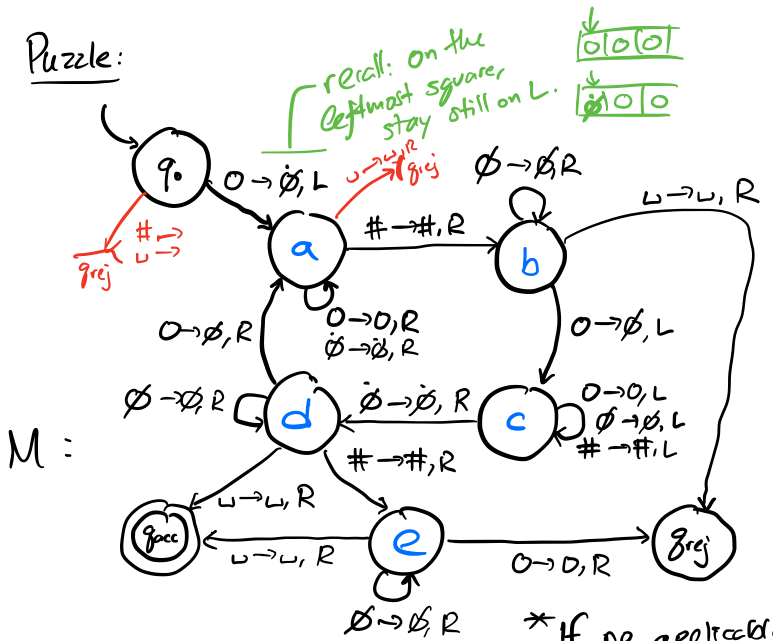
Recall: TMs are automata with RAM (an infinite tape)

On each step $(a \to b, D)$ we (1) read a char 'a' from the tape, (2) change internal state, (3) write a char 'b' to the tape, and (4) move a direction $D \in \{L, R\}$.

Puzzle:



TM state diagram:

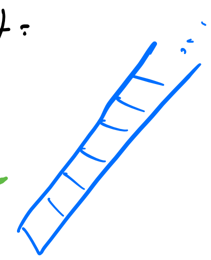
$\Sigma = \{0, \#\}$

$\Gamma = \{0, \#, \sqcup, \emptyset, \dot{\emptyset}\}$

* If no applicable transitions, proceed to q_{reject} and halt.

Do we accept:

0?	✗
0#?	✗
0#0?	✓
0#00?	✗
00#0?	✗
00#00?	✓



tape head

state	tape
q_0	0
a	$\dot{\emptyset}$
a	$\dot{\emptyset}\sqcup$
q_{rej}	

state	tape
q_0	0#
a	$\dot{\emptyset}$#
a	$\dot{\emptyset}$#
b	$\dot{\emptyset}\#\sqcup$
q_{rej}	$\dot{\emptyset}\#\sqcup\sqcup$

state	tape
q_0	00#0
a	$\dot{\emptyset}$0#0
a	$\dot{\emptyset}$0#0
a	$\dot{\emptyset}$0#0
b	$\dot{\emptyset}$0#0
c	$\dot{\emptyset}$0#$\emptyset$
c	$\dot{\emptyset}$0#$\emptyset$
c	$\dot{\emptyset}$0#$\emptyset$
d	$\dot{\emptyset}$0#$\emptyset$

state	tape
a	$\dot{\emptyset}\emptyset\#\emptyset$
b	$\dot{\emptyset}\emptyset\#\emptyset$
b	$\dot{\emptyset}\emptyset\#\emptyset$0
q_{rej}	$\dot{\emptyset}\emptyset\#\emptyset\sqcup$
c	$\dot{\emptyset}\emptyset\#\emptyset\emptyset$
c, c, c	$\dot{\emptyset}\emptyset\#\emptyset\emptyset$
d, d	$\dot{\emptyset}\emptyset\#\emptyset\emptyset$
e	$\dot{\emptyset}\emptyset\#\emptyset\emptyset$
e, e	$\dot{\emptyset}\emptyset\#\emptyset\emptyset\sqcup$
q_{acc}	

$\{0^n \# 0^n \mid n > 0\}$

$0^n 1^n$

M = "On input w:

(1) If w starts with a 0, cross it out and move right to the first #.

