

hw3 scores released  
Practice assignments  
Project Part 2 released.

Week7

Monday May 9

	$L = L(M)$	$L = L(D)$	$L = L(E)$
	Suppose $M$ is a TM that recognizes $L$	Suppose $D$ is a TM that decides $L$	Suppose $E$ is an enumerator that enumerates $L$
If string $w$ is in $L$ then ...	$M$ accepts $w$	$D$ accepts $w$	$E$ prints $w$ (in finite time)
If string $w$ is not in $L$ then ...	$M$ rejects $w$ or $M$ loops on $w$	$D$ rejects $w$	$E$ never prints $w$

### Describing Turing machines (Sipser p. 185)

The Church-Turing thesis posits that each algorithm can be implemented by some Turing machine

High-level descriptions of Turing machine algorithms are written as indented text within quotation marks.

Stages of the algorithm are typically numbered consecutively.

The first line specifies the input to the machine, which must be a string. This string may be the encoding of some object or list of objects.

**Notation:**  $\langle O \rangle$  is the string that encodes the object  $O$ .  $\langle O_1, \dots, O_n \rangle$  is the string that encodes the list of objects  $O_1, \dots, O_n$ .

**Assumption:** There are Turing machines that can be called as subroutines to decode the string representations of common objects and interact with these objects as intended (data structures).

For example, since there are algorithms to answer each of the following questions, by Church-Turing thesis, there is a Turing machine that accepts exactly those strings for which the answer to the question is "yes"

#### Computational problem

- Does a string over  $\{0, 1\}$  have even length?
- Does a string over  $\{0, 1\}$  encode a string of ASCII characters?<sup>1</sup>
- Does a DFA have a specific number of states?  
7 bit
- Do two NFAs have any state names in common?
- Do two CFGs have the same start variable?

On input  $\langle N_1, N_2 \rangle \dots$

On input  $\langle A, n \rangle$   
the string representation  
of the formal definition  
of the DFA  $A$  and a reference  
1. Parse the string  
 $\langle A \rangle$  to find  
the substring  
listing the set  
of states of  $A$ .  
2. Count the num of  
distinct elements of this set  
3. If this number equals  
accept.  
Otherwise reject.

<sup>1</sup>An introduction to ASCII is available on the w3 tutorial here.

The TM answering the question  
Can be defined in different  
ways:

- Does a string over  $\{0, 1\}$  have even length?

\* High level definition

" On input  $w$   
1. If  $|w| \% 2 == 0$ , accept  
2. Otherwise, reject. "

\* Implementation-level definition

" On input  $w$   
1. Scan across the tape  
considering two cells at a time.  
2. If both cells blank,  
halt and accept.  
3. If both cells have  
characters from input  
alphabet, cross them  
out and move to  
next-to-the-right pair  
of cells.  
4. If one cell blank and  
the other has an  
input character, halt and  
reject. "

\* Formal definition  
(extra practice)

A **computational problem** is decidable iff language encoding its positive problem instances is decidable.

*strings encoding objects for which answer is yes*

The computational problem “Does a specific DFA accept a given string?” is encoded by the language

$$\begin{aligned} & \{\text{representations of DFAs } M \text{ and strings } w \text{ such that } w \in L(M)\} \\ &= \{\langle M, w \rangle \mid M \text{ is a DFA, } w \text{ is a string, } w \in L(M)\} \end{aligned}$$

The computational problem “Is the language generated by a CFG empty?” is encoded by the language

$$\begin{aligned} & \{\text{representations of CFGs } G \text{ such that } L(G) = \emptyset\} \\ &= \{\langle G \rangle \mid G \text{ is a CFG, } L(G) = \emptyset\} \end{aligned}$$

The computational problem “Is the given Turing machine a decider?” is encoded by the language

$$\begin{aligned} & \{\text{representations of TMs } M \text{ such that } M \text{ halts on every input}\} \\ &= \{\langle M \rangle \mid M \text{ is a TM and for each string } w, M \text{ halts on } w\} \end{aligned}$$

*Note: writing down the language encoding a computational problem is only the first step in determining if it's recognizable, decidable, or ...*

Some classes of computational problems help us understand the differences between the machine models we've been studying:

### Acceptance problem

...for DFA	$A_{DFA}$	$\{\langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w\}$
...for NFA	$A_{NFA}$	$\{\langle B, w \rangle \mid B \text{ is a NFA that accepts input string } w\}$
...for regular expressions	$A_{REX}$	$\{\langle R, w \rangle \mid R \text{ is a regular expression that generates input string } w\}$
...for CFG	$A_{CFG}$	$\{\langle G, w \rangle \mid G \text{ is a context-free grammar that generates input string } w\}$
...for PDA	$A_{PDA}$	$\{\langle B, w \rangle \mid B \text{ is a PDA that accepts input string } w\}$

### Language emptiness testing

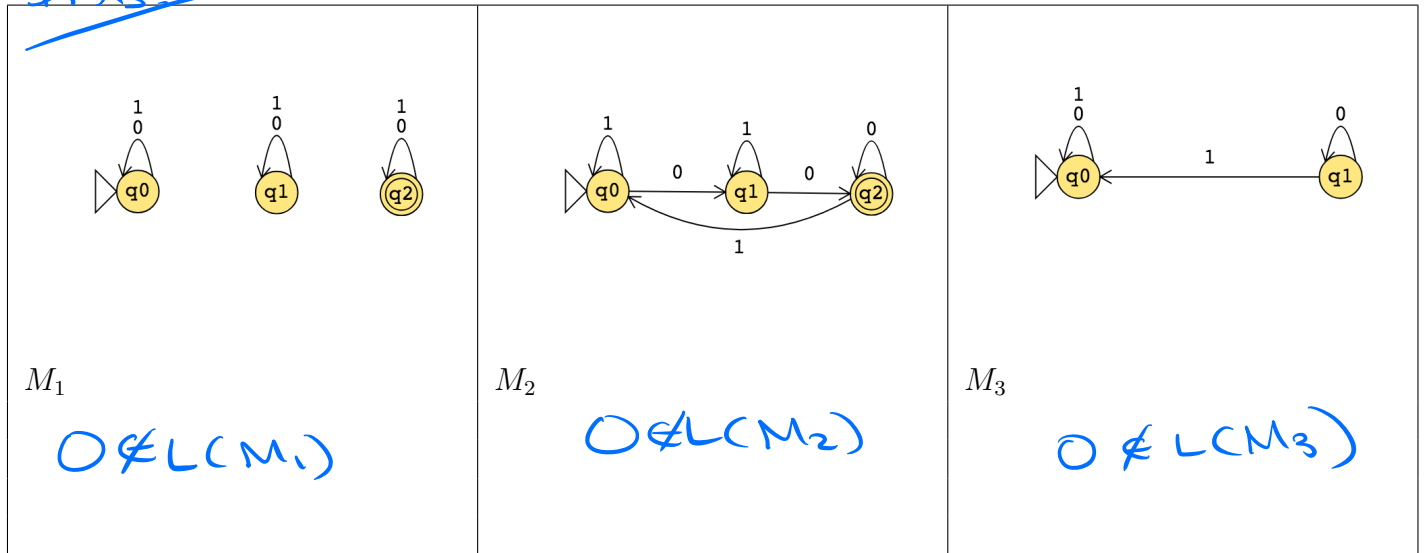
...for DFA	$E_{DFA}$	$\{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$
...for NFA	$E_{NFA}$	$\{\langle A \rangle \mid A \text{ is a NFA and } L(A) = \emptyset\}$
...for regular expressions	$E_{REX}$	$\{\langle R \rangle \mid R \text{ is a regular expression and } L(R) = \emptyset\}$
...for CFG	$E_{CFG}$	$\{\langle G \rangle \mid G \text{ is a context-free grammar and } L(G) = \emptyset\}$
...for PDA	$E_{PDA}$	$\{\langle A \rangle \mid A \text{ is a PDA and } L(A) = \emptyset\}$

### Language equality testing

...for DFA	$EQ_{DFA}$	$\{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$
...for NFA	$EQ_{NFA}$	$\{\langle A, B \rangle \mid A \text{ and } B \text{ are NFAs and } L(A) = L(B)\}$
...for regular expressions	$EQ_{REX}$	$\{\langle R, R' \rangle \mid R \text{ and } R' \text{ are regular expressions and } L(R) = L(R')\}$
...for CFG	$EQ_{CFG}$	$\{\langle G, G' \rangle \mid G \text{ and } G' \text{ are CFGs and } L(G) = L(G')\}$
...for PDA	$EQ_{PDA}$	$\{\langle A, B \rangle \mid A \text{ and } B \text{ are PDAs and } L(A) = L(B)\}$

Sipser Section 4.1

# DFA



Example strings in  $A_{DFA} = \{ \langle M, w \rangle \mid M \text{ is DFA, } w \text{ is string, } w \in L(M) \}$

specific strings in  $A_{DFA}$  depends on encodings.

