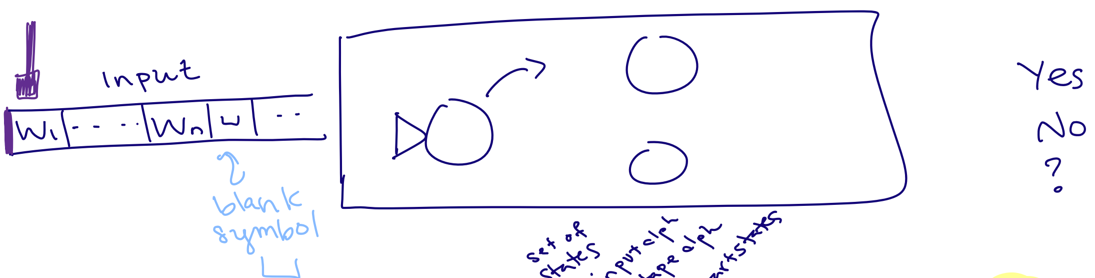
Monday: Turing machines (Ch 3)

We are ready to introduce a formal model that will capture a notion of general purpose computation.

- *Similar to DFA, NFA, PDA*: input will be an arbitrary string over a fixed alphabet.
- *Different from NFA, PDA*: machine is deterministic.
- *Different from DFA, NFA, PDA*: read-write head can move both to the left and to the right, and can extend to the right past the original input.
- *Similar to DFA, NFA, PDA*: transition function drives computation one step at a time by moving within a finite set of states, always starting at designated start state.
- *Different from DFA, NFA, PDA*: the special states for rejecting and accepting take effect immediately.

q_{rej} q_{acc}

(See more details: Sipser p. 166)



set of states, input alph, tape alph, start state

Formally: a Turing machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where δ is the **transition function**

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

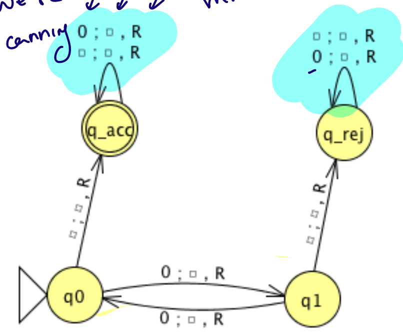
The **computation** of M on a string w over Σ is:

- Read/write head starts at leftmost position on tape.
- Input string is written on $|w|$-many leftmost cells of tape, rest of the tape cells have the blank symbol. **Tape alphabet** is Γ with $\llcorner\!\lrcorner \in \Gamma$ and $\Sigma \subseteq \Gamma$. The blank symbol $\llcorner\!\lrcorner \notin \Sigma$.
- Given current state of machine and current symbol being read at the tape head, the machine transitions to next state, writes a symbol to the current position of the tape head (overwriting existing symbol), and moves the tape head L or R (if possible). if at leftmost cell, L means stay put.
- Computation ends **if and when** machine enters either the accept or the reject state. This is called **halting**. Note: $q_{accept} \neq q_{reject}$.

The **language recognized by the Turing machine** M, is $L(M) = \{w \in \Sigma^* \mid w \text{ is accepted by } M\}$, which is defined as

$$\{w \in \Sigma^* \mid \text{computation of } M \text{ on } w \text{ halts after entering the accept state}\}$$

input char we're scanning | char to write | direction to move read/write head.



□ ␣ "blank"

Formal definition:

$(\{q0, q1, q\text{-}acc, q\text{-}rej\}, \{0\}, \{0, \sqcup\}, \delta, q0, q\text{-}acc, q\text{-}rej)$

Sample computation:

state ← read/write head

Tape

$q0 \downarrow$						
0	0	0	␣	␣	␣	␣
	$q1 \downarrow$					
␣	0	0	␣	␣	␣	␣
		$q0 \downarrow$				
␣	␣	0	␣	␣	␣	␣
			$q1 \downarrow$			
␣	␣	␣	␣	␣	␣	␣
				$qrej \downarrow$		
␣	␣	␣	␣	␣	␣	␣

trace

Transition function

δ	0	␣
q_0	$(q1, \sqcup, R)$	(, ,)
$q1$	(, ,)	(, ,)
q_acc	( )	( )
q-rej	( )	( )

Computation on input 000 rejects.

The language recognized by this machine is . . . $\{0^{2i} \mid i \geq 0\}$

$L((00)^*)$.

