

# **CSC363 Tutorial #3**

## **Decidable and Recognizable sets**

February 1, 2023

# Things covered in this tutorial

# Things covered in this tutorial

- ★ What's a language?

# Things covered in this tutorial

- ★ What's a language?
- ★ What's a decidable language? What's a recognizable language?

# Things covered in this tutorial

- ★ What's a language?
- ★ What's a decidable language? What's a recognizable language?
- ★ How many synonyms are there for “decidable” and “recognizable”?

# Things covered in this tutorial

- ★ What's a language?
- ★ What's a decidable language? What's a recognizable language?
- ★ How many synonyms are there for “decidable” and “recognizable”?
- ★ How do I show that something is decidable or recognizable?

# Things covered in this tutorial

- ★ What's a language?
  - ★ What's a decidable language? What's a recognizable language?
  - ★ How many synonyms are there for “decidable” and “recognizable”?
  - ★ How do I show that something is decidable or recognizable?
  - ★ What's an enumerator?
  - ★ Can I get a hint for A2?
-

# Things covered in this tutorial

- ★ What's a language?
  - ★ What's a decidable language? What's a recognizable language?
  - ★ How many synonyms are there for “decidable” and “recognizable”?
  - ★ How do I show that something is decidable or recognizable?
  - ★ What's an enumerator?
  - ★ Can I get a hint for A2? No, but you may cite these slides for your homework. (You still have to prove everything.)<sup>1</sup>
-



# Things covered in this tutorial

- ★ What's a language?
  - ★ What's a decidable language? What's a recognizable language?
  - ★ How many synonyms are there for “decidable” and “recognizable”?
  - ★ How do I show that something is decidable or recognizable?
  - ★ What's an enumerator?
  - ★ Can I get a hint for A2? No, but you may cite these slides for your homework. (You still have to prove everything.)<sup>1</sup>
-

# Things covered in this tutorial

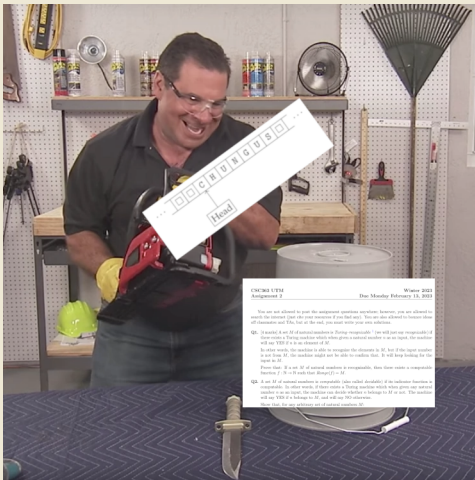
- ★ What's a language?
- ★ What's a decidable language? What's a recognizable language?
- ★ How many synonyms are there for “decidable” and “recognizable”?
- ★ How do I show that something is decidable or recognizable?
- ★ What's an enumerator?
- ★ Can I get a hint for A2? No, but you may cite these slides for your homework. (You still have to prove everything.)<sup>1</sup>

---

<sup>1</sup>Citation: Paul “sjorv” Zhang. “Sussy Tutorial #3. Decidable and Recognizable Sets.”.

**EXTRACT\_MONEY\_FROM\_STUDENTS**

# Turing Machines: Review



CSC363 UTM  
Assignment 2

Winter 2023  
Due: Monday February 13, 2023

You are not allowed to post the assignment questions anywhere; however, you are allowed to search the internet (just do your best work). You are also allowed to discuss ideas with classmates and TAs, but at the end, you must write your own solution.

Q1. Is marked. A set  $M$  of natural numbers is Turing-recognizable (we will just use recognizable) if there exists a Turing machine which when given a natural number  $n$  as an input, the machine will say YES if  $n$  is an element of  $M$ .

In other words, the machine is able to recognize the elements in  $M$ , but if the input number is not from  $M$ , the machine might not be able to confirm this. It will keep looking for the input in  $M$ .

Prove that: If a set  $M$  of natural numbers is recognizable, then there exists a computable function  $f$  ( $f$  is a fix-point function) such that  $f(n) \in M$  for all  $n \in \mathbb{N}$ .

Q2. A set  $M$  of natural numbers is co-recognizable (also called decidable) if its indicator function is computable. In other words, if there exists a Turing machine which when given any natural number  $n$  as an input, the machine can decide whether  $n$  belongs to  $M$  or not. The machine will say YES if  $n$  belongs to  $M$ , and will say NO otherwise.

