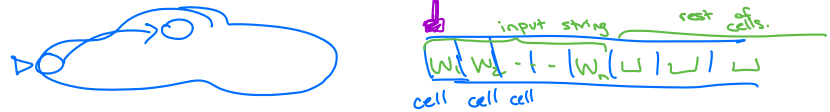


# Monday



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

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

one way infinite tape.

the **computation** of  $M$  on a string  $w$  over  $\Sigma$  is:

state char write  
state char scanning on tape  
how to move RW head on tape.

- 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  $\Gamma$  with  $\sqcup \in \Gamma$  and  $\Sigma \subseteq \Gamma$ . The blank symbol  $\sqcup \notin \Sigma$ .
- 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).
- 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

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

Example:  $(\{q_{acc}, q_{rej}\}, \{0, 1\}, \{0, 1, \sqcup\}, \delta, q_{acc}, q_{acc}, q_{rej})$



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

$\sqcup$  Blank.

Trace computation on

0 0 1

accepted

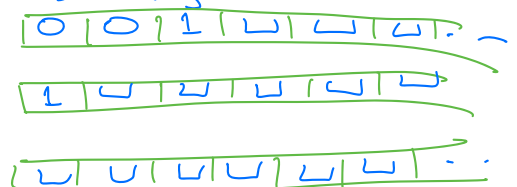
1

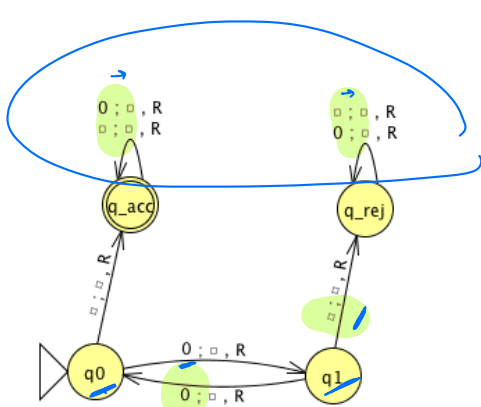
accepted

$\epsilon$

accepted.

Tape at start of computation of this machine on each input string





$\Sigma = \{0\}$

Formal definition:

(RM)

Sample computation: on 000

initial

step 1

step 2

step 3

step 4.

state	read	write	tape
q0	0	0	0
q1	0	0	0
q0	0	0	0
q1	0	0	0
q0	0	0	0
q1	0	0	0
q0	0	0	0
q1	0	0	0
q0	0	0	0
q1	0	0	0

$$\delta(q_0, 0) = (q_1, \sqcup, R)$$

$$\delta(q_1, 0) = (q_0, \sqcup, R)$$

$$\delta(q_1, \sqcup) = (q_{rej}, \sqcup, R)$$

computation halts  
in  $q_{rej}$ .  
so machine rejects 000

The language recognized by this machine is ...

To define a Turing machine, we could give a

- **Formal definition**, namely 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.

Conventions for drawing state diagrams of Turing machines: (1) omit the reject state from the diagram (unless it's the start state), (2) any missing transitions in the state diagram have value  $(q_{reject}, \sqcup, R)$ .

3.10 need to match 0



A hand-drawn diagram showing a large oval labeled  $S$  with a dashed line. Inside  $S$  is a smaller oval labeled  $S_1$ . Inside  $S_1$  is a hash symbol  $\#$ .

$q_2, q_4$   
 $q_3, q_5$   
 $q_7$   
 $q_1, q_6$

28

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

$$\Sigma = \{0, 1, \# \}$$

$$\Gamma = \{0, 1, \#, \sqcup, \times\}$$

[illegible]

*Extra practice*

Computation on input string 01#1

[illegible]

Note: over  $\{0, 1, \#\}$

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

is regular

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

is nonregular  
but context-free

$$\{\omega \neq \omega \mid \omega \in \{0,1\}^*\}$$

is not context-free

but is the recognized language by some Turing machine

## Review: Week 6 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 formal and implementation-level descriptions of Turing machines.