What about computation on $00\sqcup000$?

Not valid input to TM because not a string over $\{0\}$.

What about computation on ε?

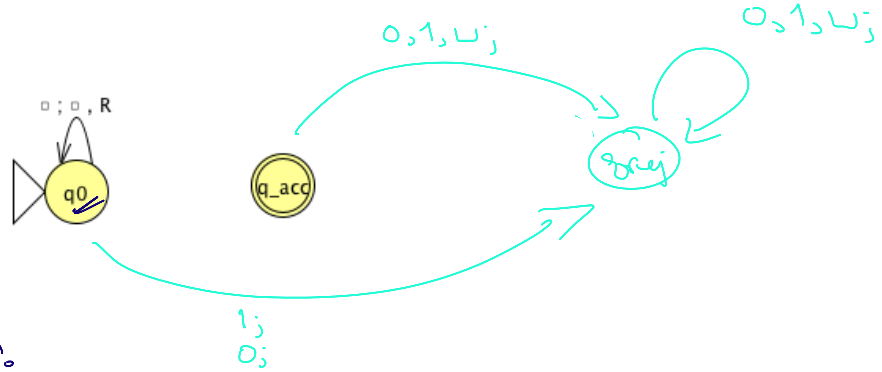
Tape start all blanks.

Describing Turing machines (Sipser p. 185) To define a Turing machine, we could give a

- **Formal definition**: the 7-tuple of parameters including set of states, input alphabet, tape alphabet, transition function, start state, accept state, and reject state; or,
- **Implementation-level definition**: English prose that describes the Turing machine head movements relative to contents of tape, and conditions for accepting / rejecting based on those contents.
- **High-level description**: description of algorithm (precise sequence of instructions), without implementation details of machine. As part of this description, can "call" and run another TM as a subroutine.

CONVENTION: OMIT REJECT STATE IN DIAGRAM *so long as it's not the start state.
CONSIDER ALL MISSING ARROWS AS POINTING TO IT

Fix $\Sigma = \{0,1\}$, $\Gamma = \{0,1,\llcorner\!\lrcorner\}$ for the Turing machines with the following state diagrams:



Example of string accepted: None!

Example of string rejected: 0, 1, 00

Example of string that is neither accepted nor rejected: ε. Turing machine loops on this string.

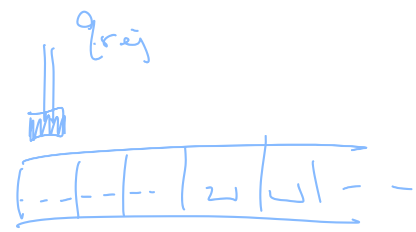
Implementation-level description

1. While scanning a blank symbol, move R.
2. When scan 0 or 1, reject.

High-level description

On input x

1. If $x = \varepsilon$, go to 1.
2. Otherwise, reject.



Example of string accepted: None!

Example of string rejected: Each string over $\{0,1\}$

Example of string on which TM loops: None!

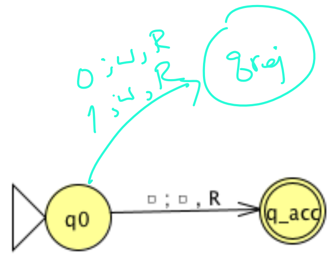
Implementation-level description

1. Reject immediately

High-level description

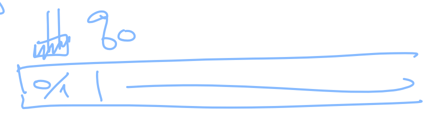
On input x

1. Reject.



On input ε: initial config'n of TM

On input $w \in \{0,1\}^*$ that is not empty: initial configuration of TM



Example of string accepted: ε
Example of string rejected: 0, 1, 001
Example of string on which TM loops: None.

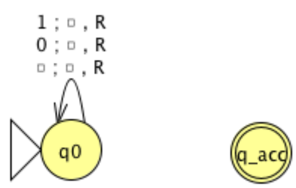
Implementation-level description

1. If first tape symbol is blank, move R and accept
2. Otherwise, reject.

High-level description

On input x
1. If x = ε, accept
2. Otherwise, reject.