Prove that: For any arbitrary set of natural numbers,  $M$ .

# Turing Machines: Review

Let  $T$  be a Turing machine. Recall that given a string input  $w \in \Sigma^*$ , one of the following will happen:

# Turing Machines: Review

Let  $T$  be a Turing machine. Recall that given a string input  $w \in \Sigma^*$ , one of the following will happen:

- ★  $T(w)$  **accepts**, and outputs whatever is left on the tape.

# Turing Machines: Review

Let  $T$  be a Turing machine. Recall that given a string input  $w \in \Sigma^*$ , one of the following will happen:

- ★  $T(w)$  **accepts**, and outputs whatever is left on the tape.
- ★  $T(w)$  **rejects**, and outputs whatever is left on the tape.

# Turing Machines: Review

Let  $T$  be a Turing machine. Recall that given a string input  $w \in \Sigma^*$ , one of the following will happen:

- ★  $T(w)$  **accepts**, and outputs whatever is left on the tape.
- ★  $T(w)$  **rejects**, and outputs whatever is left on the tape.
- ★  $T(w)$  **loops**.

# Turing Machines: Review

Let  $T$  be a Turing machine. Recall that given a string input  $w \in \Sigma^*$ , one of the following will happen:

- ★  $T(w)$  **accepts**, and outputs whatever is left on the tape.
- ★  $T(w)$  **rejects**, and outputs whatever is left on the tape.
- ★  $T(w)$  **loops**.

## Church-Turing Thesis:

Anything that your computer can do, so can a Turing machine.



# Turing Machines: Review

Let  $T$  be a Turing machine. Recall that given a string input  $w \in \Sigma^*$ , one of the following will happen:

- ★  $T(w)$  **accepts**, and outputs whatever is left on the tape.
- ★  $T(w)$  **rejects**, and outputs whatever is left on the tape.
- ★  $T(w)$  **loops**.

## Church-Turing Thesis:

Anything that your computer can do, so can a Turing machine.

For this tutorial, we will use the Church-Turing Thesis to write pseudocode instead of low-level TMs.

# Languages

If you still remember from CSC263, recall what a *language* is.

# Languages

If you still remember from CSC263, recall what a *language* is.

Let  $\Sigma$  be an **alphabet**. Then  $\Sigma^*$  is the set of all finite strings using characters from  $\Sigma$ .

# Languages

If you still remember from CSC263, recall what a *language* is.

Let  $\Sigma$  be an **alphabet**. Then  $\Sigma^*$  is the set of all finite strings using characters from  $\Sigma$ . A **language** is any subset of  $\Sigma^*$ .



68 DAVID LIU UTM EDITS BY DANIEL ZINGARO

$[a, b, c]$ :

- $\{e, a, b, c\}$
- $\{w \in \{a, b, c\}^* \mid |w| \leq 3\}$
- $\{w \in \{a, b, c\}^* \mid w \text{ has the same number of } a\text{'s and } c\text{'s}\}$
- $\{w \in \{a, b, c\}^* \mid w \text{ can be found in an English dictionary}\}$

These are pretty mundane examples. Somewhat surprisingly, however, this notion of languages also captures solutions to computational problems. Consider the following languages over the alphabet of all standard ASCII characters.

- $L_1 = \{A \mid A \text{ is a string representation of a sorted list of numbers}\}$
- $L_2 = \{(A, x) \mid A \text{ is a list of numbers, } x \text{ is the minimum of } A\}$
- $L_3 = \{(a, b, c) \mid a, b, c \in \mathbb{N} \text{ and } \gcd(a, b) = c\}$

# Decidable and Recognizable

Let  $L \subseteq \Sigma^*$  be a language.

# Decidable and Recognizable

Let  $L \subseteq \Sigma^*$  be a language. We say  $L$  is **decidable** if there is a Turing machine  $M$  (called the **decider**) such that for any input  $s \in \Sigma^*$ :

# Decidable and Recognizable

Let  $L \subseteq \Sigma^*$  be a language. We say  $L$  is **decidable** if there is a Turing machine  $M$  (called the **decider**) such that for any input  $s \in \Sigma^*$ :

- ★ If  $s \in L$ , then  $M$  accepts  $s$ .