(2) cross out a zero to the right of the first #. (If no uncrossed zeros, reject).

(3) Return to leftmost square; accept if all 0's are crossed out; else return to (1)."

Today:

1. Build an "API / Library" for TMs.
2. Simulating automata.
3. The Church-Turing Thesis.

Def. Given a TM M, L(M) denotes the language recognized by M: all strings that M accepts.

Def. A TM decides a language L if it accepts all strings in L and rejects all strings not in L. (A TM that always halts is a decider.)

Corr. TM-decidable $\subseteq$ TM-recognizable.

Things TMs can do.

- TMs can check membership in a given regular language.

Proof. Let L(D) be the language corresponding to the DFA D.
Let $\delta_D : Q \times \Sigma \to Q$ be D's transition function.
Build a TM M with

$$\delta_M : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$$

$$\delta_M(q,a) = (\underbrace{\delta_D(q,a)}_{\text{change states according to } \delta_D},\ \underbrace{a}_{\text{write a}},\ \underbrace{R}_{\text{move R}}).$$

M "simulates" D by "hard-coding".

- TMs can detect the leftmost and rightmost tape squares.

Proof - add L- and R-marked versions of each character to the tape alphabet. Before execution,

(1) mark leftmost symbol with L,
(2) move right to the first ␣ and mark it with an R,
(3) move back to start.

Treat L- and R-marked symbols the same as unmarked symbols, preserving markings.

- TMs can "count" by shuffling back and forth, crossing off symbols.
  - check if two substrings are the same length
  - check if two substrings are the same,
  - "count" the length of a substring w by writing $|w|$ markers (say, x's) to the tape, etc.

Example: $A = \{0^k \# 0^k \mid k \geq 0\}$ ⟼ see above

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

M_B = " On input u:

(1) Check, using the DFA which accepts $\Sigma^* \# \Sigma^*$, that my input matches this regular expression.
(If no: reject. If yes: proceed.)

(2) Move back to the leftmost tape square

(3) Accept if u = "#" uncrossed.

(4) Cross off the first uncrossed character and "remember" if it is a 0 or 1 using two different internal states.

(5) Move right until #

(6) Cross off the next uncrossed character and reject unless it matches the character crossed off in step 4.

(7) Return to leftmost square and continue from (4) crossing off the next uncrossed character.

(8) If I run out of characters on either side of # first, reject. "

$\cancel{0}\cancel{1}\cancel{0}\#\cancel{0}\cancel{1}\cancel{0}$ ✓

$\cancel{0}\cancel{1}\cancel{1}\#\cancel{0}\cancel{1}0$ → reject.

$\cancel{0}10\#\cancel{0}10$

Back at 2:31

- TMs can "hard-code" and simulate the functions of other TMs.

Proof sketch. Let M_1 be some TM. We'll build a TM M_2 that simulates M_1 on M_2's input.

M_2 = "On input w:

(1) Move to the rightmost tape square and add a delimiting '#' character.

(2) Shuttle back and forth, copying the input character by character until the tape contains the string $w\#w$.

(3) Move to the first character of the second w substring, and simulate M_1 by following M_1's transition function with the following exceptions.

- treat # as the leftmost tape square: if we reach #, we go back one square right.
- If we reach M_1's q_{acc} or q_{rej}, we don't halt, but instead "break" and continue execution of M_2.

(4) ... M_2 continues... "

Example. element distinctness

$$E = \{\#x_1\#x_2\#\cdots\#x_\ell \mid \text{each } x_i \in \{0,1\}^*, \text{ and } x_i \neq x_j \text{ for all } i \neq j\}.$$

M_E = "On input w:

(1) Simulate a hard-coded DFA to check that w matches the regular expression $(\#(0 \cup 1)^*)^+$.
If not, reject.

(2) Mark the first two #'s with a dot ($\dot{\#}$).
(If we have only one input string, accept.)

(3) Simulate a hard-coded copy of M_B

to check if the two strings preceded by $\dot{\#}$ are equal.
If so, reject. If not, continue.

(4) Move the second dot to the next # and repeat (3); if the second dot is on the last #, move the first dot instead and move the second dot to the # immediately after.

(5) Accept after comparing all pairs of inputs."