$\langle M_1, 0 \rangle \notin A_{DFA}$

$\langle M_2, 00 \rangle \in A_{DFA}$

Example strings in  $E_{DFA} = \{ \langle M \rangle \mid M \text{ is DFA, } L(M) = \emptyset \}$

$\langle M_1 \rangle \in E_{DFA}$

$\langle M_3 \rangle \in E_{DFA}$

$\langle M_2 \rangle \notin E_{DFA}$

Example strings in  $EQ_{DFA} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ DFA, } L(M_1) = L(M_2) \}$

$\langle M_1, M_3 \rangle \in EQ_{DFA}$

$\langle M_1, M_1 \rangle \in EQ_{DFA}$

$\langle M_1, M_2 \rangle \notin EQ_{DFA}$

Food for thought: which of the following computational problems are decidable:  $A_{DFA}$ ?,  $E_{DFA}$ ?,  $EQ_{DFA}$ ?

## **Review: Week 7 Monday**

Recall: Review quizzes based on class material are assigned each day. These quizzes will help you track and confirm your understanding of the concepts and examples we work in class. Quizzes can be submitted on Gradescope as many times (with no penalty) as you like until the quiz deadline: the three quizzes each week are all due on Friday (with no penalty late submission open until Sunday).

Please complete the review quiz questions on Gradescope about computational problems.

**Pre class reading for next time:** Decidable problems concerning regular languages, Sipser pages 194-196.

Wednesday May 11

Solving a computational problem means defining a TM that is a decider and whose language is the language encoding this computational problem.

<b>Acceptance problem</b>
for ... $A_{...}$ $\{\langle B, w \rangle \mid B \text{ is a ... that accepts input string } w\}$
<b>Language emptiness testing</b>
for ... $E_{...}$ $\{\langle A \rangle \mid A \text{ is a ... and } L(A) = \emptyset\}$
<b>Language equality testing</b>
for ... $EQ_{...}$ $\{\langle A, B \rangle \mid A \text{ and } B \text{ are ... and } L(A) = L(B)\}$
Sipser Section 4.1

$M_1$  = "On input  $\langle M, w \rangle$ , where  $M$  is a DFA and  $w$  is a string:

0. Type check encoding to check input is correct type. *if correct type, continue to step 1 if not, reject.*
1. Simulate  $M$  on input  $w$  (by keeping track of states in  $M$ , transition function of  $M$ , etc.)
2. If the simulation ends in an accept state of  $M$ , accept. If it ends in a non-accept state of  $M$ , reject. "

finitely many steps  
compute DFA always finite

finitely many steps  
What is  $L(M_1)$ ?

$\langle \rightarrow \text{start}, 0100 \rangle \in L(M_1)$   
 $\langle \rightarrow \text{start} \rightarrow \text{start}, \epsilon \rangle \notin L(M_1)$   
 $\langle \rightarrow \text{start} \rightarrow \text{start} \rightarrow \text{start}, \epsilon \rangle \notin L(M_1)$

$$L(M_1) = \{ \langle A, w \rangle \mid \text{A DFA } w \in L(A) \} = A_{\text{DFA}}$$

Is  $L(M_1)$  a decider?

✓ Check whether  $M_1$  is guaranteed to halt for all input. Checking each step of high-level description, see computation of  $M_1$  halts in finite time for all inputs.

$M_2$  = "On input  $\langle M, w \rangle$  where  $M$  is a DFA and  $w$  is a string, IMPLICIT TYPE CHECK

1. Run  $M$  on input  $w$ .
2. If  $M$  accepts, accept; if  $M$  rejects, reject."

What is  $L(M_2)$ ?

$$L(M_2) = L(M_1) = A_{\text{DFA}}$$

Is  $L(M_2)$  a decider?