# Decidable and Recognizable

Let  $L \subseteq \Sigma^*$  be a language. We say  $L$  is **decidable** if there is a Turing machine  $M$  (called the **decider**) such that for any input  $s \in \Sigma^*$ :

- ★ If  $s \in L$ , then  $M$  accepts  $s$ .
- ★ If  $s \notin L$ , then  $M$  rejects  $s$ .



# Decidable and Recognizable

Let  $L \subseteq \Sigma^*$  be a language. We say  $L$  is **decidable** if there is a Turing machine  $M$  (called the **decider**) such that for any input  $s \in \Sigma^*$ :

- ★ If  $s \in L$ , then  $M$  accepts  $s$ .
- ★ If  $s \notin L$ , then  $M$  rejects  $s$ .

# Decidable and Recognizable

Let  $L \subseteq \Sigma^*$  be a language. We say  $L$  is **decidable** if there is a Turing machine  $M$  (called the **decider**) such that for any input  $s \in \Sigma^*$ :

- ★ If  $s \in L$ , then  $M$  accepts  $s$ .
- ★ If  $s \notin L$ , then  $M$  rejects  $s$ .

We say  $L$  is **recognizable** if there is a Turing machine  $M$  (called the **recognizer**) such that for any input  $s \in \Sigma^*$ :

- ★ If  $s \in L$ , then  $M$  accepts  $s$ .
- ★ If  $s \notin L$ , then  $M$  rejects or loops on  $s$ .

# Decidable and Recognizable

Let  $L \subseteq \Sigma^*$  be a language. We say  $L$  is **decidable** if there is a Turing machine  $M$  (called the **decider**) such that for any input  $s \in \Sigma^*$ :

- ★ If  $s \in L$ , then  $M$  accepts  $s$ .
- ★ If  $s \notin L$ , then  $M$  rejects  $s$ .

We say  $L$  is **recognizable** if there is a Turing machine  $M$  (called the **recognizer**) such that for any input  $s \in \Sigma^*$ :

- ★ If  $s \in L$ , then  $M$  accepts  $s$ .
- ★ If  $s \notin L$ , then  $M$  rejects or loops on  $s$ .

**Note the difference!**  $M$  always has to halt in order to be a decider.

# Decidable and Recognizable

Let  $L \subseteq \Sigma^*$  be a language. We say  $L$  is **decidable** if there is a Turing machine  $M$  (called the **decider**) such that for any input  $s \in \Sigma^*$ :

- ★ If  $s \in L$ , then  $M$  accepts  $s$ .
- ★ If  $s \notin L$ , then  $M$  rejects  $s$ .

---

**Example:**

# Decidable and Recognizable

Let  $L \subseteq \Sigma^*$  be a language. We say  $L$  is **decidable** if there is a Turing machine  $M$  (called the **decider**) such that for any input  $s \in \Sigma^*$ :

- ★ If  $s \in L$ , then  $M$  accepts  $s$ .
- ★ If  $s \notin L$ , then  $M$  rejects  $s$ .

---

**Example:** Let  $\Sigma = \{0, 1\}$ , and  $L = \{0^n 1^n : n \in \mathbb{N}\}$ .<sup>2</sup> Show that  $L$  is decidable.

---

<sup>2</sup>That is,  $L = \{\epsilon, 01, 0011, 000111, \dots\}$ .

# Decidable and Recognizable

Let  $L \subseteq \Sigma^*$  be a language. We say  $L$  is **decidable** if there is a Turing machine  $M$  (called the **decider**) such that for any input  $s \in \Sigma^*$ :

- ★ If  $s \in L$ , then  $M$  accepts  $s$ .
- ★ If  $s \notin L$ , then  $M$  rejects  $s$ .

---

**Example:** Let  $\Sigma = \{0, 1\}$ , and  $L = \{0^n 1^n : n \in \mathbb{N}\}$ .<sup>2</sup> Show that  $L$  is decidable.

*Proof.* The following is the pseudocode of a decider  $M$  for  $L$ :

---

<sup>2</sup>That is,  $L = \{\epsilon, 01, 0011, 000111, \dots\}$ .

# Decidable and Recognizable

Let  $L \subseteq \Sigma^*$  be a language. We say  $L$  is **decidable** if there is a Turing machine  $M$  (called the **decider**) such that for any input  $s \in \Sigma^*$ :

- ★ If  $s \in L$ , then  $M$  accepts  $s$ .
- ★ If  $s \notin L$ , then  $M$  rejects  $s$ .

