Lecture 15: Enumerators and Church-Turing Thesis

CSC 320: Foundations of Computer Science

Quinton Yong

quintonyong@uvic.ca

University of Victoria

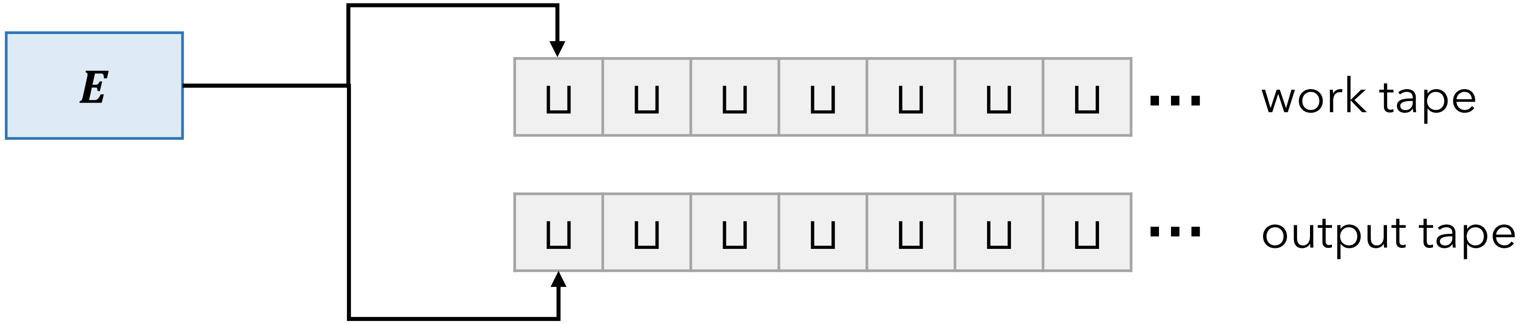
Enumerators

An **enumerator** $\boldsymbol{E}$ is a very different variant of a Turing machine:

- It **receives no input**
- It **outputs every string in its language** $\boldsymbol{L(E)}$

An enumerator is a **2-tape** Turing machine

- Initially, both tapes are blank
- The first tape is for performing **computation**
- The second tape is the **output tape** ("printer")



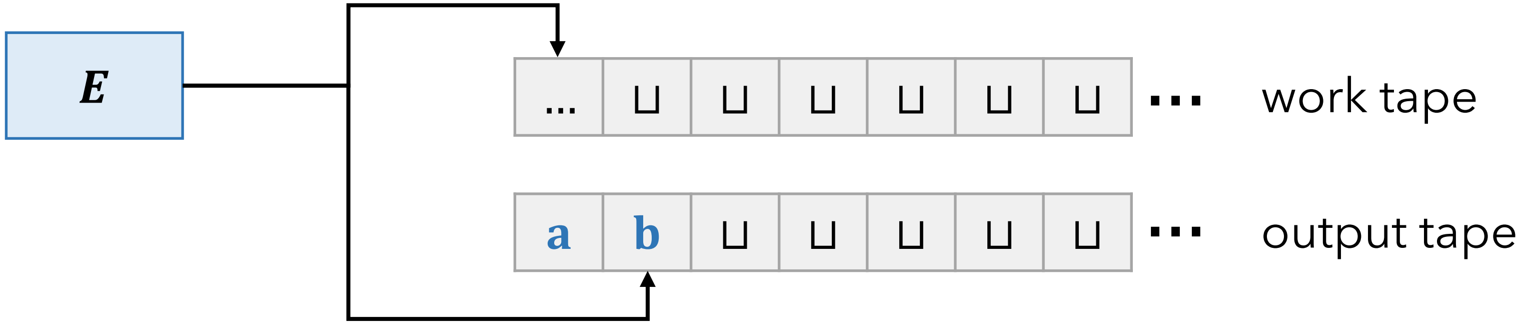
Enumerators

An **enumerator** E is a very different variant of a Turing machine:

- It **receives no input**
- It **outputs every string in its language** $L(E)$

An enumerator is a **2-tape** Turing machine

- Initially, both tapes are blank
- The first tape is for performing **computation**
- The second tape is the **output tape** ("printer")