Yes.

$$A_{REG} = \{ \langle R, w \rangle \mid R \text{ is regular expression, } w \text{ a string, } w \in L(R) \}$$

$$A_{NFA} = \{ \langle M, w \rangle \mid M \text{ is NFA, } w \text{ a string, } w \in L(M) \}$$

~~True~~ False:  $A_{REG} = A_{NFA} = A_{DFA}$        $\langle (0|1)^*, 0 \rangle \in A_{REG}$   
 $\langle (0|1)^*, 0 \rangle \notin A_{NFA}$

True: ~~False~~:  $A_{REG} \cap A_{NFA} = \emptyset, A_{REG} \cap A_{DFA} = \emptyset, A_{DFA} \cap A_{NFA} = \emptyset$

A Turing machine that decides  $A_{NFA}$  is:

$M_{NFA} =$  "On input  $\langle M, w \rangle$  M NFA, w string

1. Use subset construction from Chapter 1 to transform M to a DFA  $M_0$  with  $L(M_0) = L(M)$ .

2. Run  $M_0$  on w.

i.e. Run  $M_1$  on input  $\langle M_0, w \rangle$

3. If accepts, accept; if rejects, reject".

To confirm  $L(M_{NFA}) = A_{NFA}$  and  $M_{NFA}$  is decider.

A Turing machine that decides  $A_{REG}$  is:

(extra practice)

$$E_{DFA} = \{ \langle M \rangle \mid M \text{ DFA}, L(M) = \emptyset \}.$$

$$\langle \emptyset^0 \rangle \in E_{DFA}$$

$M_3$  = "On input  $\langle M \rangle$  where  $M$  is a DFA,

1. For integer  $i = 1, 2, \dots$  *infinite loop.* *Keep coming!*
2. Let  $s_i$  be the  $i$ th string over the alphabet of  $M$  (ordered in string order).
3. Run  $M$  on input  $s_i$ .
4. If  $M$  accepts, reject. If  $M$  rejects, increment  $i$  and keep going."

$$L(M_3) = \emptyset$$

Choose the correct option to help fill in the blank so that  $M_3$  recognizes  $E_{DFA}$

*Not possible - this approach won't work for recognizing  $E_{DFA}$ .*

- accepts
- rejects
- loop for ever
- We can't fill in the blank in any way to make this work
- None of the above

$$E_{DFA} \neq \emptyset.$$

$M_4$  = "On input  $\langle M \rangle$  where  $M$  is a DFA,

1. Mark the start state of  $M$ .
2. Repeat until no new states get marked:
3. Loop over the states of  $M$ .
4. Mark any unmarked state that has an incoming edge from a marked state.
5. If no accept state of  $A$  is marked, accept; otherwise, reject."

*Search underlying graph of state diagram of  $M$  for path from the start state to an accept state.*

*transition*

$$L(M) = \emptyset \text{ so } \langle M \rangle \in E_{DFA}$$

$$L(M) \neq \emptyset \text{ so } \langle M \rangle \notin E_{DFA}$$

To build a Turing machine that decides  $EQ_{DFA}$ , notice that

$$L_1 = L_2 \quad \text{iff} \quad ((L_1 \cap \overline{L_2}) \cup (L_2 \cap \overline{L_1})) = \emptyset$$

There are no elements that are in one set and not the other

$M_{EQ_{DFA}}$  = "On input  $\langle M_1, M_2 \rangle$  where  $M_1, M_2$  are DFA

1. Construct (using flip states approach) DFAs  $\tilde{M}_1$  and  $\tilde{M}_2$  such that  $L(\tilde{M}_1) = \overline{L(M_1)}$ ,  $L(\tilde{M}_2) = \overline{L(M_2)}$
2. Construct (using Cartesian product construction) DFAs  $D_1$  and  $D_2$  such that  $L(D_1) = L(M_1) \cap L(\tilde{M}_2)$  and  $L(D_2) = L(\tilde{M}_1) \cap L(M_2)$
3. Construct (using Cartesian product construction) DFA  $D$  such that  $L(D) = L(D_1) \cup L(D_2)$
4. Run  $M_4$  on input  $\langle D \rangle$
5. If accepts, accept; if rejects, reject"