---

**Example:** Let  $\Sigma = \{0, 1\}$ , and  $L = \{0^n 1^n : n \in \mathbb{N}\}$ .<sup>2</sup> Show that  $L$  is decidable.

*Proof.* The following is the pseudocode of a decider  $M$  for  $L$ :

$M(w)$ :

```
n = length(w)
if n is odd:
    reject
for i in 0 to (n/2 - 1):
    if w[n/2] != 0 or w[n/2 + i] != 1:
        reject
accept
```

---

<sup>2</sup>That is,  $L = \{\epsilon, 01, 0011, 000111, \dots\}$ .

# Decidable and Recognizable

Decidable languages are also known as recursive languages.



# Decidable and Recognizable

Decidable languages are also known as recursive languages.

Here are some synonyms for recognizable:

- ★ Listable.
- ★ Recursively enumerable (r.e.).
- ★ Computably enumerable (c.e.).
- ★ Partially decidable.
- ★  $\Sigma_1^0$ .

# Worksheet time!

Try doing Exercise -1. Here's the definition of *decidable* again:

Let  $L \subseteq \Sigma^*$  be a language. We say  $L$  is **decidable** if there is a Turing machine  $M$  (called the **decider**) such that for any input  $s \in \Sigma^*$ :

# Worksheet time!

Try doing Exercise -1. Here's the definition of *decidable* again:

Let  $L \subseteq \Sigma^*$  be a language. We say  $L$  is **decidable** if there is a Turing machine  $M$  (called the **decider**) such that for any input  $s \in \Sigma^*$ :

- ★ If  $s \in L$ , then  $M$  accepts  $s$ .

- ★ If  $s \notin L$ , then  $M$  rejects  $s$ .

If you are done, try Exercise 0 as well.

# Decidable Languages

Here are some more examples of decidable languages:

---

<sup>3</sup>Recall from CSC236 that regular language is a language that is decidable by a DFA.

# Decidable Languages

Here are some more examples of decidable languages:

- ★ Any regular language.<sup>3</sup>

---

<sup>3</sup>Recall from CSC236 that regular language is a language that is decidable by a DFA.

# Decidable Languages

Here are some more examples of decidable languages:

- ★ Any regular language.<sup>3</sup>
- ★  $L = \{w : \text{Someone in this room has a credit card with number } w\}$ .

---

<sup>3</sup>Recall from CSC236 that regular language is a language that is decidable by a DFA.

# Decidable Languages

Here are some more examples of decidable languages:

- ★ Any regular language.<sup>3</sup>
- ★  $L = \{w : \text{Someone in this room has a credit card with number } w\}$ .
- ★ In fact, any finite language  $L$  is decidable!

---

<sup>3</sup>Recall from CSC236 that regular language is a language that is decidable by a DFA.

# Decidable Languages

Here are some more examples of decidable languages:

- ★ Any regular language.<sup>3</sup>
- ★  $L = \{w : \text{Someone in this room has a credit card with number } w\}$ .
- ★ In fact, any finite language  $L$  is decidable!
- ★  $L = \{w : w \text{ is a grammatically correct English sentence}\}$ .

---

<sup>3</sup>Recall from CSC236 that regular language is a language that is decidable by a DFA.



# Decidable Languages

Here are some more examples of decidable languages:

- ★ Any regular language.<sup>3</sup>
- ★  $L = \{w : \text{Someone in this room has a credit card with number } w\}$ .
- ★ In fact, any finite language  $L$  is decidable!
- ★  $L = \{w : w \text{ is a grammatically correct English sentence}\}$ .
- ★  $L = \{w : w \text{ is a valid solution to the P vs NP problem}\}$ .

---

<sup>3</sup>Recall from CSC236 that regular language is a language that is decidable by a DFA.

# Decidable Languages

Here are some more examples of decidable languages:

- ★ Any regular language.<sup>3</sup>
- ★  $L = \{w : \text{Someone in this room has a credit card with number } w\}$ .
- ★ In fact, any finite language  $L$  is decidable!
- ★  $L = \{w : w \text{ is a grammatically correct English sentence}\}$ .
- ★  $L = \{w : w \text{ is a valid solution to the P vs NP problem}\}$ .

and (infinitely) many more!