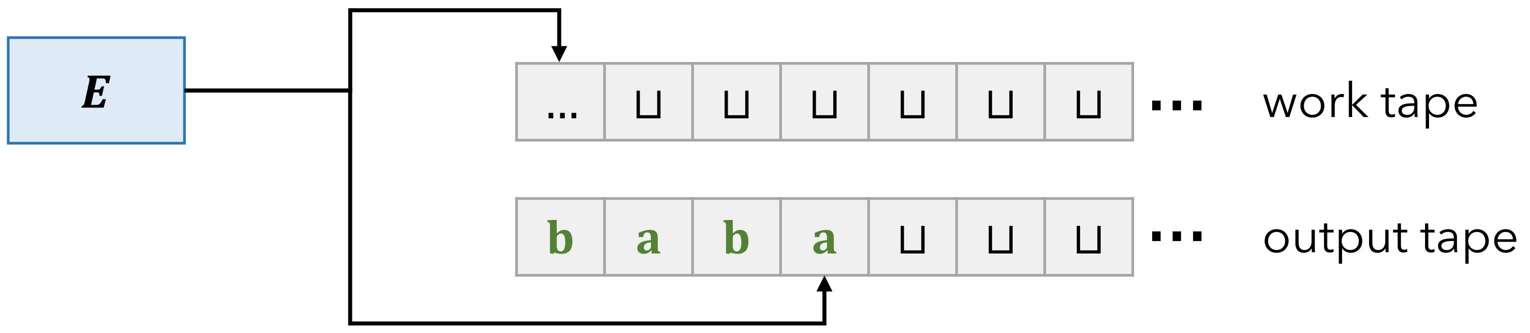
Enumerators

An **enumerator** E is a very different variant of a Turing machine:

- It **receives no input**
- It **outputs every string in its language** $L(E)$

An enumerator is a **2-tape** Turing machine

- Initially, both tapes are blank
- The first tape is for performing **computation**
- The second tape is the **output tape** ("printer")



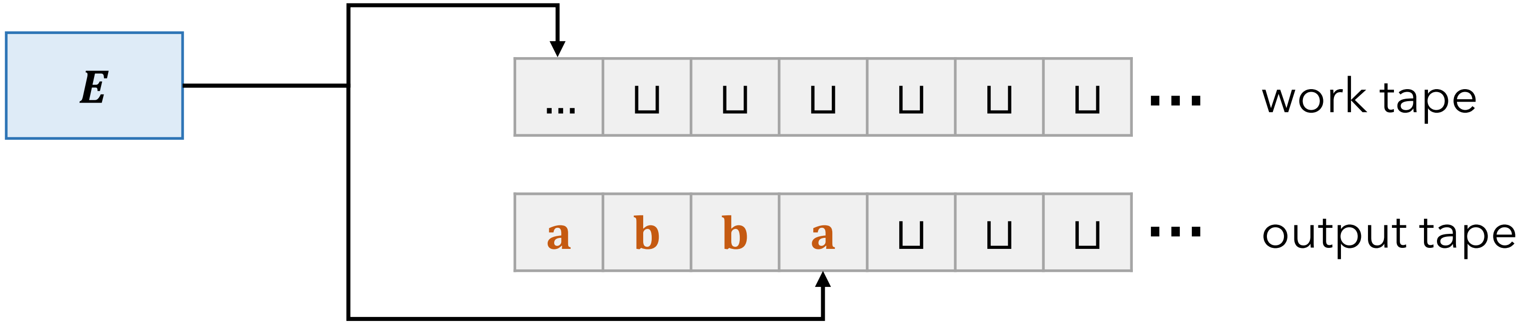
Enumerators

An **enumerator** $\boldsymbol{E}$ is a very different variant of a Turing machine:

- It **receives no input**
- It **outputs every string in its language** $\boldsymbol{L(E)}$

An enumerator is a **2-tape** Turing machine

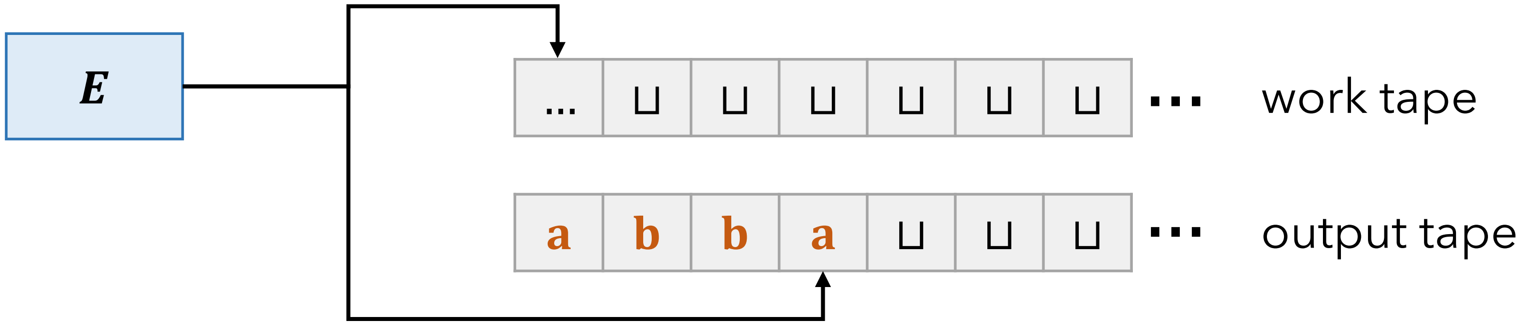
- Initially, both tapes are blank
- The first tape is for performing **computation**
- The second tape is the **output tape** ("printer")