$\# x_1 \# x_2 \# x_3 \cdots \# x_\ell \quad \#' \underline{x_1 \# x_2}$ ← M_2 on this string.

$$F = \{a^i b^j c^k \mid ij = k \text{ and } i, j, k \geq 1\}$$

M_F = "On input w:

aaabbcccccc
abbbccc
aabbcccc

1. Mark left, right ends of tape.
2. (Use a hard-coded DFA to) Check that the input matches $a^+b^+c^+$, reject if not.
3. Cross off one a:
   - 3a. (shuttle back and forth), crossing off a 'c' for each 'b.' If we run out of c's, reject.
   - 3b. If we run out of b's: uncross all b's, and repeat from (3).
4. If we run out of a's, accept if and only if all c's are crossed out."

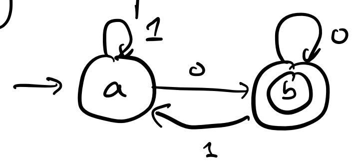
~~aa~~bbb~~cccccc~~

Simulating Automata

$A_{DFA} = \{\langle D, w\rangle \mid D$ encodes a DFA, w is a string, and D accepts $w\}$

input string contains a complete representation of D, written in some alphabet, that our TM can be programmed to decode.

Encoding example:

D: 

"ends in 0"

$$D = (Q, \Sigma, q_0, F, \delta) = \left(\{a,b\}, \{0,1\}, a, \{b\}, \begin{array}{c|cc} \delta & a & b \\ \hline 0 & b & b \\ 1 & a & a \end{array}\right)$$

$\langle D\rangle = [[ab][01]a[b][[a0b][a1a][b0b][b1a]]]$

$\Sigma = \{[,], a, b, 0, 1\}$

M_{DFA} = "On input x:

(1) Check to see if the input encodes some DFA and string D, w.

(e.g., match reg. ex $[[\Sigma^+][\Sigma^+]\Sigma[\Sigma^*][([\Sigma\Sigma\Sigma])^*]]$

and check is $q_0 \in Q$? is $F \subseteq Q$? etc.)

(2) Place a "tape head marker" ($\downarrow$) on the leftmost character of w. Write down q_0, our "current state" on the tape.

tape: $\langle D\rangle \overset{\downarrow}{w_1} w_2 \cdots w_k \# a$

(3) Simulate D on w by following D's transition function: read the ↓-marked char and current DFA state, read D's transition function, change the DFA state accordingly, and increment (↓) to the next character.
If our simulation of D on w accepts, accept."

Back at 3:35

$A_{TM} = \{\langle M, w\rangle \mid M$ is an encoded TM, w a string, M accepts $w\}$

U_{TM} = " On input x:
(1) Check that x encodes a TM M and a string w
(2) Mark M's position of tape head on w, and record M's current (start) state.
(3) Follow M's transition function to simulate M on w.
↳ Accept if $M(w)$ accepts,
↳ reject if $M(w)$ rejects. " *

* runs forever if $M(w)$ runs forever

"M run on w"

Recall: a decider halts and accepts or rejects each string.

Is U_{TM} a decider? No.

U_{TM} recognizes A_{TM}, but doesn't decide it.

"accepts every string in"

$HALT_{TM} = \{\langle M, w\rangle \mid M$ encodes a TM, w a string, and $M(w)$ halts.$\}$

The Church-Turing Thesis

Things doable by a TM ≈ things doable by a 'recipe', 'algorithm', 'computational process' generally.

A "computational system" (machine, prog. language, game, etc.) is Turing-complete if it can simulate a TM.

Under the Church-Turing thesis, this is equivalent to being able to perform arbitrary computation.

- Msft Excel, PPT
- Most PLs
- Conway's game of life.
- Minecraft, Portal, MtG

$E_{DFA} = \{\langle D\rangle \mid \langle D\rangle$ encodes a DFA that rejects all strings.$\}$

Idea 1: Simulate all strings on D, in turn:

$\varepsilon, 0, 1, 00, 01, \cdots$

$M_{E_{DFA}}$: "On input w:

1) Check that the input is an encoded DFA.
2) Mark the 'start state', and add the start state to a list of reachable states.
3) Follow all transitions from start state, and add reachable states to our list
4) Continue this BFS until we try all out-transitions from reachable states
(5) Accept if and only if no accept state is reachable."