---

<sup>3</sup>Recall from CSC236 that regular language is a language that is decidable by a DFA.

# Decidable Languages

Here are some more examples of decidable languages:

- ★ Any regular language.<sup>3</sup>
- ★  $L = \{w : \text{Someone in this room has a credit card with number } w\}$ .
- ★ In fact, any finite language  $L$  is decidable!
- ★  $L = \{w : w \text{ is a grammatically correct English sentence}\}$ .
- ★  $L = \{w : w \text{ is a valid solution to the P vs NP problem}\}$ .

and (infinitely) many more!

Are there any non-decidable languages?

---

<sup>3</sup>Recall from CSC236 that regular language is a language that is decidable by a DFA.

# Decidable Languages

Are there any non-decidable languages?

---

<sup>4</sup>Recall that a Diophantine equation is a multivariable equation like  $x^3 + 3xyz - w^{420} = 2$ .

# Decidable Languages

Are there any non-decidable languages? Yes.

$$L = \{p : p \text{ is a Diophantine equation that has a natural solution}\}$$

is not decidable!<sup>4</sup>

---

<sup>4</sup>Recall that a Diophantine equation is a multivariable equation like  $x^3 + 3xyz - w^{420} = 2$ .

# Decidable Languages

Are there any non-decidable languages? Yes.

$$L = \{p : p \text{ is a Diophantine equation that has a natural solution}\}$$

is not decidable!<sup>4</sup>

**Source:** trust me bro.<sup>5</sup>

---

<sup>4</sup>Recall that a Diophantine equation is a multivariable equation like  $x^3 + 3xyz - w^{420} = 2$ .

<sup>5</sup>You can look up “Hilbert’s tenth problem”.

# Recognizable Languages

$L = \{p : p \text{ is a Diophantine equation that has a natural solution}\}$   
is not decidable!<sup>6</sup>

---

<sup>6</sup>Recall that a Diophantine equation is a multivariable equation like

# Recognizable Languages

$L = \{p : p \text{ is a Diophantine equation that has a natural solution}\}$   
is not decidable!<sup>6</sup>  $L$  is **recognizable** though.

---

<sup>6</sup>Recall that a Diophantine equation is a multivariable equation like



# Recognizable Languages

$L = \{p : p \text{ is a Diophantine equation that has a natural solution}\}$

is not decidable!<sup>6</sup>  $L$  is **recognizable** though.

**Source:**

$M(p)$ :

```
if p isn't a valid Diophantine equation:
    reject
n = number of variables in p
s = 0
while True:
    for all x1, x2, ..., xn
    with x1 + x2 + ... + xn = s:
        if (x1, x2, ... xn) is a solution to p:
            accept
    s += 1
```

---

<sup>6</sup>Recall that a Diophantine equation is a multivariable equation like

# Recognizable Languages

For example, if  $p$  is the equation  $3x^5 - xy + y^2 = 3$ ,  $M(p)$  will:

# Recognizable Languages

For example, if  $p$  is the equation  $3x^5 - xy + y^2 = 3$ ,  $M(p)$  will:

- ★ Check if  $x = 0, y = 0$  is a solution. If not,
- ★ Check if  $x = 0, y = 1$  is a solution. If not,
- ★ Check if  $x = 1, y = 0$  is a solution. If not,
- ★ Check if  $x = 0, y = 2$  is a solution. If not,
- ★ Check if  $x = 1, y = 1$  is a solution. If not,
- ★ Check if  $x = 2, y = 0$  is a solution. If not,
- ★ Check if  $x = 0, y = 3$  is a solution. If not,
- ★ Check if  $x = 1, y = 2$  is a solution. If not,
- ★ Check if  $x = 2, y = 1$  is a solution. If not,
- ★ Check if  $x = 3, y = 0$  is a solution. If not,
- ★ Check if  $x = 0, y = 4$  is a solution. If not,
- ★ Check if  $x = 1, y = 3$  is a solution. If not,
- ★ Check if  $x = 2, y = 2$  is a solution. If not,
- ★ Check if  $x = 3, y = 1$  is a solution. If not,
- ★ Check if  $x = 4, y = 0$  is a solution. If not,
- ★ Check if  $x = 0, y = 5$  is a solution. If not,
- ★ Check if  $x = 1, y = 4$  is a solution. If not,

# Recognizable Languages

If  $3x^5 - xy + y^2 = 3$  has a natural solution  $(x, y) \in \mathbb{N}^2$ ,  $M(p)$  will

# Recognizable Languages

If  $3x^5 - xy + y^2 = 3$  has a natural solution  $(x, y) \in \mathbb{N}^2$ ,  $M(p)$  will eventually accept.

