Monday May 2

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

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

- 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

$L(M) = \{w \in \Sigma^* \mid \text{computation of } M \text{ on } w \text{ halts after entering the accept state}\} = \{w \in \Sigma^* \mid w \text{ is accepted by } M\}$

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.

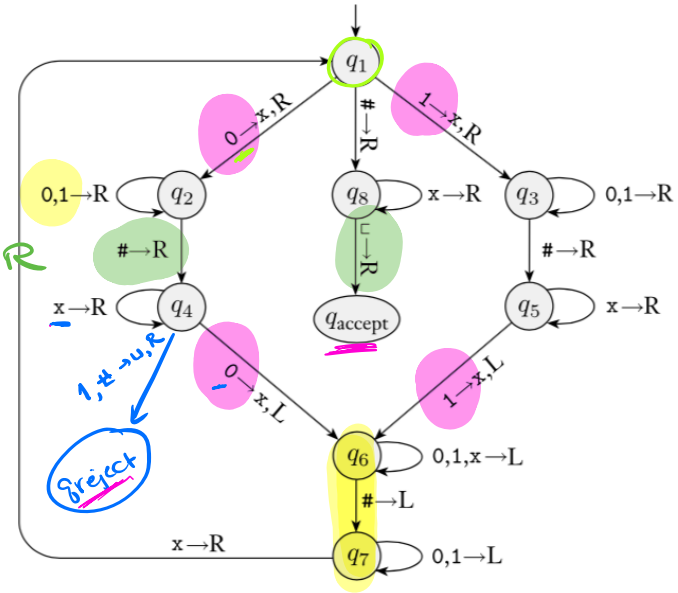
Conventions for drawing state diagrams of Turing machines: (1) omit the reject state from the diagram (unless it's the start state), (2) any missing transitions in the state diagram have value $(q_{reject}, \sqcup, R)$.

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

$\{w\#w^R \mid w \in \{0,1\}^*\}$ context-free and nonregular

$\{w\#w \mid w \in \{0,1\}^*\}$ not context free but is the language of a Turing mach.

Sipser Figure 3.10



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

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

Computation on input string 01#01

q_1						
0	1	#	0	1	␣	␣
X	q_2	#	0	1	␣	␣
X	1	#	q_4	1	␣	␣
X	1	#	0	q_5	␣	␣
X	1	#	X	1	q_6	␣
X	1	#	X	q_7	␣	␣
X	1	#	X	1	q_7	␣
X	1	#	X	1	␣	q_8
X	q_1	#	X	1	␣	␣
X	X	#	q_3	1	␣	␣
X	X	#	X	q_5	␣	␣
X	X	#	X	1	q_5	␣
X	X	#	X	X	␣	q_6
X	X	#	X	X	q_6	␣
X	X	#	X	X	␣	q_7
X	X	#	X	X	q_7	␣
X	X	#	X	X	␣	q_8
X	X	#	X	X	q_8	␣
X	X	#	X	X	␣	␣
X	X	#	X	X	␣	q_{accept}

01#01 is accepted by this M because the computation of this TM on input 01#01 halts in finite time by entering q_{accept}.

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

i.e. For each string, if the string is in L, M accepts it, and. if the string is not in L, M will not accept it.

A Turing machine M **recognizes** a language L means $L = L(M) = \{w \mid M \text{ accepts } w\}$

A Turing machine M is a **decider** means M is a Turing machine such that for any input string w, the computation of M on w halts.

A language L is **decided by** a Turing machine M means $L(M) = L$ and M is a decider

i.e. For each string, if string is in L, M accepts it, and if the string is not in L, M rejects it.

A Turing machine M **decides** a language L means M is a decider and $L(M) = L$.

Fix $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \sqcup\}$ for the Turing machines with the following state diagrams: $L(M_1) = L(M_2)$

M_1

Implementation level description:
If leftmost tape cell is blank, scan to the right forever
Otherwise, reject

Example of string accepted: None
Example of string rejected: 0

Decider? Yes / (No)

Example of string on which machine loops: ε

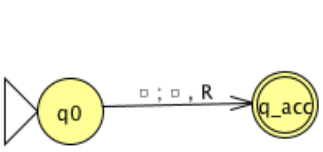
M_2

Implementation level description:
Reject

Example of string accepted: None
Example of string rejected: ε 0 0110

Decider? (Yes) / No

M_3



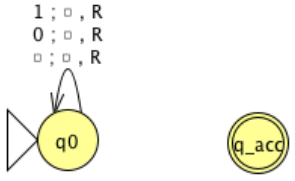
Implementation level description:
If leftmost tape cell is blank, accept.
Else reject.