Enumerators

Notes:

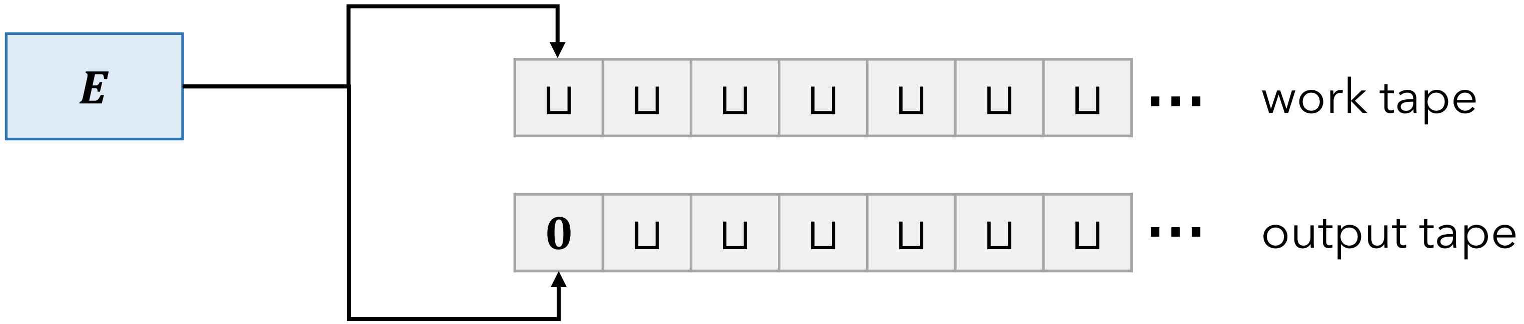
- An enumerator may **output the same string(s)** many times (even an infinite number of times)
- The enumerator $\boldsymbol{E}$ may **loop infinitely** and continue generating output forever if $\boldsymbol{L(E)}$ is infinite
- However, it cannot be stuck in an infinite loop **before** generating all strings in $\boldsymbol{L(E)}$



Enumerator Example

Enumerator $\boldsymbol{E}$ which recognizes $\boldsymbol{L(E)}$ = **binary numbers**:

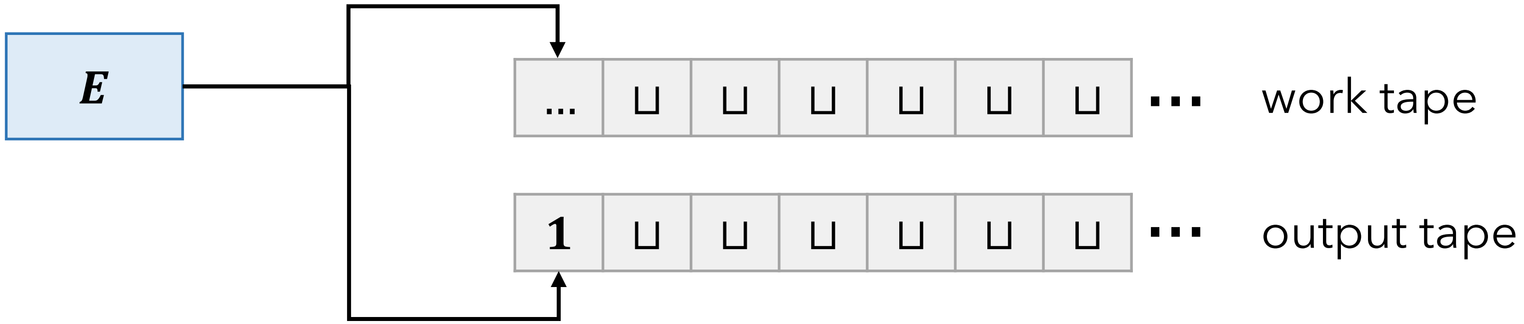
- Write $\boldsymbol{\varepsilon}$ to output tape
- Write $\mathbf{0}$ to output tape
- Increment current binary string by $\mathbf{1}$ to obtain next string and write result to output tape



Enumerator Example

Enumerator $\boldsymbol{E}$ which recognizes $\boldsymbol{L(E)}$ = **binary numbers**:

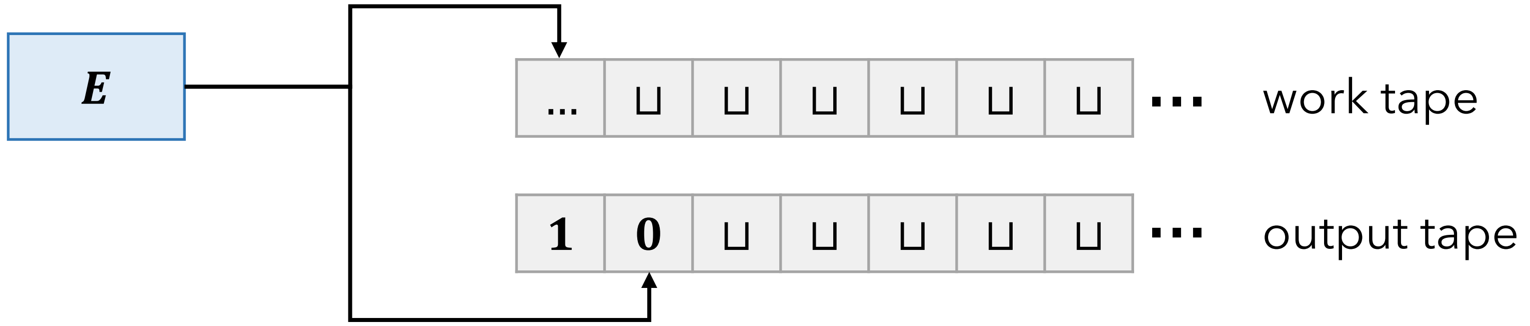
- Write $\boldsymbol{\varepsilon}$ to output tape
- Write $\mathbf{0}$ to output tape
- Increment current binary string by $\mathbf{1}$ to obtain next string and write result to output tape



Enumerator Example

Enumerator $\boldsymbol{E}$ which recognizes $\boldsymbol{L}(\boldsymbol{E})$ = **binary numbers**:

- Write $\boldsymbol{\varepsilon}$ to output tape
- Write $\mathbf{0}$ to output tape
- Increment current binary string by $\mathbf{1}$ to obtain next string and write result to output tape



Equivalence of Enumerators

Theorem: A language is **Turing-recognizable** if and only if some **enumerator** outputs it.

Proof:

$\Leftarrow$ Given **enumerator** E, build **TM** M that recognizes $L(E)$

$\Rightarrow$ Given **TM** M, design **enumerator** E that outputs $L(M)$

Equivalence of Enumerators

$\Leftarrow$ Given **enumerator** E, build **TM** M that recognizes $L(E)$

- Build a (multitape) **TM** M which accepts input strings in $L(E)$ as follows:

  M = "On input w:
  - Run **enumerator** E
  - Every time E outputs a string s, compare it with w
  - If $w = s$, then **accept**
  - If E terminates without outputting a string $s = w$, then **reject**"

- M **accepts** strings in $L(E)$ and **rejects** or **loops forever** on strings not in $L(E)$
- Therefore, M recognizes $L(E)$

Equivalence of Enumerators

$\Rightarrow$ Given **TM** M, design **enumerator** E that outputs $L(M)$

- Build an **enumerator** E which outputs all strings in $L(M)$
- Let $s_1, s_2, s_3, \ldots$ be a list of all strings in Σ^*

Incorrect construction of E:

E = "For each string $s_k = s_1, s_2, s_3, \ldots$
- Run M on input s_k
- If M accepts, then print s_k"

Problem: If M loops infinitely on any string s_k, then the enumerator will be stuck in an infinite loop before printing all strings in $L(M)$

Equivalence of Enumerators

$\Rightarrow$ Given **TM** M, design **enumerator** E that outputs $L(M)$

- Build an **enumerator** E which outputs all strings in $L(M)$
- Let $s_1, s_2, s_3, \ldots$ be a list of all strings in Σ^*

Another incorrect construction of E:

E = "For each $i = 1, 2, 3, \ldots$

- Run M on for i **steps** on each input $s_k = s_1, s_2, s_3, \ldots$
- If M accepts, then print s_k"

Problem: Since Σ^* is infinite, we would be stuck in an infinite loop of running just **1 step** on each $s_1, s_2, s_3, \ldots \in \Sigma^*$

Equivalence of Enumerators

$\Rightarrow$ Given **TM** M, design **enumerator** E that outputs $L(M)$

- Build an **enumerator** E which outputs all strings in $L(M)$
- Let $s_1, s_2, s_3, \ldots$ be a list of all strings in Σ^*

Correct construction of E:

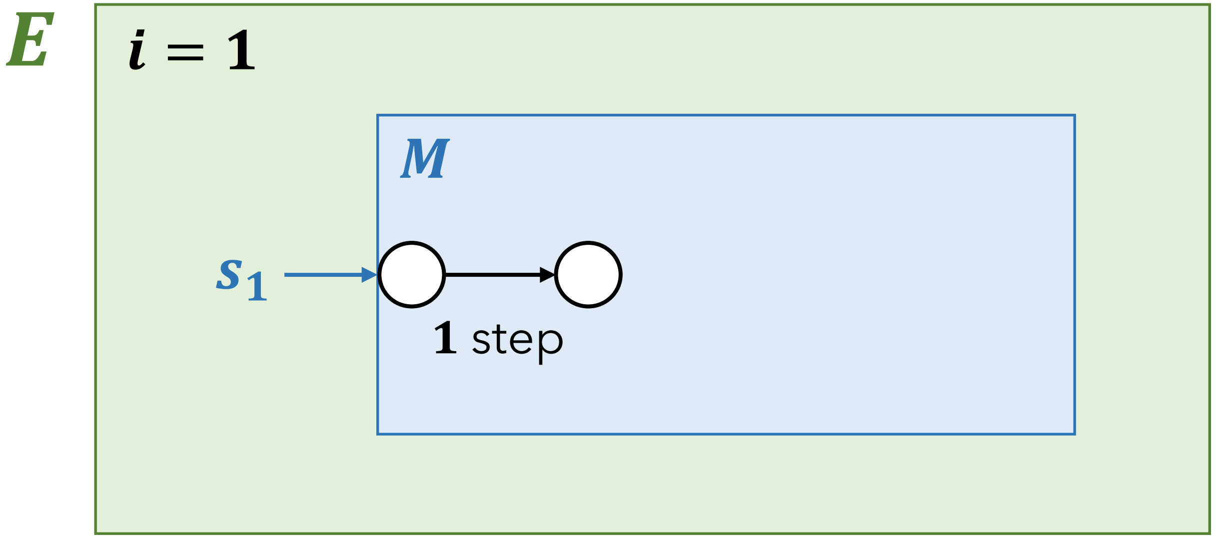
E = "For each $i = 1, 2, 3, \ldots$
- Run M on for i **steps** on each input s_k from $s_1, s_2, s_3, \ldots, s_i$
- If M accepts, then print s_k"

Equivalence of Enumerators

Correct construction of E:

E = "For each $i = 1, 2, 3, \dots$

- Run M on for i **steps** on each input s_k from $s_1, s_2, s_3, \dots, s_i$
- If M accepts, then print s_k"

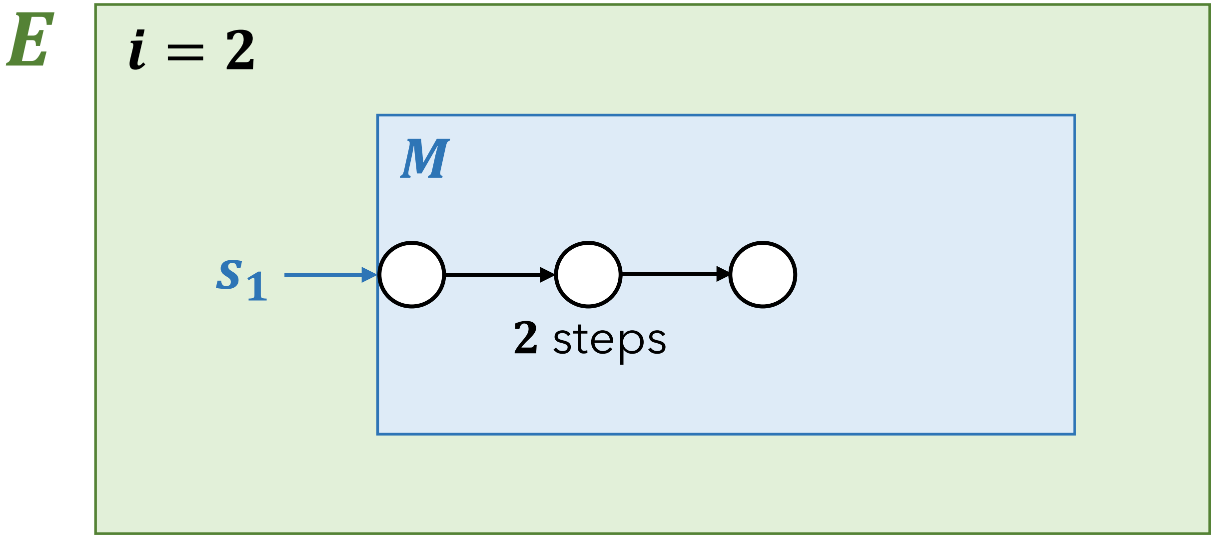


Equivalence of Enumerators

Correct construction of E:

E = "For each $i = 1, 2, 3, \ldots$

- Run M on for i **steps** on each input s_k from $s_1, s_2, s_3, \ldots, s_i$
- If M accepts, then print s_k"