# Recognizable Languages

If  $3x^5 - xy + y^2 = 3$  has a natural solution  $(x, y) \in \mathbb{N}^2$ ,  $M(p)$  will eventually accept.

If  $3x^5 - xy + y^2 = 3$  has no natural solutions,  $M(p)$  will

# Recognizable Languages

If  $3x^5 - xy + y^2 = 3$  has a natural solution  $(x, y) \in \mathbb{N}^2$ ,  $M(p)$  will eventually accept.

If  $3x^5 - xy + y^2 = 3$  has no natural solutions,  $M(p)$  will run forever, i.e. **loop**.

# Recognizable Languages

Another recognizable but not decidable language:



# Recognizable Languages

Another recognizable but not decidable language:

$$\star L = \{M\#x : M \text{ is a TM that halts on } x\}.$$

# Recognizable Languages

Another recognizable but not decidable language:

- ★  $L = \{M\#x : M \text{ is a TM that halts on } x\}$ . This is known as the **Halting problem**.

# Recognizable Languages

Another recognizable but not decidable language:

- ★  $L = \{M\#x : M \text{ is a TM that halts on } x\}$ . This is known as the **Halting problem**.

Some non-recognizable languages:

# Recognizable Languages

Another recognizable but not decidable language:

- ★  $L = \{M\#x : M \text{ is a TM that halts on } x\}$ . This is known as the **Halting problem**.

Some non-recognizable languages:

- ★  $L = \{M\#x : M \text{ is a TM that loops on } x\}$ .

# Recognizable Languages

Another recognizable but not decidable language:

- ★  $L = \{M\#x : M \text{ is a TM that halts on } x\}$ . This is known as the **Halting problem**.

Some non-recognizable languages:

- ★  $L = \{M\#x : M \text{ is a TM that loops on } x\}$ .
- ★  $L = \{M : M \text{ is a TM that halts on all inputs}\}$ .

# Enumerators (possibly useful for assignment!)

# Enumerators (possibly useful for assignment!)

Let  $L$  be a language.

# Enumerators (possibly useful for assignment!)

Let  $L$  be a language. An **enumerator** for  $L$  is a program that accepts no input and:



# Enumerators (possibly useful for assignment!)

Let  $L$  be a language. An **enumerator** for  $L$  is a program that accepts no input and:

- ★ Eventually prints out  $w$  for every  $w \in L$  (possibly with duplicates).

# Enumerators (possibly useful for assignment!)

Let  $L$  be a language. An **enumerator** for  $L$  is a program that accepts no input and:

- ★ Eventually prints out  $w$  for every  $w \in L$  (possibly with duplicates).
- ★ Never prints out anything other than strings in  $L$ .

# Enumerators (possibly useful for assignment!)

Let  $L$  be a language. An **enumerator** for  $L$  is a program that accepts no input and:

- ★ Eventually prints out  $w$  for every  $w \in L$  (possibly with duplicates).
- ★ Never prints out anything other than strings in  $L$ .

Example: the following program is an enumerator for the language  $\{0^n 1^n : n \in \mathbb{N}\}$ .

```
def enum():  
    n = 0  
    while True:  
        print('0'*n + '1'*n)  
        n += 1
```

# Enumerators (possibly useful for assignment!)

Let  $L$  be a language. An **enumerator** for  $L$  is a program that accepts no input and:

- ★ Eventually prints out  $w$  for every  $w \in L$  (possibly with duplicates).
- ★ Never prints out anything other than strings in  $L$ .

Example: the following program is an enumerator for the language  $\{0^n 1^n : n \in \mathbb{N}\}$ .

```
def enum():  
    n = 0  
    while True:  
        print('0'*n + '1'*n)  
        n += 1
```

Try running `enum.py` on your computer!

# Enumerators (possibly useful for assignment!)

Here's something that might be useful. Remember to cite!

# Enumerators (possibly useful for assignment!)

Here's something that might be useful. Remember to cite!

## Theorem (Sipser 3.21)

*A language  $L$  is recognizable if and only if it has an enumerator.*

# Enumerators (possibly useful for assignment!)

Here's something that might be useful. Remember to cite!