**Pre class reading for next time:** pages 176-177 on variants of Turing machines

### 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.

For Turing machine  $M$ ,

Wednesday

$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$$

$$= \{w \in \Sigma^* \mid \text{computation of } M \text{ on } w \text{ halts and enters accept state}\}$$

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

<p>①</p> <p><math>L(M) = \emptyset</math></p> <p>Implementation level description:  1. While current cell of tape is blank, scan to right.  2. When scan 0 or 1, reject.</p> <p>Example of string accepted: None  Example of string rejected: 0</p> <p>Notice <math>\epsilon</math> is neither accepted nor rejected.</p>	<p>②</p> <p><math>L(M) = \emptyset</math></p> <p>Implementation level description:  1. Reject.</p> <p>Example of string accepted: None  Example of string rejected: <math>\epsilon</math></p>
<p>③</p> <p><math>L(M) = \{\epsilon\}</math></p> <p>Implementation level description:  1. If current cell is blank, accept  2. Otherwise, reject.</p> <p>Example of string accepted: <math>\epsilon</math>  Example of string rejected: 0</p>	<p>④</p> <p><math>L(M) = \emptyset</math></p> <p>Implementation level description:  1. Scan the tape from left to right, erasing cells as we go.</p> <p>Example of string accepted: None  Example of string rejected: None</p> <p>Example of string on which TM does not halt: 1</p>

Two models of computation are called **equally expressive** when every language recognizable with the first model is recognizable with the second, and vice versa.

True / False: NFAs and PDAs are equally expressive.

True / False: Regular expressions and CFGs are equally expressive.

reg exp and DFAs are equally expressive.  
reg exp and NFAs are equally expressive.

To say a language is **Turing-recognizable** means that there is some Turing machine that recognizes it.

Some examples of models that are **equally expressive** with deterministic Turing machines:

new model

# May-stay machines

The May-stay machine model is the same as the usual Turing machine model, except that on each transition, the tape head may move L, move R, or Stay.

Formally:  $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$  where

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

**Claim:** Turing machines and May-stay machines are equally expressive. To prove ...

To translate a standard TM to a may-stay machine:

Given a TM, define May-stay machine that mimics this TM exactly (never uses the "S" option).

Formally, given  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$  a TM, define the May-stay machine  $(Q, \Sigma, \Gamma, \delta_{new}, q_0, q_{acc}, q_{rej})$  with  $\delta_{new} : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$  given by  $\delta_{new}(q, x) = \delta(q, x)$  for all  $q \in Q, x \in \Gamma$ .

To translate one of the may-stay machines to standard TM: any time TM would Stay, move right then left.

Formally: suppose  $M_S = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$  has  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$ . Define the Turing-machine

$$M_{new} = (Q_{new}, \Sigma, \Gamma, \delta_{new}, (q_0, 0), (q_{acc}, 0), (q_{rej}, 0))$$

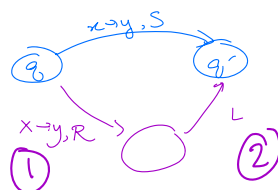
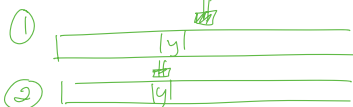
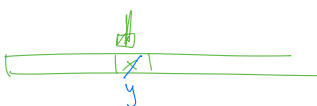
with  $Q_{new} = Q \times \{0, 1\}$  two tagged copies of each state

and  $\delta_{new} : Q_{new} \times \Gamma \rightarrow Q_{new} \times \Gamma \times \{L, R\}$  given by

$$\delta_{new} \left( \begin{matrix} \text{state} \\ (q, i) \end{matrix}, \begin{matrix} \text{content of tape cell} \\ x \end{matrix} \right) = \begin{cases} ((q', 0), y, d) & \text{if } q \in Q, i=0, \delta(q, x) = (q', y, d), d \in \{L, R\} \\ \text{---} \\ ((q', 1), y, R) & \text{if } q \in Q, i=0, \delta(q, x) = (q', y, S) \\ \text{---} \\ ((q, 0), x, L) & \text{if } i=1, q \in Q, x \in \Gamma \end{cases}$$