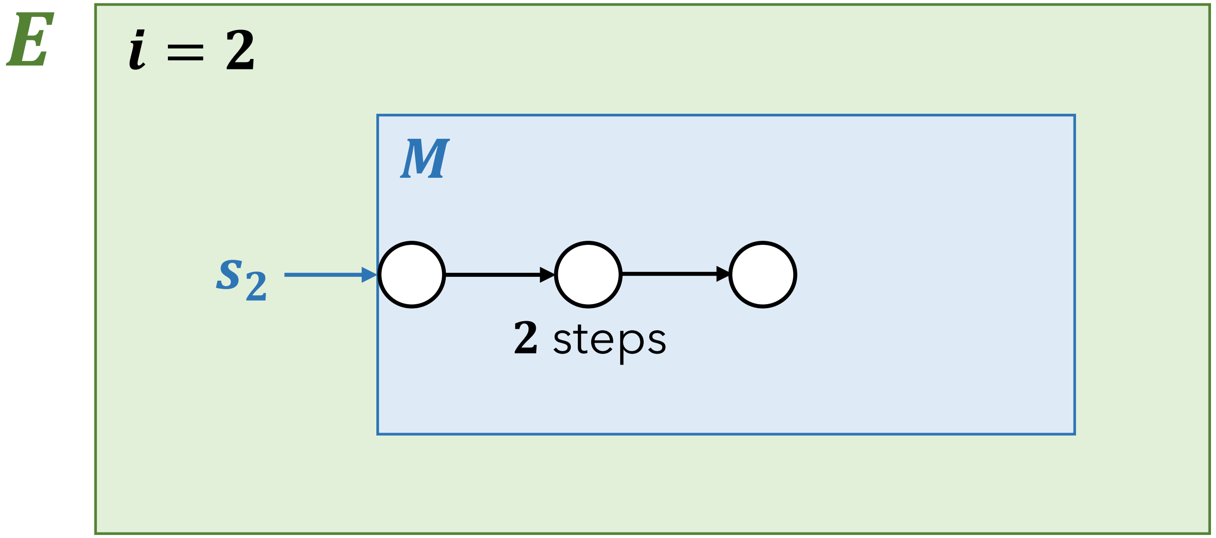


Equivalence of Enumerators

Correct construction of E:

E = "For each $i = 1, 2, 3, \ldots$

- Run M on for i **steps** on each input s_k from $s_1, s_2, s_3, \ldots, s_i$
- If M accepts, then print s_k"

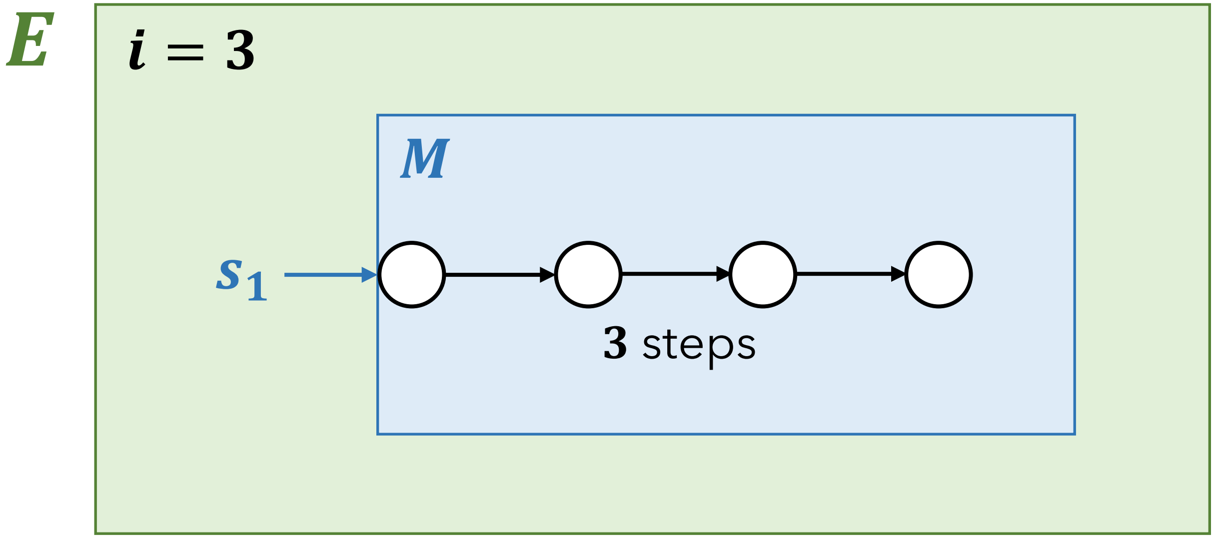


Equivalence of Enumerators

Correct construction of E:

E = "For each $\boldsymbol{i} = \mathbf{1, 2, 3, \ldots}$

- Run $\boldsymbol{M}$ on for $\boldsymbol{i}$ **steps** on each input $\boldsymbol{s_k}$ from $\boldsymbol{s_1, s_2, s_3, \ldots, s_i}$
- If $\boldsymbol{M}$ accepts, then print $\boldsymbol{s_k}$"