## Theorem (Sipser 3.21)

*A language  $L$  is recognizable if and only if it has an enumerator.*

*Proof.* First, suppose  $L$  has an enumerator  $\text{enum}$ . Define  $M$  as follows:

$M(x)$ :

```
run enum in the background
while True:
    if enum has printed x:
        accept
```

# Enumerators (possibly useful for assignment!)

Here's something that might be useful. Remember to cite!

## Theorem (Sipser 3.21)

*A language  $L$  is recognizable if and only if it has an enumerator.*

*Proof.* First, suppose  $L$  has an enumerator `enum`. Define  $M$  as follows:

$M(x)$ :

```
run enum in the background
while True:
    if enum has printed x:
        accept
```

★ If  $x \in L$ , then  $x$  is eventually printed by the enumerator, and  $M(x)$  accepts.



# Enumerators (possibly useful for assignment!)

Here's something that might be useful. Remember to cite!

## Theorem (Sipser 3.21)

*A language  $L$  is recognizable if and only if it has an enumerator.*

*Proof.* First, suppose  $L$  has an enumerator `enum`. Define  $M$  as follows:

$M(x)$ :

```
run enum in the background
while True:
    if enum has printed x:
        accept
```

- ★ If  $x \in L$ , then  $x$  is eventually printed by the enumerator, and  $M(x)$  accepts.
- ★ If  $x \notin L$ , then  $x$  is never printed, and  $M(x)$  loops.

# Enumerators (possibly useful for assignment!)

Here's something that might be useful. Remember to cite!

## Theorem (Sipser 3.21)

*A language  $L$  is recognizable if and only if it has an enumerator.*

*Proof.* First, suppose  $L$  has an enumerator `enum`. Define  $M$  as follows:

$M(x)$ :

```
run enum in the background
```

```
while True:
```

```
    if enum has printed x:
```

```
        accept
```

- ★ If  $x \in L$ , then  $x$  is eventually printed by the enumerator, and  $M(x)$  accepts.
- ★ If  $x \notin L$ , then  $x$  is never printed, and  $M(x)$  loops.

Thus  $M$  recognizes  $L$ .

# Enumerators (possibly useful for assignment!)

Here's something that might be useful. Remember to cite!

# Enumerators (possibly useful for assignment!)

Here's something that might be useful. Remember to cite!

## **Theorem (Sipser 3.21)**

*A language  $L$  is recognizable if and only if it has an enumerator.*

# Enumerators (possibly useful for assignment!)

Here's something that might be useful. Remember to cite!

## Theorem (Sipser 3.21)

*A language  $L$  is recognizable if and only if it has an enumerator.*

*Proof.* Now, suppose  $L$  has a recognizer  $M$ . Define `enum` as follows:

# Enumerators (possibly useful for assignment!)

Here's something that might be useful. Remember to cite!

## Theorem (Sipser 3.21)

*A language  $L$  is recognizable if and only if it has an enumerator.*

*Proof.* Now, suppose  $L$  has a recognizer  $M$ . Define `enum` as follows:

```
enum():  
    n = 0  
    while True:  
        for all strings w of length <= n:  
            run M(w) for n steps  
            if M(w) accepts:  
                print(w)  
        n += 1
```

# Enumerators (possibly useful for assignment!)

Here's something that might be useful. Remember to cite!

## Theorem (Sipser 3.21)

*A language  $L$  is recognizable if and only if it has an enumerator.*

*Proof.* Now, suppose  $L$  has a recognizer  $M$ . Define `enum` as follows:

```
enum():  
    n = 0  
    while True:  
        for all strings  $w$  of length  $\leq n$ :  
            run  $M(w)$  for  $n$  steps  
            if  $M(w)$  accepts:  
                print( $w$ )  
        n += 1
```

## Task:

- ★ Given any  $w \notin L$ , Why does `enum()` never print out  $w$ ?
- ★ Given any  $w \in L$ , Why does `enum()` eventually print out  $w$ ?

# Useful trick

```
enum():  
    n = 0  
    while True:  
        for all strings w of length <= n:  
            run M(w) for n steps  
            if M(w) accepts:  
                print(w)  
        n += 1
```

This idea of “running for  $n$  steps, then increasing the maximum time allowed and trying again” is very useful in CSC363!