On input 1:



Example of string accepted: None
Example of string rejected: None
Example of string on which TM loops: 1, ε, 0, 00, 01

Implementation-level description

1. Scan tape left-to-right, erasing all cells.

High-level description

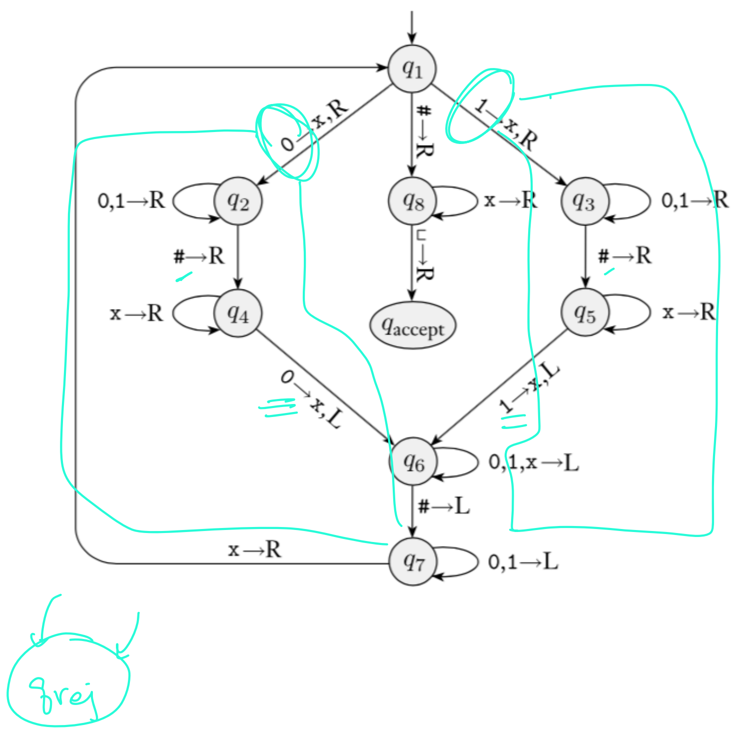
On input x
1. Goto 1.

Wednesday: Describing Turing machines and algorithms

Sipser Figure 3.10

Conventions in state diagram of TM: $b \to R$ label means $b \to b, R$ and all arrows missing from diagram represent transitions with output $(q_{reject}, \llcorner\!\lrcorner, R)$

$\Sigma = \{0, 1, \#\}$ $\Gamma = \{0, 1, \#, \llcorner\!\lrcorner, X\}$



Computation on input string 01#01

$q_1 \downarrow$							initial configuration.
0	1	#	0	1	␣	␣	
	$q_2 \downarrow$						after 1 step.
X	1	#	0	1	␣	␣	
		$q_2 \downarrow$					after 2 steps.
X	1	#	0	1	␣	␣	
			$q_4 \downarrow$				
X	1	#	0	1	␣	␣	
		$q_6 \downarrow$					
X	1	#	X	1	␣	␣	
	$q_7 \downarrow$						
X	1	#	X	1	␣	␣	
$\downarrow q_7$							
X	1	#	X	1	␣	␣	
	$\downarrow q_1$						
X	1	#	X	1	␣	␣	
		$\downarrow q_3$					
X	X	#	X	1	␣	␣	
			$\downarrow q_5$				
X	X	#	X	1	␣	␣	
				$\downarrow q_5$			
X	X	#	X	1	␣	␣	
			$\downarrow q_6$				
X	X	#	X	X	␣	␣	
		$\downarrow q_6$					
X	X	#	X	X	␣	␣	
	$\downarrow q_7$						
X	X	#	X	X	␣	␣	
		$\downarrow q_1$					
X	X	#	X	X	␣	␣	
			$\downarrow q_8$				
X	X	#	X	X	␣	␣	
				$\downarrow q_8$			
X	X	#	X	X	␣	␣	
					$\downarrow q_8$		
X	X	#	X	X	␣	␣	
						$\downarrow q_{accept}$!	
X	X	#	X	X	␣	␣	

Implementation level description of this machine:

q2+q4 Zig-zag across tape to corresponding positions on either side of # to check whether the characters in these positions agree. If they do not, or if there is no #, reject. If they do, cross them off.

q3+q5

Once all symbols to the left of the # are crossed off, check for any un-crossed-off symbols to the right of #; if there are any, reject; if there aren't, accept.

The language recognized by this machine is

$$\{w\#w \mid w \in \{0,1\}^*\}$$

High-level description of this machine is

Recall: 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.