Example of string accepted: ε
Example of string rejected: 0

Decider? (Yes) / No

$L(M_3) \neq L(M_1)$ $L(M_3) = \{\varepsilon\}$ $L(M_1) = \{\}$ $L(M_3)$

M_4



Implementation level description:
Scan right along tape

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

Decider? Yes / (No)

$L(M_4) = \{\} = L(M_1) = L(M_2)$

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

$Q = \{q_0, q_{acc}, q_{rej}\}$

$Q \times \{0,1\} = \{(q_0,0), (q_0,1), (q_{acc},0), (q_{acc},1), (q_{rej},0), (q_{rej},1)\}$.

Wednesday May 4

CFGs and PDAs are equally expressive (Ch 2 of Sipser)

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.

Consider the language $\{a^n b^n \mid n \geq 0\}$ which is recognizable by a PDA but not recognizable by any NFA. The language $\{a^n b^n \mid n \geq 0\}$ is generated by some CFG but not described by any regular expression.

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

Fact: NFA and DFA are equally expressive.

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

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

7-tuple

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

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

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

Given $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ a TM,

define the May-Stay machine $(Q, \Sigma, \Gamma, \delta_{new}, q_0, q_{accept}, q_{reject})$

where $\delta_{new}: Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ with

$\delta_{new}((q,x)) = \delta((q,x))$ 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 $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$. Define the Turing-machine

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

Copy of each state to use intermediate version

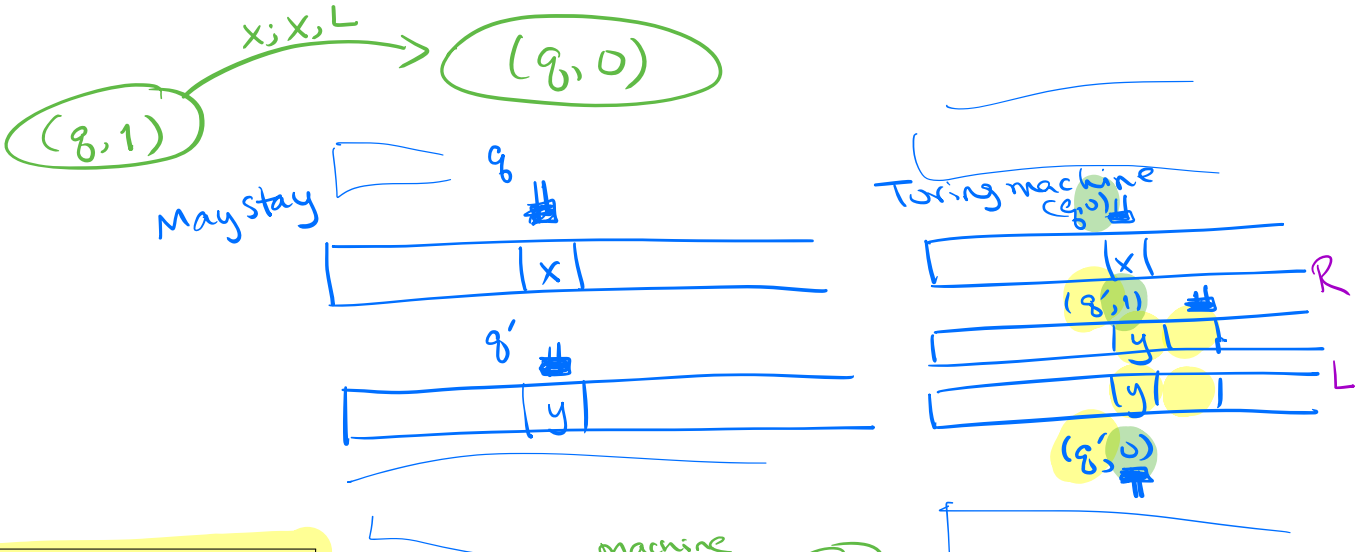
$Q_{new} = Q \times \{0, 1\}$ two tagged copies of each state

$\delta_{new}: Q_{new} \times \Gamma \longrightarrow Q_{new} \times \Gamma \times \{L, R\}$

curr contents of tape

$\delta_{new}((q,i),x) = \begin{cases} ((q',0),y,d) & \text{if } \delta(q,x) = (q',y,d) \text{ and } d \in \{L,R\} \\ & i=0,\ q \in Q,\ x \in \Gamma \\ ((q',1),y,R) & \text{if } \delta(q,x) = (q',y,S) \\ \text{first half} & i=0,\ q \in Q,\ x \in \Gamma \\ ((q',0),\underline{X},L) & i=1\ q \in Q\ x \in \Gamma \\ \text{second half} & \end{cases}$

$q \in Q$
$i \in \{0,1\}$
$x \in \Gamma$



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

If M is a standard TM, it is a 1-tape machine.

