Week 6 at a glance

Textbook reading: Chapter 3

Before Monday, Page 165-166 Introduction to Section 3.1.
Before Wednesday, Example 3.9 on page 173.
Before Friday, Page 184-185 Terminology for describing Turing machines.
For Week 7 ~~Monday~~: Introduction to Chapter 4.

We will be learning and practicing to:

- Clearly and unambiguously communicate computational ideas using appropriate formalism. Translate across levels of abstraction.
  - Use precise notation to formally define the state diagram of a Turing machine
  - Use clear English to describe computations of Turing machines informally.
    - **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**
  - Give examples of sets that are recognizable and decidable (and prove that they are).
    - **State the definition of the class of recognizable languages**
    - **State the definition of the class of decidable languages**
    - **State the definition of the class of co-recognizable languages**
- Know, select and apply appropriate computing knowledge and problem-solving techniques. Reason about computation and systems.
- Describe and prove closure properties of classes of languages under certain operations.
  - **Apply a general construction to create a new Turing machine from an example one.**
  - **Formalize a general construction from an informal description of it.**
  - **Use general constructions to prove closure properties of the class of decidable langugages.**
  - **Use general constructions to prove closure properties of the class of recognizable langugages.**

TODO:

This week: Test 1 Attempt 1 in CBTF.

Review Quiz 5 on PrairieLearn (http://us.prairielearn.com), due 2/12/2025

Mid quarter feedback survey on Canvas.

Monday: Descriptions of Turing machines

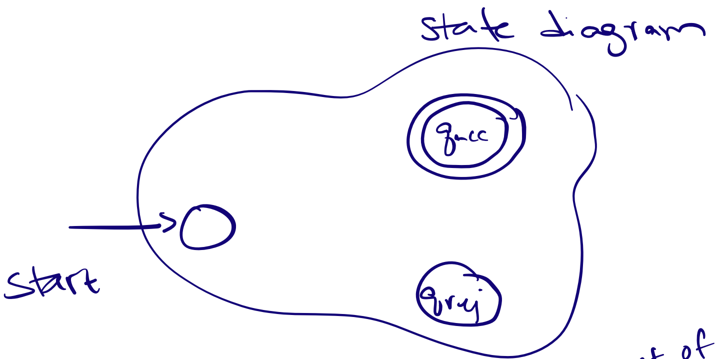
We are ready to introduce a formal model that will capture a notion of general purpose computation.

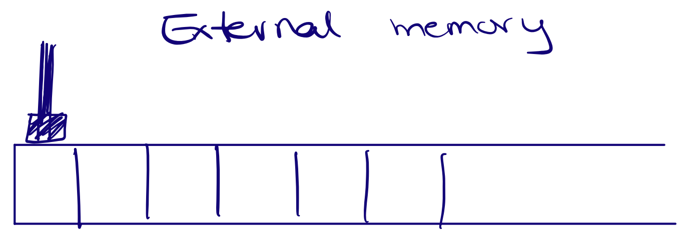
- *Similar to DFA, NFA, PDA*: input will be an arbitrary string over a fixed alphabet.
- *Different from NFA, PDA*: machine is deterministic.
- *Different from DFA, NFA, PDA*: read-write head can move both to the left and to the right, and can extend to the right past the original input.

Tape w_1 w_2 ... w_n ⊔ ⊔ ⊔ ...

- *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.

(See more details: Sipser p. 166)





set of states
input alphabet
tape alphabet

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

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

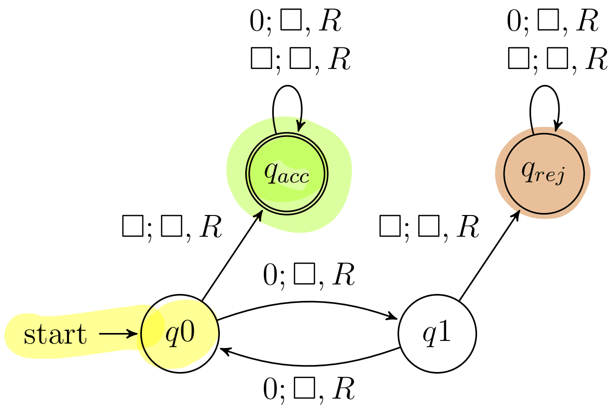
deterministic

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 $\text{␣} \in \Gamma$ and $\Sigma \subseteq \Gamma$. The blank symbol $\text{␣} \notin \Sigma$.
- 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 read/write head is at leftmost cell and transition function says it should move L, read/write head stays in its location.
- 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

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



Formal definition:

given by arrows.