*to confirm:  $L(M_{EQ_{DFA}}) = EQ_{DFA}$  and  $M_{EQ_{DFA}}$  is decider.*



**Summary:** We can use the decision procedures (Turing machines) of decidable problems as subroutines in other algorithms. For example, we have subroutines for deciding each of  $A_{DFA}$ ,  $E_{DFA}$ ,  $EQ_{DFA}$ . We can also use algorithms for known constructions as subroutines in other algorithms. For example, we have **subroutines for**: counting the number of states in a state diagram, counting the number of characters in an alphabet, converting DFA to a DFA recognizing the complement of the original language or a DFA recognizing the Kleene star of the original language, constructing a DFA or NFA from two DFA or NFA so that we have a machine recognizing the language of the union (or intersection, concatenation) of the languages of the original machines; converting regular expressions to equivalent DFA; converting DFA to equivalent regular expressions, etc.

## **Review: Week 7 Wednesday**

Please complete the review quiz questions on Gradescope about decidable computational problems.

**Pre class reading for next time:** An undecidable language, Sipser pages 207-209.

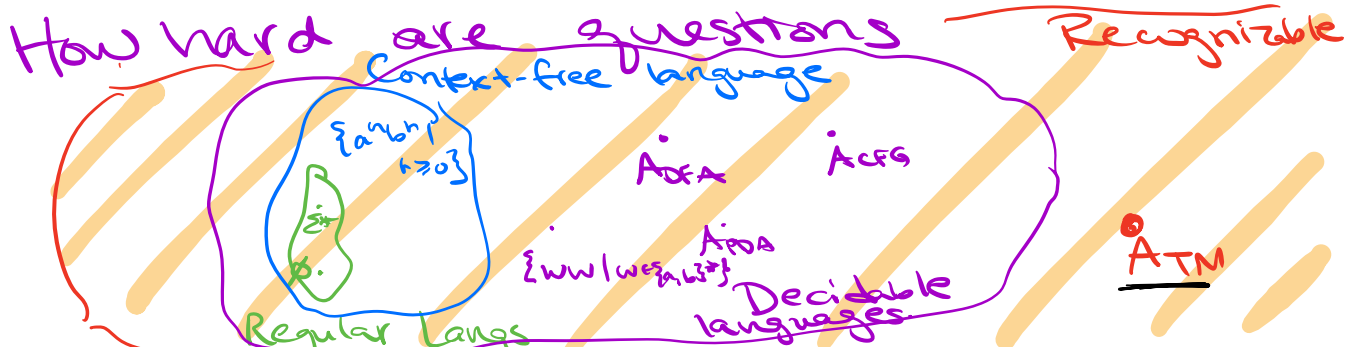
Friday May 13

Some classes of computational problems help us understand the differences between the machine models we've been studying:

Acceptance problem			
...for DFA	$A_{DFA}$	$\{\langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w\}$	Decidable
...for NFA	$A_{NFA}$	$\{\langle B, w \rangle \mid B \text{ is a NFA that accepts input string } w\}$	Decidable
...for regular expressions	$A_{REX}$	$\{\langle R, w \rangle \mid R \text{ is a regular expression that generates input string } w\}$	Decidable
...for CFG	$A_{CFG}$	$\{\langle G, w \rangle \mid G \text{ is a context-free grammar that generates input string } w\}$	Decidable
...for PDA	$A_{PDA}$	$\{\langle B, w \rangle \mid B \text{ is a PDA that accepts input string } w\}$	Decidable
Language emptiness testing			
...for DFA	$E_{DFA}$	$\{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$	
...for NFA	$E_{NFA}$	$\{\langle A \rangle \mid A \text{ is a NFA and } L(A) = \emptyset\}$	
...for regular expressions	$E_{REX}$	$\{\langle R \rangle \mid R \text{ is a regular expression and } L(R) = \emptyset\}$	
...for CFG	$E_{CFG}$	$\{\langle G \rangle \mid G \text{ is a context-free grammar and } L(G) = \emptyset\}$	
...for PDA	$E_{PDA}$	$\{\langle A \rangle \mid A \text{ is a PDA and } L(A) = \emptyset\}$	
Language equality testing			
...for DFA	$EQ_{DFA}$	$\{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$	
...for NFA	$EQ_{NFA}$	$\{\langle A, B \rangle \mid A \text{ and } B \text{ are NFAs and } L(A) = L(B)\}$	
...for regular expressions	$EQ_{REX}$	$\{\langle R, R' \rangle \mid R \text{ and } R' \text{ are regular expressions and } L(R) = L(R')\}$	
...for CFG	$EQ_{CFG}$	$\{\langle G, G' \rangle \mid G \text{ and } G' \text{ are CFGs and } L(G) = L(G')\}$	
...for PDA	$EQ_{PDA}$	$\{\langle A, B \rangle \mid A \text{ and } B \text{ are PDAs and } L(A) = L(B)\}$	
Sipser Section 4.1			