Informally:

if old machine



## Multitape Turing machine

A multitape Turing machine with  $k$  tapes can be formally represented as  $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$  where  $Q$  is the finite set of states,  $\Sigma$  is the input alphabet with  $\sqcup \notin \Sigma$ ,  $\Gamma$  is the tape alphabet with  $\Sigma \subsetneq \Gamma$ ,  $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k$  (where  $k$  is the number of states)

If  $M$  is a standard TM, it is a 1-tape machine.   
*cur state* *cells on all tape* *next state* *update all tapes* *move all RW heads*

To translate a  $k$ -tape machine to a standard TM: Use a new symbol to separate the contents of each tape and keep track of location of head with special version of each tape symbol. Sipser Theorem 3.13

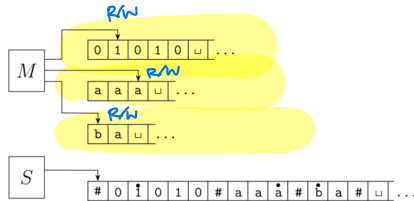


FIGURE 3.14  
Representing three tapes with one

A multitape Turing machine is equally expressive to a (regular, one tape (one-way infinite), deterministic Turing machine)

*Extra practice:* **Wikipedia Turing machine** Define a machine  $(Q, \Gamma, b, \Sigma, q_0, F, \delta)$  where  $Q$  is the finite set of states  $\Gamma$  is the tape alphabet,  $b \in \Gamma$  is the blank symbol,  $\Sigma \subsetneq \Gamma$  is the input alphabet,  $q_0 \in Q$  is the start state,  $F \subseteq Q$  is the set of accept states,  $\delta : (Q \setminus F) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$  is a partial transition function. If computation enters a state in  $F$ , it accepts. If computation enters a configuration where  $\delta$  is not defined, it rejects. Hopcroft and Ullman, cited by Wikipedia



**Enumerators** Enumerators give a different model of computation where a language is **produced, one string at a time**, rather than recognized by accepting (or not) individual strings.

Each enumerator machine has finite state control, unlimited work tape, and a printer. The computation proceeds according to transition function; at any point machine may "send" a string to the printer.