Q Σ Γ

$(\{q_0, q_1, q_{acc}, q_{rej}\}, \{0\}, \{␣, 0\}, \delta, q_0, q_{acc}, q_{rej})$

Sample computation: $w = 000$

R/W HEAD

Tape

snapshot / configuration

q0↓						
0	0	0	␣	␣	␣	␣
	q1↓					
␣	0	0	␣	␣	␣	␣
		q0↓				
␣	␣	0	␣	␣	␣	␣
			q1↓			
␣	␣	␣	␣	␣	␣	␣
				qrej↓		
␣	␣	␣	␣	␣	␣	␣

Halt and reject 000

___ ; ___ , ___

tape symbol being scanned

tape symbol to write at current location of read/write head

direction to move read/write head.

$$\delta((q_0, ␣)) = (q_{acc}, ␣, R)$$

The language recognized by this machine is . . .

$$\{0^{2i} \mid i \geq 0\}$$

examples of strings that are not accepted:
0
000

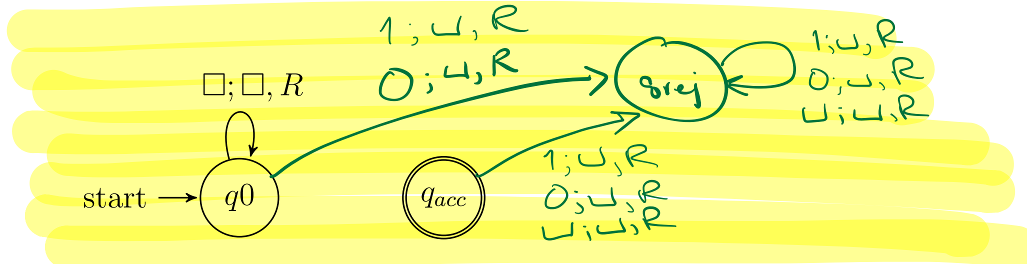
Note: this language is regular, context-free

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

- machine code **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,
- assembly **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.
- Python Java C pseudocode **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.

Conventions when draw diagrams: sometimes omit qreject from the picture ; any missing arrows directed to qreject

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



Therefore, $L(M) = \emptyset$

Example of string accepted: NONE !

Example of string rejected: 0 , 00

ε neither accepted nor rejected !

Implementation-level description

Scan to the right until we see the first 0 or 1, at which point we reject.

The language recognized by this TM is $\emptyset$

High-level description

On input x:

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

Example of string accepted: None

Example of string rejected: ε, 0, 1, 00, 01,

This TM halts on each input.

The language recognized by this TM is $\emptyset$

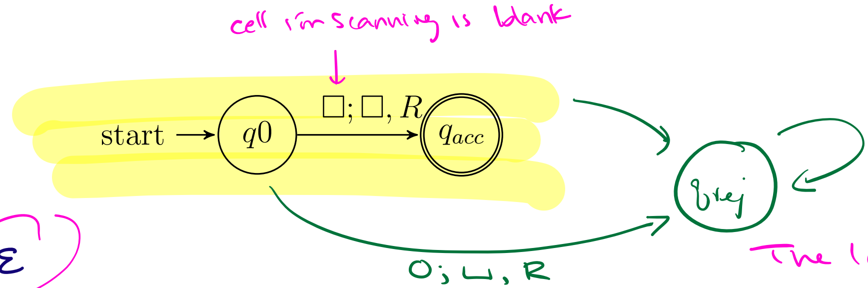
Implementation-level description

Reject (immediately).

High-level description

On input x:

1. Reject.



The language recognized by this TM is $\{\varepsilon\}$

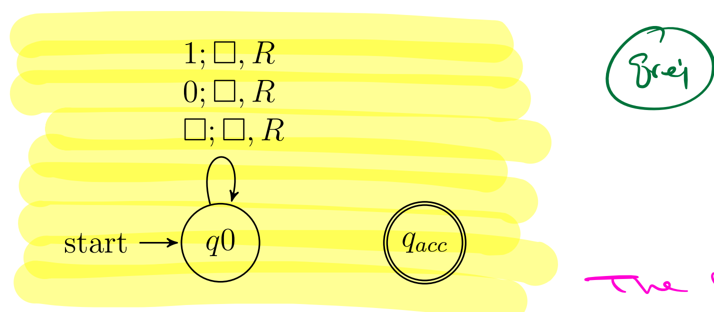
Example of string accepted: ε
Example of string rejected: 0, 1, 00, 01

Implementation-level description

If first tape symbol is blank, accept. Otherwise, reject.

High-level description

On input x:
1. If $x = \varepsilon$, accept.
2. Otherwise, reject.



The language recognized by this TM is $\emptyset$

Example of string accepted: None!
Example of string rejected: None!