using  
M. from  
Wednesday  
Decidable  
Ch 4  
Acfg.

We could use an algorithm deciding  $A_{PDA}$  as a subroutine for an algorithm deciding  $A_{CFG}$  and vice versa (relying on Chapter 2 result that there's a transformation between CFGs and PDAs). \*



## Acceptance problem

for Turing machines  $A_{TM} = \{\langle M, w \rangle \mid M \text{ is a Turing machine that accepts input string } w\}$

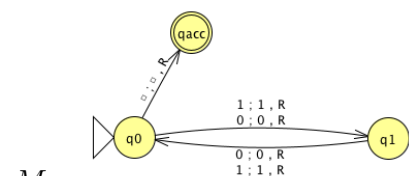
## Language emptiness testing

for Turing machines  $E_{TM} = \{\langle M \rangle \mid M \text{ is a Turing machine and } L(M) = \emptyset\}$

## Language equality testing

for Turing machines  $EQ_{TM} = \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are Turing machines and } L(M_1) = L(M_2)\}$

Sipser Section 4.1



$M_1$

$$L(M_1) = \{w \in \{0,1\}^* \mid |w| \text{ is even}\}$$

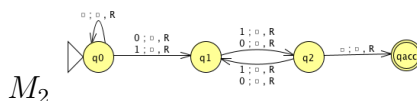
$$\varepsilon \in L(M_1)$$

Example strings in  $A_{TM}$

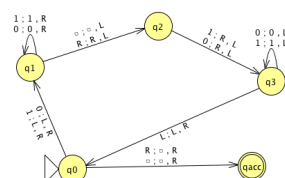
$$\langle M_1, \varepsilon \rangle \in A_{TM}$$

$$\langle M_1, 111111 \rangle \notin A_{TM}$$

$$\langle M_1 \rangle \notin A_{TM}$$



$M_2$



$M_3$

Example strings in  $E_{TM}$

Example strings in  $EQ_{TM}$

**Theorem:**  $A_{TM}$  is Turing-recognizable.

**Strategy:** To prove this theorem, we need to define a Turing machine  $R_{ATM}$  such that  $L(R_{ATM}) = A_{TM}$ .

Define  $R_{ATM} =$  " On input  $\langle M, w \rangle$   $M$  TM,  $w$  string.  
implicit type check: reject strings not well-typed.

1. Run  $M$  on  $w$   
start at the initial configuration, apply transition function, check at each step if compn enters  $q_{accept}$  or  $q_{reject}$
2. If computation of  $M$  on  $w$  accepts, accept.
3. If computation of  $M$  on  $w$  rejects, reject"

Proof of correctness:

WTS ① For each string in  $A_{TM}$ ,  $R_{ATM}$  accepts this string.

Let  $x \in A_{TM}$ . WTS  $R_{ATM}$  accepts  $x$ . Trace comp'n of  $R_{ATM}$  on  $x$ : By def of  $A_{TM}$ ,  $x = \langle M, w \rangle$  for some  $M$  TM,  $w$  string so type check passes.

In step 1 of  $R_{ATM}$ , simulate  $M$  on  $w$ .

By def of  $A_{TM}$ , since  $x \in A_{TM}$  and  $x = \langle M, w \rangle$  have  $w \in L(M)$ , namely computation of  $M$  on  $w$  halts and accepts. So  $R_{ATM}$  will move to step 2 in finite time and accept ✓

WTS ② For each string not in  $A_{TM}$ ,  $R_{ATM}$  rejects this string OR loops on this string.

