
- So, there are $O(b^{t(n)})$ nodes to be simulated

Nondeterministic TM Time Complexity

Proof continued...



- It takes $O(t(n))$ time to traverse from root to a node
- Total time to simulate all nodes $= O\left(t(n) \cdot b^{t(n)}\right)$
  - $O\left(t(n) \cdot b^{t(n)}\right) = O\left(t(n) \cdot 2^{\log_2\left(b^{t(n)}\right)}\right) = O\left(t(n) \cdot 2^{\log_2(b) \cdot t(n)}\right) = O\left(t(n) \cdot 2^{O(t(n))}\right)$
  - $O\left(t(n) \cdot 2^{O(t(n))}\right) = O\left(2^{\log_2(\mathrm{t(n)})} \cdot 2^{O(t(n))}\right) = O\left(2^{O(t(n))}\right)$

- It takes $O\left(2^{O(t(n))}\right)$ to simulate an NTM on a multitape TM
- Therefore, it takes $O\left(2^{O(t(n))}\right)^2 = O\left(2^{O(2 \cdot t(n))}\right) = O\left(2^{O(t(n))}\right)$ on a single tape TM

Complexity Class P

$$P = \bigcup_{k} TIME(n^k)$$

$\boldsymbol{P}$ is the class of languages that are **decidable in polynomial time** on a (deterministic single-tape) Turing machine

- If a problem $\boldsymbol{A}$ is in $\boldsymbol{P}$, then $\boldsymbol{A}$ can be solved in polynomial time (there exists an **$\boldsymbol{n^c}$-time algorithm** to solve $\boldsymbol{A}$, for some constant $\boldsymbol{c}$)
- $\boldsymbol{P}$ is often considered to be the class of problems that are **solvable in practice** on a classical computer

Problems in P

CONNECTED GRAPH

- **Input:** A simple undirected graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ where $|\boldsymbol{V}| = \boldsymbol{n}$
- **Question:** Is $\boldsymbol{G}$ connected

- $\boldsymbol{L}_{\textbf{CONNECTED GRAPH}} = \{\langle \boldsymbol{G} \rangle \mid \boldsymbol{G} \text{ is a simple undirected connected graph}\}$
- $\boldsymbol{L}_{\textbf{CONNECTED GRAPH}}$ is in $\boldsymbol{P}$ since a $\boldsymbol{O}(\boldsymbol{n}^2)$**-time** decider can decide (solve) it

PATH

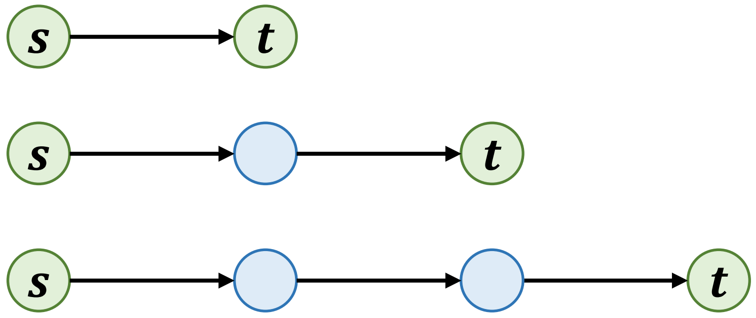
- **Input:** A directed graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ and vertices $\boldsymbol{s}, \boldsymbol{t} \in \boldsymbol{V}$ ($|\boldsymbol{V}| = \boldsymbol{n}$)
- **Question:** Does there exist a directed path from $\boldsymbol{s}$ to $\boldsymbol{t}$ in $\boldsymbol{G}$

- $\boldsymbol{L}_{\textbf{PATH}} = \{\langle \boldsymbol{G}, \boldsymbol{s}, \boldsymbol{t} \rangle \mid \boldsymbol{G} \text{ is a directed graph that has a directed path from } \boldsymbol{s} \text{ to } \boldsymbol{t}\}$
- Is $\boldsymbol{L}_{\textbf{PATH}}$ in $\boldsymbol{P}$?