Implementation-level description

Scan tape left to right, erasing each cell in turn.

write blank symbol

High-level description

On input x:

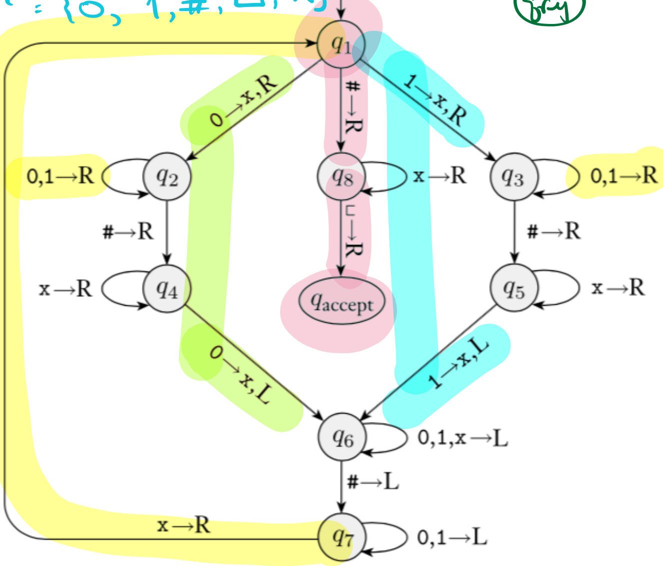
1. Go to step 1

Wednesday: Recognizable and decidable languages

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 $(q_{reject}, \sqcup, R)$

$\Sigma = \{0, 1, \#\}$
$\Gamma = \{0, 1, \#, \sqcup, x\}$



10 states

Computation on input string 01#01

$q_1 \downarrow$						
0	1	#	0	1	␣	␣
	$q_2 \downarrow$					
X	1	#	0	1	␣	␣
		$q_2 \downarrow$				
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	␣	␣
$q_7 \downarrow$						
X	1	#	X	1	␣	␣
	$q_1 \downarrow$					
X	1	#	X	1	␣	␣
		$q_3 \downarrow$				
X	X	#	X	1	␣	␣
			$q_5 \downarrow$			
X	X	#	X	1	␣	␣
				$q_5 \downarrow$		
X	X	#	X	1	␣	␣
			$q_6 \downarrow$			
X	X	#	X	X	␣	␣
		$q_6 \downarrow$				
X	X	#	X	X	␣	␣
	$q_7 \downarrow$					
X	X	#	X	X	␣	␣
		$q_1 \downarrow$				
X	X	#	X	X	␣	␣
			$q_8 \downarrow$			
X	X	#	X	X	␣	␣
				$q_8 \downarrow$		
X	X	#	X	X	␣	␣
					$q_8 \downarrow$	
X	X	#	X	X	␣	␣
						$q_{acc} \downarrow$
X	X	#	X	X	␣	␣

Notice 01#1␣ is rejected.

but 01#01 is accepted!

Implementation level description of this machine:

> 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.
>
> 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	␣	␣	␣

Motivated by our warmup examples:

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

$$L = \{ w \mid M \text{ accepts } w \}.$$

i.e. for each string in L, M accepts the string
for each string not in L, M rejects or doesn't halt on the string

A Turing machine M **recognizes** a language L means

$$L = \{ w \mid M \text{ accepts } w \}.$$

i.e. for each string, if M accepts string, then string is in L.
if M rejects string or doesn't halt on string, then string is not in L.

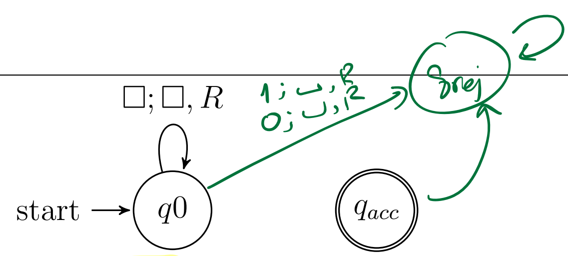
A Turing machine M is a **decider** means for each string, the computation of the M on this string halts (in only finitely many steps).

A language L is **decided by** a Turing machine M means M is a decider and M recognizes L.

i.e. for each string in L, M accepts the string
for each string not in L, M rejects the string.