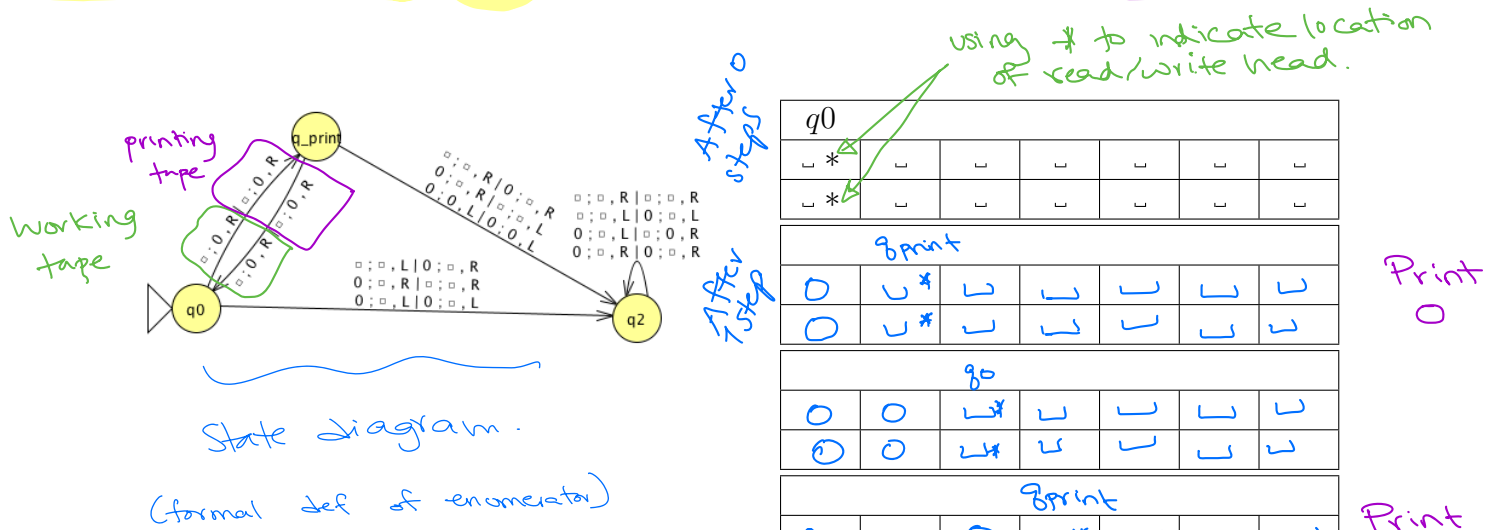
$$E = (Q, \Sigma, \Gamma, \delta, q_0, \underline{q_{print}})$$

$Q$  is the finite set of states,  $\Sigma$  is the output alphabet,  $\Gamma$  is the tape alphabet ( $\Sigma \subsetneq \Gamma, \sqcup \in \Gamma \setminus \Sigma$ ),

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

where in state  $q$ , when the working tape is scanning character  $x$  and the printer tape is scanning character  $y$ ,  $\delta((q, x, y)) = (q', x', y', d_w, d_p)$  means transition to control state  $q'$ , write  $x'$  on the working tape, write  $y'$  on the printer tape, move in direction  $d_w$  on the working tape, and move in direction  $d_p$  on the printer tape. The computation starts in  $q_0$  and each time the computation enters  $q_{print}$  the string from the leftmost edge of the printer tape to the first blank cell is considered to be printed.

The language **enumerated** by  $E$ ,  $L(E)$ , is  $\{w \in \Sigma^* \mid E \text{ eventually, at finite time, prints } w\}$ .



Implementation level description:

1. Write a 0 on each of the working and printer tapes and move each R/W head one step to the right.
2. for  $i=1, 2, \dots$ 
  - a. Write two consecutive 0s on each of the working and printer tapes, moving R/W head to the R each time.
  - b. Print contents of printer tape.

High level description: extra practice

$$L(E) = \{ 0^{2n+1} \mid n \geq 0 \}$$

(✓)

$$L(M) = \{w \mid M \text{ accepts } w\}$$

$$L(E) = \{w \mid E \text{ prints } w \text{ at finite time}\}$$

**Theorem 3.21** A language is Turing-recognizable iff some enumerator enumerates it.

**Proof:**

Assume  $L$  is enumerated by some enumerator,  $E$ , so  $L = L(E)$ . We'll use  $E$  in a subroutine within a high-level description of a new Turing machine that we will build to recognize  $L$ .

**Goal:** build Turing machine  $M_E$  with  $L(M_E) = L(E)$ .

Define  $M_E$  as follows:  $M_E =$  "On input  $w$ ,  
subroutine"

instructions  
for a  
computation  
of TM  $M_E$

1. Run  $E$ . For each string  $x$  printed by  $E$ .
2. Check if  $x = w$ . If so, accept (and halt); otherwise, continue."

WTS  $L(M_E) = L(E) = L$

need to prove (based on construction),

why? Let  $w$  be arbitrary string.