$PATH \in P$

Suppose we use a brute force algorithm:

- Examine all **potential** paths (all sequences of nodes in V of length at most n):
  - One potential path of length 2
  - $n - 2$ potential paths of length 3
  - $\binom{n-2}{2}$ paths of length 4
  - $\binom{n-2}{n-3}$ paths of length n



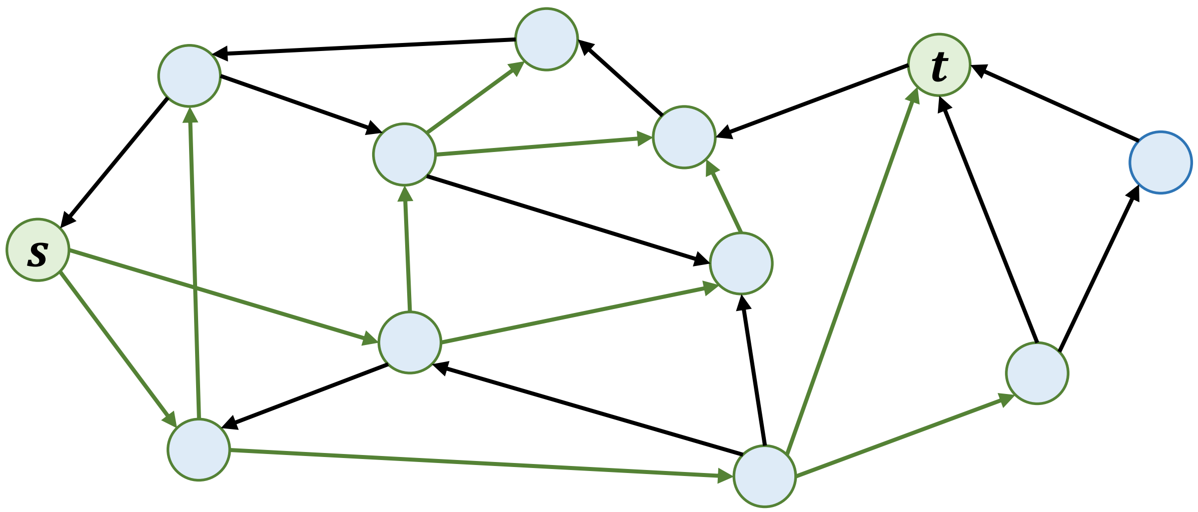
- This algorithm **is not polynomial time**

If this was the only algorithm for PATH, $L_{\mathbf{PATH}}$ would not be in P

$\boldsymbol{PATH \in P}$

However, we have a faster algorithm for PATH:

- Use BFS traversal:
  - Mark all nodes that are reachable from $\boldsymbol{s}$ by directed paths of length $\mathbf{1, 2, 3, \dots} \boldsymbol{n}$



- This is algorithm is **polynomial time**

$\boldsymbol{PATH \in P}$

- Polynomial time decider for $\boldsymbol{L_{PATH}}$ (a bit slower than BFS traversal)

$\boldsymbol{M}$ = "On input $\langle \boldsymbol{G, s, t} \rangle$, where $\boldsymbol{G}$ is a directed graph with nodes $\boldsymbol{s}$ and $\boldsymbol{t}$

  - Mark node $\boldsymbol{s}$
  - Repeat until no new nodes are marked:
    - Scan edges of $\boldsymbol{G}$
    - If edge $(\boldsymbol{u, v})$ is found where $\boldsymbol{u}$ is marked and $\boldsymbol{v}$ is unmarked, then mark $\boldsymbol{v}$
  - If $\boldsymbol{t}$ is marked, **accept**
  - Otherwise, **reject**

- Since $\boldsymbol{L_{\mathrm{PATH}}}$ can be decided in polynomial time (on deterministic single tape TM), $\boldsymbol{L_{PATH} \in P}$

Complexity Class NP

- Next lecture…