Extra practice

Computation on input string 01#1

$q_1 \downarrow$						
0	1	#	1	␣	␣	␣

"Yes"

A language L is **recognized by** a Turing machine M means

each string in L is accepted by M and

each string not in L is not accepted by M.

(strings not accepted by M may be rejected by M or M loops on them)

A Turing machine M **recognizes** a language L means that L is recognized M

and we write

$L = L(M)$ ← set of strings that are each accepted by M.

↖ set

A Turing machine M is a **decider** means that for each string over the input alphabet of M, the computation of M on that string halts in finite time (i.e. enters q_{accept} or q_{reject} after finitely many applications of transition function).

A language L is **decided by** a Turing machine M means

"Yes"

each string in L is accepted by M and

each string not in L is rejected by M.

"No"

A Turing machine M **decides** a language L means M is a decider

and $L(M) = L$

Fix $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \llcorner\!\lrcorner\}$ for the Turing machines with the following state diagrams:

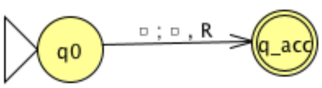
M_1

$L(M_1) = \emptyset$ Decider? Yes / No

M_2

$L(M_2) = \emptyset$ Decider? Yes / No

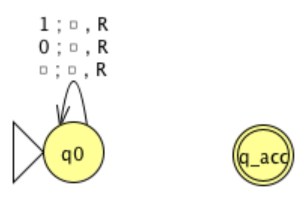
M_3



Decider? Yes / No

$L(M_3) = \{\varepsilon\} \neq \emptyset$.

M_4



Decider? Yes / No

$L(M_4) = \emptyset$

algorithm answers "Yes"

Friday: Decidable and Recognizable Languages

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.

algorithm answers "Yes" and "No"

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 decidable language is also recognizable.

Pf: Let L be an arbitrary decidable language. That is, there is a TM, M, that is a decider and such that L(M) = L. Since M is a TM, it witnesses that L is recognizable

True or **False**: Any recognizable language is also decidable.

Stay tuned for Counterexample: recognizable language that is not decidable

Wed: TMs that are not deciders exist.

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

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

Classes of languages.

collection of finite languages $\subseteq$ collection of regular languages $\subseteq$ collection of context-free languages $\subseteq$ collection of decidable languages $\subseteq$ collection of recognizable languages

Ch2.

Closure properties

Which classes of languages are closed under complementation?

Consider over $\Sigma = \{a,b\}$
$\overline{\{a\}} = \{w \mid w \text{ is a string and } w \neq a\}$

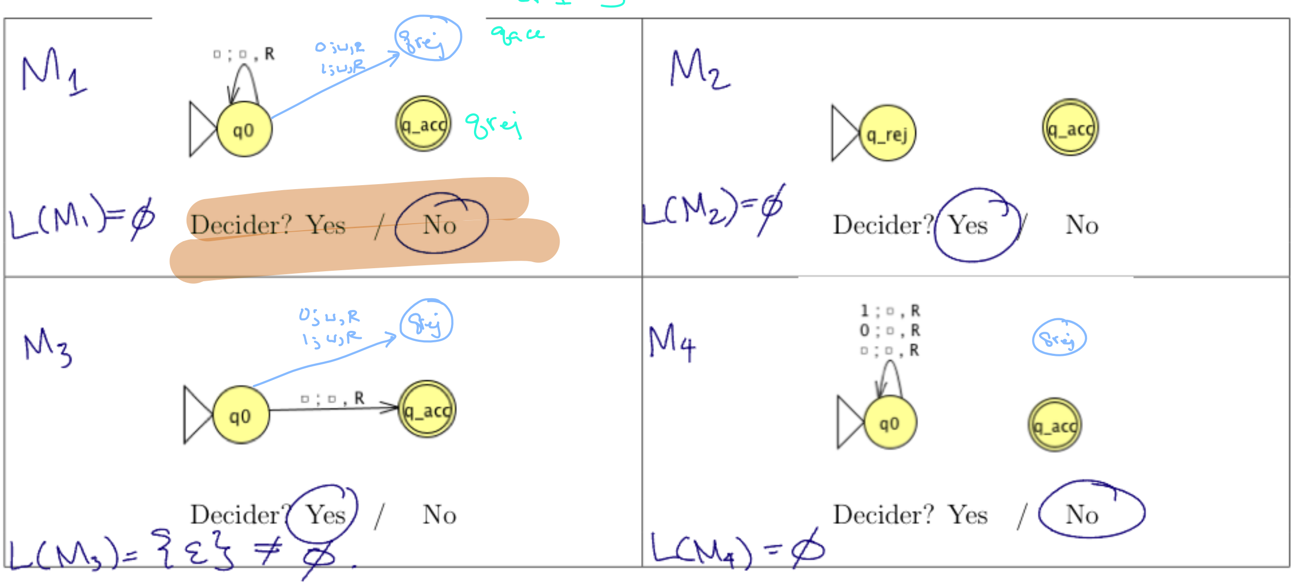
- class of finite languages? No!
- Class of regular languages? Yes (Pf using $Q \setminus F$)
- class of context-free languages? No (Pf in Ch2)
- class of decidable languages? Yes (Pf soon)
- class of recognizable languages? No (Pf later)