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

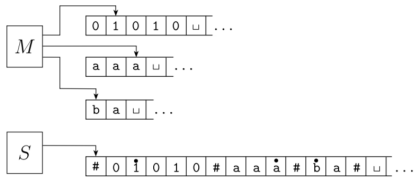


FIGURE 3.14
Representing three tapes with one

Will help simulate Nondeterminism!

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

Friday

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.

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

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

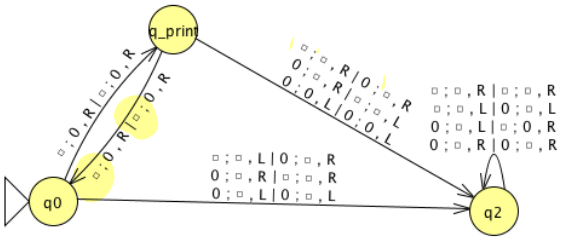
$$\delta : Q \times \Gamma \times \Gamma \to Q \times \Gamma \times \Gamma \times \{L, R\} \times \{L, R\}$$

with both tapes & blank

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 $\{w \in \Sigma^* \mid E$ eventually, at finite time, prints $w\}$.

E



read-write head.

$q0$						
$\sqcup$ *	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\sqcup$ *	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
q_{print}						
0	$\sqcup$ *	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
0	$\sqcup$ *	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
q_0						
0	0	$\sqcup$ *	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
0	0	$\sqcup$ *	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$

$0 \in L(E)$

$L(E) = \{ 0^{2i+1} \mid i \geq 0, i \in \mathbb{Z} \}$

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

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 (nondeterministism)

Friday May 6

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.

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,

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

high level description of TMs.

Notice: for any string w, if $w \in L(E)$ then M accepts w.
for any string w, if $w \notin L(E)$ then M on input w loops on w so does not accept w.

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

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

1. Run the computations of M on s_1, s_2, ..., s_i for (at most) i steps each
2. For each of these i computations that accept during the (at most) i steps, print out the accepted string."

For any string w not in $L(M)$, then M never accepts w so E will never print w because step 2 of definition of E requires evidence that M accepts a string before it prints that string.

For any string w if $w \in L(M)$ then M accepts w in N_2 steps and w is S_{N_2} for some N_2 so the high level description of E will print w in stage where $i = \max(N_2, N_2)$.

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

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

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 TM equally expressive with (regular deterministic) Turing machines.

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.

Extra Practice

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

Proof using Turing machines:

Given TMs M_1, M_2 want to build a TM M with $L(M) = L(M_1) \cup L(M_2)$

Define M = "On input w

1. For $i = 1, \ldots$
2. Run computation of M_1 on w for up to i steps. If M_1 accepts during these i steps, halt and accept. If M_1 rejects during these i steps, go to step 3.
3. Run computation of M_2 on w for up to i steps. If M_2 accepts during these i steps, halt and accept. If M_2 rejects during these i steps, continue"

To prove: Consider arbitrary string accepted by M; WTS in $L(M_1) \cup L(M_2)$ -- Conversely, consider arbitrary string in $L(M_1) \cup L(M_2)$ WTS accepted by M. ---

Proof using nondeterministic Turing machines:

Given TMs M_1, M_2 want to build a nondeterministic TM N with $L(N) = L(M_1) \cup L(M_2)$. Define

N = "On input w

1. Nondeterministically choose whether to simulate M_1 or M_2.
2. Run M_i on w. Accept if M_i accepts. Reject if M_i rejects".

To prove: Consider arbitrary string accepted by N; WTS in $L(M_1) \cup L(M_2)$. Conversely, consider arbitrary string in $L(M_1) \cup L(M_2)$ WTS accepted by N ... ---

use nondeterministic definition of acceptance.

Proof using enumerators:

Given enumerators E_1, E_2 want to build an enumerator E with $L(E) = L(E_1) \cup L(E_2)$. Define

E = "<no/ignore input>

1. Alternate simulating E_1, E_2 one step from each at a time.
2. When either simulation would print a string, print that string, and continue."

To prove: Consider arbitrary string printed by E; WTS in $L(E_1) \cup L(E_2)$. Conversely, consider arbitrary string in $L(E_1) \cup L(E_2)$ WTS that string printed by E --

use properties of sets! repetition in list is ok!

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