Assume  $w \in L$ . Then  $w \in L(E)$  (since we picked  $E$  such that  $L(E) = L$ ) so  $w$  is printed by  $E$  at some finite time. Namely, it shows up as  $x$  at some point in step 2 of the alg defining  $M$ . At this point,  $M$  halts and accept  $w (=x)$  ✓

Now, assume  $w \notin L$ . Then  $w$  is never printed by  $E$ . So  $M$  in step 2 always evaluates " $x=w$ " to F and never halts. so  $w \notin L(M)$  ✓.

Assume  $L$  is Turing-recognizable and there is a Turing machine  $M$  with  $L = L(M)$ . We'll use  $M$  in a subroutine within a high-level description of an enumerator that we will build to enumerate  $L$ .

**Goal:** build enumerator  $E_M$  with  $L(E_M) = L(M)$ .

**Idea:** check each string in turn to see if it is in  $L$ .

**How?** Run computation of  $M$  on each string. But: need to be careful about computations that don't halt.

Recall String order for  $\Sigma = \{0,1\}$ :  $s_1 = \varepsilon$ ,  $s_2 = 0$ ,  $s_3 = 1$ ,  $s_4 = 00$ ,  $s_5 = 01$ ,  $s_6 = 10$ ,  $s_7 = 11$ ,  $s_8 = 000$ , ...

high level description

Define  $E_M$  as follows:  $E_M =$  "ignore any input. Repeat the following for  $i = 1, 2, 3, \dots$

1. Run the computations of  $M$  on  $s_1, s_2, \dots, s_i$  for (at most)  $i$  steps each

Subroutines

2. For each of these  $i$  computations that accept during the (at most)  $i$  steps, print out the accepted string."

bound on  
resources for  
single iterations

Claim:  $L(E) = L(M)$ .

## Review: Week 6 Wednesday

Please complete the review quiz questions on [Gradescope](#) about variants of Turing machines.

**Pre class reading for next time:** Theorem 3.16 on page 178 (nondeterminism)

Friday

Turing machines are equally expressive w/ Multi-tape machines and enumerators.

### Nondeterministic Turing machine

At any point in the computation, the nondeterministic machine may proceed according to several possibilities:  $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$  where

$$\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

The computation of a nondeterministic Turing machine is a tree with branching when the next step of the computation has multiple possibilities. A nondeterministic Turing machine accepts a string exactly when some branch of the computation tree enters the accept state.

Given a nondeterministic machine, we can use a 3-tape Turing machine to simulate it by doing a breadth-first search of computation tree: one tape is "read-only" input tape, one tape simulates the tape of the nondeterministic computation, and one tape tracks nondeterministic branching. Sipser page 178

Nondeterministic TMs equally expressive with deterministic TMs.

Two models of computation are called **equally expressive** when every language recognizable with the first model is recognizable with the second, and vice versa.

**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.

Computation  
precise step-by-step.

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

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

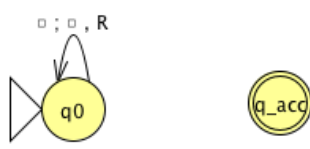


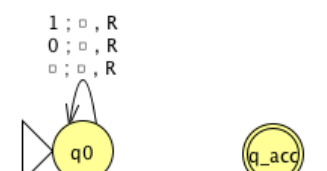
A Turing machine  $M$  is a **decider** means

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

A Turing machine  $M$  **decides** a language  $L$  means

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

Postpone  
to  
week 7

 <p>Decider? Yes / No</p>	 <p>Decider? Yes / No</p>
 <p>Decider? Yes / No</p>	 <p>Decider? Yes / No</p>

## 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.

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”

- 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?
- Do two NFAs have any state names in common?
- Do two CFGs have the same start variable?

Postpone  
to  
week 7

---

<sup>1</sup>An introduction to ASCII is available on the w3 tutorial [here](#).

## **Review: Week 6 Friday**

Please complete the review quiz questions on [Gradescope](#) about descriptions of Turing machines.

**Pre class reading for next time:** Page 184-185 Terminology for describing Turing machines