Pf that the class of decidable languages is closed under complementation:

Given L, a decidable language we WTS $\overline{L} = \Sigma^* \setminus L$ is decidable.
Have TM M_L that is a decider and $L(M_L) = L$. $M_L = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$
Want is to build a decider that recognizes $\overline{L}$.
Define $M_{new} = (Q, \Sigma, \Gamma, \delta, q_0, q_{rej}, q_{acc})$. Notice: for each $w \in \Sigma^*$ computations of M_L and M_{new} are identical because the machines have same transition functions, and guaranteed to land in either q_{rej} or q_{acc}. When it does so if we land in q_{rej}, M_L rejects but M_{new} accepts, and when computation lands in q_{acc}, M_L accepts but M_{new} rejects.
So $w \in L(M_L) = L$ iff $w \notin L(M_{new})$ so $L(M_{new}) = \overline{L}$ $\square$

Fix $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$ for the Turing machines with the following state diagrams:

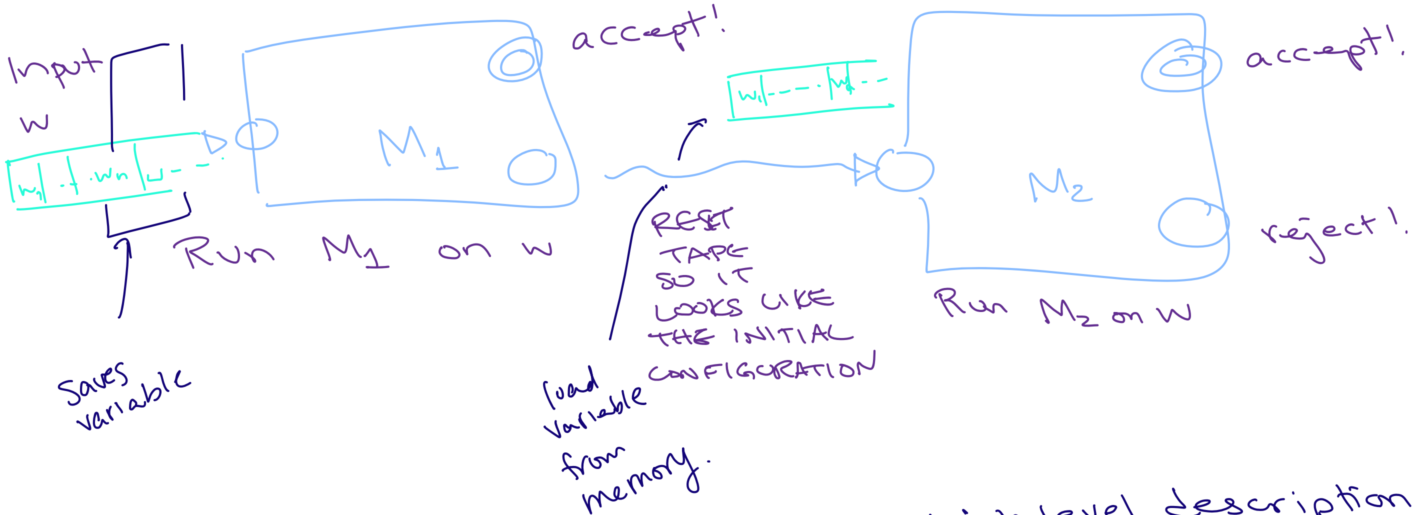
$L(M_{1\,new}) = \{w \mid w \text{ starts w/ } 0 \text{ or } 1\} \neq \overline{\emptyset}$ $\qquad$ $L(M_{2\,new}) = \Sigma^* = \overline{\emptyset}$



$L(M_{3\,new}) = \{w \in \Sigma^* \mid w \neq \varepsilon\} = \overline{\{\varepsilon\}}$ $\qquad$ $L(M_{4\,new}) = \emptyset \neq \overline{\emptyset}$

Claim: If two languages (over a fixed alphabet Σ) are Turing-decidable, then their union is as well.