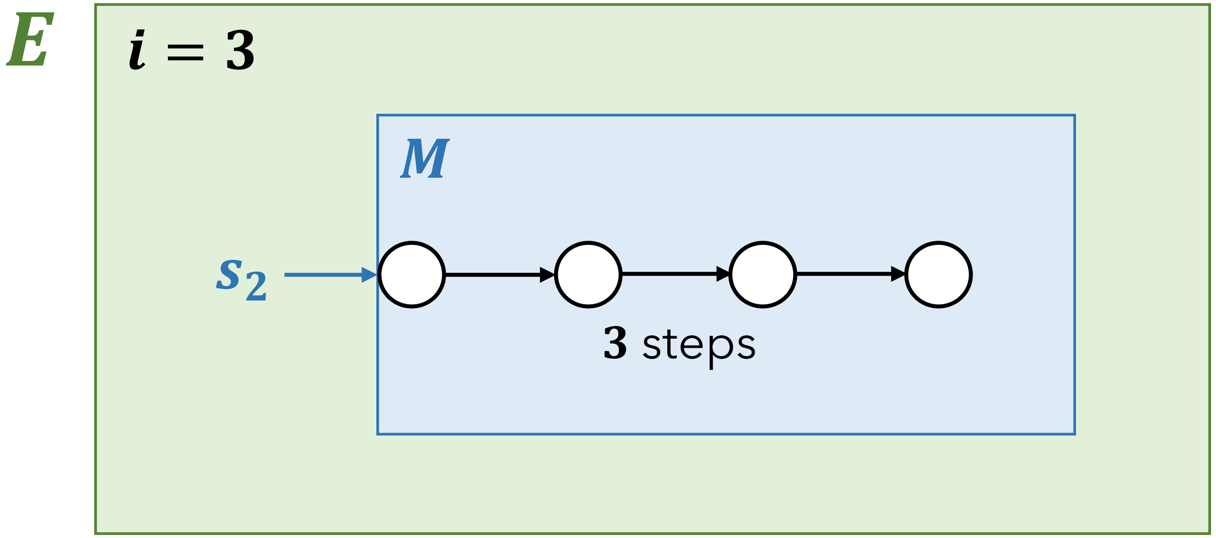


Equivalence of Enumerators

Correct construction of E:

E = "For each $i = 1, 2, 3, \ldots$

- Run M on for i **steps** on each input s_k from $s_1, s_2, s_3, \ldots, s_i$
- If M accepts, then print s_k"

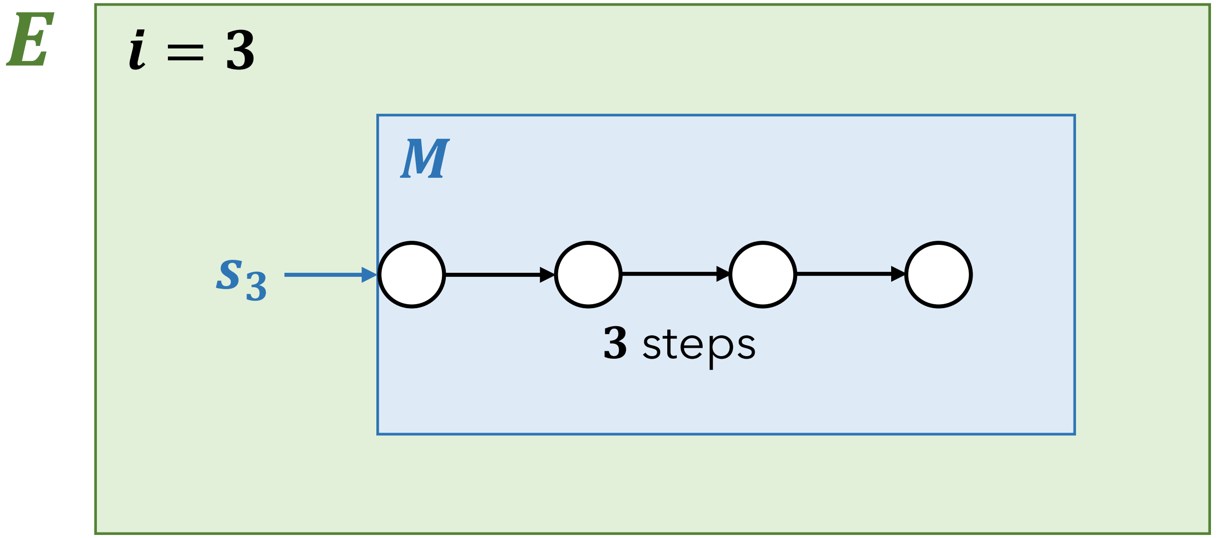


Equivalence of Enumerators

Correct construction of E:

E = "For each $i = 1, 2, 3, \ldots$

- Run M on for i **steps** on each input s_k from $s_1, s_2, s_3, \ldots, s_i$
- If M accepts, then print s_k"



Equivalence of Enumerators

Correct construction of E:

E = "For each $i = 1, 2, 3, \ldots$

- Run M on for i **steps** on each input s_k from $s_1, s_2, s_3, \ldots, s_i$
- If M accepts, then print s_k"

- For each $s_k \in \Sigma^*$
  - We will try to execute s_k on M when $i = k$ and will run for k steps
  - If $s_k \in L(M)$, then eventually it will run for enough steps to accept
- Since we are always running a **finite number of steps** for each string, we will **not get stuck** in an infinite loop before getting to some other string
- Therefore, E outputs every string in $L(M)$