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

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

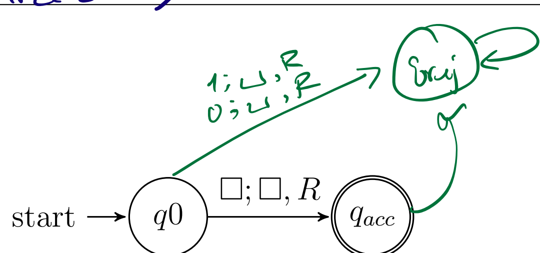


Decider? Yes / No

Recognizes $\emptyset$

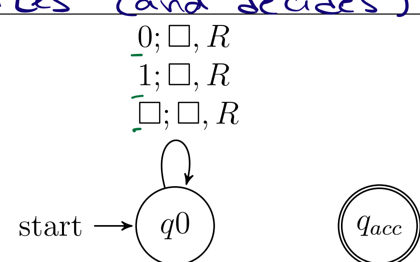
Decider? Yes / No

Recognizes (and decides) $\emptyset$



Decider? Yes / No

Recognizes (and decides) $\{\varepsilon\}$



Decider? Yes / No

Recognizes $\emptyset$

Friday: Closure for the classes of recognizable and decidable 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.

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

For all TMs, the language of TM isn't this one.

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

For all TMs that are deciders, the language of the TM isn't this one

Decider: Yes / No in only finite time

Not a decider: Yes in only finite time

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

Given a decidable language, by definition there is a TM (that is a decider) that recognizes it, so the language is recognizable.

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

Based just on definitions, Probably not ... will see counterexample soon.

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

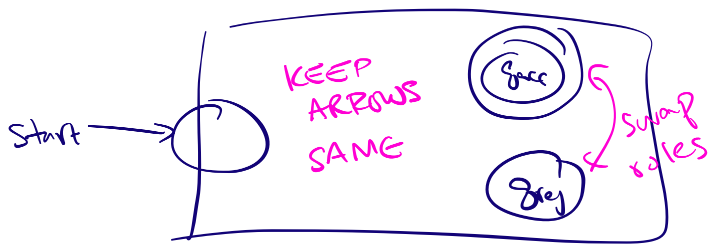
True or ~~**False**~~: Any unrecognizable language is also undecidable.

If not, we'd have an unrecognizable language that is decidable. But being decidable guarantees the language is recognizable.

True or **False**: The class of Turing-decidable languages is closed under complementation.

Suppose L is decidable. Is it the case that $\overline{L}$ is too?

Want to swap accept / reject!



deterministic.

This construction works because deciders have exactly one computation for each input string and that computation is guaranteed to halt.

Using formal definition:

Given decider $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$

we will build a new TM M_{new}

and show that $L(M_{new}) = \overline{L(M)}$ and M_{new} is a decider.

Define $M_{new} = (Q, \Sigma, \Gamma, \delta, q_0, q_{rej}, q_{acc})$

Note that M_{new} is a decider because the computations of M_{new} and M are identical (since their start states and transition functions are equal), so since all computations of M halt within finitely many steps, so do all computations of M_{new}.

Claim $L(M_{new}) = \overline{L(M)}$. To prove, consider arbitrary $w \in \Sigma^*$

Case ① $w \in L(M)$: By case assum'n M accepts w. So comp'n of M on w ends in q_{acc}. Since M_{new} has same q_0 and δ as M, computation of M_{new} is identical to that M on w and ends in q_{acc}, which has role of reject state in M_{new}. So M_{new} rejects w. i.e. $w \notin L(M_{new})$ ✓

Case ② $w \notin L(M)$: By case assum'n and since M is a decider M rejects w. So comp'n of M on w ends in q_{rej}. Since M_{new} has same q_0 and δ as M, computation of M_{new} is identical to that M on w and ends in q_{rej}, which has role of accept state in M_{new}. So M_{new} accepts w. i.e. $w \in L(M_{new})$ ✓

Using high-level description:

Given decider M we build a new TM M_{new} and show that $L(M_{new}) = \overline{L(M)}$ and M_{new} is a decider.

Define M_{new} = "On input w:

1. Run M on w
2. If M accepts, reject.
3. If M rejects, accept."

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 that $L(M_{new}) = \overline{L(M)}$ and that M_{new} is a decider:

Equivalently, we will prove that for each string w, if M accepts w then M_{new} rejects it and if M rejects w then M_{new} accepts it.

Note: these cases are exhaustive because M is a decider so there are no string for which it doesn't halt (accept or reject). Moreover, these cases will establish that M_{new} is a decider because we will show that M_{new} either accepts or rejects each input and therefore is guaranteed to halt.

Let w be an arbitrary string.

Case 1 Assume M accepts w.
We trace the computation of M_{new} on w.
In step 1, we run M on w. By case assumption, this subroutine stops after finitely many steps. In step 2, since M accepts w, we reject w. ✓

Case 2 Assume M rejects w.
We trace the computation of M_{new} on w
In step 1, we run M on w. By case assumption, this subroutine stops after finitely many steps. In step 2, since M doesn't accept w, we fall through to step 3.
In step 3, since M rejects w, we accept w. ✓

∎