Proof: Given L_1, L_2 decidable languages, have M_1, M_2 deciders with $L(M_1) = L_1$, $L(M_2) = L_2$.



high level description

Church-Turing Thesis (Sipser p. 183): The informal notion of algorithm is formalized completely and correctly by the formal definition of a Turing machine. In other words: all reasonably expressive models of computation are equally expressive with the standard Turing machine.

Pf: Let L_1, L_2 be decidable languages, with deciders M_1, M_2 recognizing L_1, L_2 respectively.

Define new Turing machine M

M = " On input w

1. Run M_1 on w
2. If accepts, accept.
3. If rejects, go to step 4.
4. Run M_2 on w
5. If accepts, accept.
6. If rejects, reject."

Pf of correctness

Claim: $L(M) = L(M_1) \cup L(M_2)$

WTS ① If a string is accepted by M then it's accepted by at least one of M_1, M_2. ...

WTS ② If a string is accepted by M then it's accepted by M

AND

if a string is accepted by M then it's accepted by M ...

Claim: If two languages (over a fixed alphabet Σ) are Turing-recognizable, then their union is as well.

Proof: Let L_1, L_2 be recognizable languages, with TMs M_1, M_2 recognizing L_1, L_2 respectively.

Define DOVETAILING

$M =$ " On input w
1. For $i = 0, 1, 2, \ldots$
2. Run M_1 on w for (at most) i steps.
2a. If it accepts, accept.
2b. If it rejects or doesn't halt within the i steps, go to step 3.
3 Run M_2 on w for (at most) i steps
3a. If it accepts, accept.
3b. If it rejects or doesn't halt within the i steps, increment i and go back to step 2. "

Claim: $L(M) = L(M_1) \cup L(M_2)$

WTS 1 If a string is accepted by M then it's accepted by at least one of M_1, M_2.

WTS 2 If a string is accepted by M_1 then it's accepted by M

AND

if a string is accepted by M_2 then it's accepted by M.

Keep for later...

Definition: A language L over an alphabet Σ is called **co-recognizable** if its complement, defined as $\Sigma^* \setminus L = \{x \in \Sigma^* \mid x \notin L\}$, is Turing-recognizable.

Theorem (Sipser Theorem 4.22): A language is Turing-decidable if and only if both it and its complement are Turing-recognizable.

Proof, first direction: Suppose language L is Turing-decidable. WTS that both it and its complement are Turing-recognizable.

Proof, second direction: Suppose language L is Turing-recognizable, and so is its complement. WTS that L is Turing-decidable.

Notation: The complement of a set X is denoted with a superscript c, X^c, or an overline, $\overline{X}$.

Week 6 at a glance

Textbook reading: Chapter 3, Section 4.1

For Monday: Page 165-166 Introduction to Section 3.1

For Wednesday: Example 3.9 on page 173

For Friday: Page 184-185 Terminology for describing Turing machines

Make sure you can:

- Use and design automata both formally and informally, including DFA, NFA, PDA, TM.
  - Use precise notation to formally define the state diagram of DFA, NFA, PDA, TM.
  - Use clear English to describe computations of DFA, NFA, PDA, TM informally
  - Determine whether a language is recognizable by a (D or N) FA and/or a PDA
  - Motivate the definition of a Turing machine
  - Trace the computation of a Turing machine on given input
  - Describe the language recognized by a Turing machine
  - Determine if a Turing machine is a decider
  - Given an implementation-level description of a Turing machine
  - Use high-level descriptions to define and trace Turing machines
  - Apply dovetailing in high-level definitions of machines
  - State and use the Church-Turing thesis
- Classify the computational complexity of a set of strings by determining whether it is regular, context-free, decidable, or recognizable.
- Give examples of sets that are regular, context-free, decidable, or recognizable.

TODO:

Review quizzes based on class material each day.

Homework assignment 3 due this Thursday.

Project due next Thursday.