Turing Machine Equivalence Corollary

A language L is **Turing-recognizable**

if and only if

some **single-tape Turing machine** recognizes it

if and only if

some **multitape Turing machine** recognizes it

if and only if

some **nondeterministic Turing machine** recognizes it

if and only if

some **enumerator** outputs it

Definition of Algorithm

Algorithm:

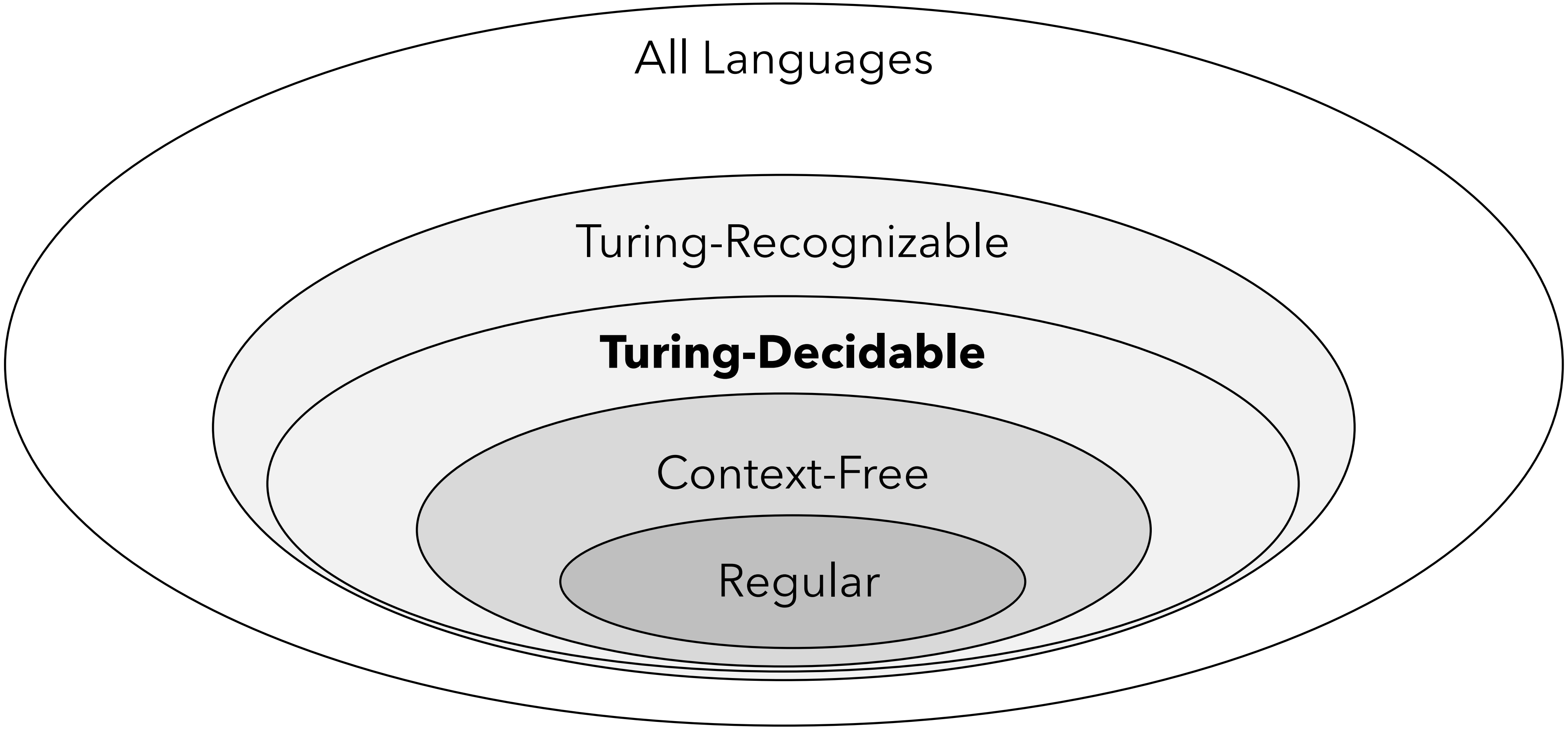
- Finite number of exact instructions (each instruction is of finite length)
- Produces the desired result in a **finite number of steps** (always halts)

The definition of an algorithm is very similar to the definition of a **decider**…

Church-Turing Thesis

$$\boxed{\textbf{ALGORITHMS = DECIDERS}}$$

A problem can be solved following an **algorithm**
if and only if
it is **decidable** by a Turing machine.



- Turing-decidable languages are **problems that can be solved** using an algorithm (by a computer)
- Languages which **are not Turing-decidable** are problems which cannot be solved using an algorithm or by a computer…

Halting Problem

Does there exist a decider / algorithm, when **given a program with any input**, determines **if the program halts**?