Let  $x \notin A_{TM}$ . Either  $x \neq \langle M, w \rangle$  for any TM  $M$ ,  $w$  string or  $x = \langle M, w \rangle$  for  $M$  TM,  $w$  string and  $w \notin L(M)$ .

Trace  $R_{ATM}$  in either case to show  $R_{ATM}$  does not accept  $x$ .

(keep going for extra practice...)

We will show that  $A_{TM}$  is undecidable.

(Monday)

A **Turing-recognizable** language is a set of strings that is the language recognized by some Turing machine. We also say that such languages are recognizable.

A **Turing-decidable** language is a set of strings that is the language recognized by some decider. We also say that such languages are decidable.

An **unrecognizable** language is a language that is not Turing-recognizable.

An **undecidable** language is a language that is not Turing-decidable.

~~True~~ or **False**: Any undecidable language is also unrecognizable.

counterexample: ATM.

**True** or ~~False~~: Any unrecognizable language is also undecidable.

To prove that a computational problem is **decidable**, we find/ build a Turing machine that recognizes the language encoding the computational problem, and that is a decider.

How do we prove a specific problem is **not decidable**?

How would we even find such a computational problem?

*Counting arguments for the existence of an undecidable language:*

- The set of all Turing machines is countably infinite.
- Each Turing-recognizable language is associated with a Turing machine in a one-to-one relationship, so there can be no more Turing-recognizable languages than there are Turing machines.
- Since there are infinitely many Turing-recognizable languages (think of the singleton sets), there are countably infinitely many Turing-recognizable languages.
- Such the set of Turing-decidable languages is an infinite subset of the set of Turing-recognizable languages, the set of Turing-decidable languages is also countably infinite.

Since there are uncountably many languages (because  $\mathcal{P}(\Sigma^*)$  is uncountable), there are uncountably many unrecognizable languages and there are uncountably many undecidable languages.

What's a specific example of a language that is unrecognizable or undecidable?

Key idea: self-referential disagreement.

## Review: Week 7 Friday

Please complete the review quiz questions on Gradescope about undecidability and unrecognizability.

Some classes of computational problems help us understand the differences between the machine models we've been studying:

### Acceptance problem

... for DFA	$A_{DFA}$	$\{\langle B, w \rangle \mid B \text{ is a DFA that accepts input string } w\}$
... for NFA	$A_{NFA}$	$\{\langle B, w \rangle \mid B \text{ is a NFA that accepts input string } w\}$
... for regular expressions	$A_{REX}$	$\{\langle R, w \rangle \mid R \text{ is a regular expression that generates input string } w\}$
... for CFG	$A_{CFG}$	$\{\langle G, w \rangle \mid G \text{ is a context-free grammar that generates input string } w\}$
... for PDA	$A_{PDA}$	$\{\langle B, w \rangle \mid B \text{ is a PDA that accepts input string } w\}$

### Language emptiness testing

... for DFA	$E_{DFA}$	$\{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$
... for NFA	$E_{NFA}$	$\{\langle A \rangle \mid A \text{ is a NFA and } L(A) = \emptyset\}$
... for regular expressions	$E_{REX}$	$\{\langle R \rangle \mid R \text{ is a regular expression and } L(R) = \emptyset\}$
... for CFG	$E_{CFG}$	$\{\langle G \rangle \mid G \text{ is a context-free grammar and } L(G) = \emptyset\}$
... for PDA	$E_{PDA}$	$\{\langle A \rangle \mid A \text{ is a PDA and } L(A) = \emptyset\}$

### Language equality testing

... for DFA	$EQ_{DFA}$	$\{\langle A, B \rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$
... for NFA	$EQ_{NFA}$	$\{\langle A, B \rangle \mid A \text{ and } B \text{ are NFAs and } L(A) = L(B)\}$
... for regular expressions	$EQ_{REX}$	$\{\langle R, R' \rangle \mid R \text{ and } R' \text{ are regular expressions and } L(R) = L(R')\}$
... for CFG	$EQ_{CFG}$	$\{\langle G, G' \rangle \mid G \text{ and } G' \text{ are CFGs and } L(G) = L(G')\}$
... for PDA	$EQ_{PDA}$	$\{\langle A, B \rangle \mid A \text{ and } B \text{ are PDAs and } L(A) = L(B)\}$

Sipser Section 4.1