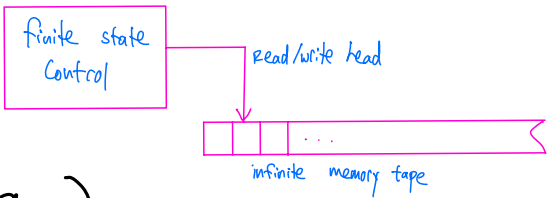
Announcements:

- Quiz 1 solution sketch + regrade open soon } will announce
- Handout 2 out } yesterday
- HW2 due



Last time:

- Defined TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$
- Configuration, acceptance, $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$
  - Defined decider ⇝ halts (acc/rej) on all inputs!
  - L is TM-recognizable, TM decidable

∃ TM M s.t. L(M)=L
i.e.: ∀x
x ∈ L ⇒ M accepts x
x ∉ L ⇒ M either rejects x or runs forever on x

∃ TM M s.t. L(M)=L, and M is a decider
i.e.: ∀x
x ∈ L ⇒ M accepts x
x ∉ L ⇒ M rejects x

Differences between TMs and DFAs:

- can go left and right (not just right)
- unbounded memory (on tape)
- can read and write (not just read)
- immediately halts upon entering q_{acc} or q_{rej} (while DFA keeps going till end of string)

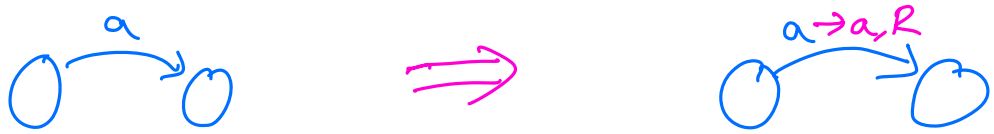
$\{\text{regular languages}\} \subsetneq \{\text{TM-decidable}\} \subsetneq \{\text{TM-recognizable}\} \subsetneq \{\text{all } L\}$

why? $\subseteq$ $\subseteq$ (by def) $\subseteq$ (by def) ?

Claim: If L is regular then L is TM-decidable

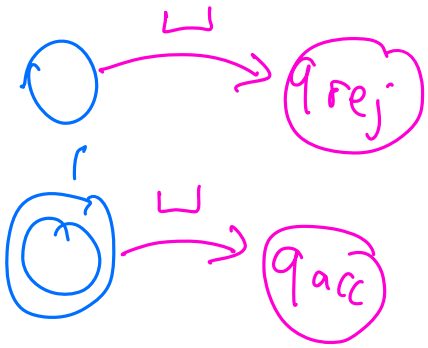
proof sketch: We can simulate a DFA with a TM

just go right, never write anything:



When you get to ␣, accept or reject depending on DFA state:

if accepting, go to q_{acc},
if rejecting, go to q_{rej}



↗ see solutions at the end

Exercise: (1) write this construction formally

(2) Draw the resulting decider for $\{w \in \{a,b\}^* \mid w \text{ contain a 'b'}\}$

But TMs can also decide non-regular languages!

Ex: $L = \{a^n b^n \mid n \geq 0\}$ $(\Sigma = \{a, b\})$ ⇐ non-regular lang.

TM decider for L?

Idea: use $\Gamma = \{a, b, \sqcup, X, Y\}$

X, Y: helper characters we can write on tape

In section 1 and 2 different algorithms were suggested, both following the same high level idea: keep going through tape, each time "crossing off" one a and one b, if you finished without any a/b remaining - accept, if along the way there's imbalance or wrong order, reject.

I'm writing both versions here, even though they are very similar, to be consistent with students' notes. The only difference is in which order we cross the bs, right to left or left-to-right.

Section 1 algorithm:

Repeat:

- move right, replace first 'a' you see with 'X'
  - if you see 'b' before any 'a' - reject
  - if you see '␣' before any 'a' - accept
- move right till you see ␣ (i.e. end of string)
- move left, replace first 'b' you see with Y
  - if you see 'a' or X before any 'b' - reject
- go back left till you see X

Section 2 algorithm: (first step same)

Repeat:

- move right, replace first 'a' you see with 'X'.
  - if you see 'b' before any 'a' – reject
  - if you see '␣' before any 'a' – accept
- move right, replace first 'b' you see with Y
  - if you see '␣' before b – reject
- go back left till you see X

Idea in both cases:

- Whenever we're at the top of the loop:
  we have marked off same # of a's as 'b's.
  (X ↙ Y ↘)
  and we are to the left of any remaining a/b
- After that step, we marked off one extra a (↘ X)
  and searching for b to mark off.

Exercise: TM (state diagram) implementing this?

Exercise: Construct a TM decider for checking if input is power of 2:

- in unary: $\{1^{2^n} \mid n \geq 0\}$ over $\Sigma = \{1\}$

- in binary: $\{x \in \{0,1\}^* \mid x = 2^n, n \geq 0 \text{ in binary}\}$

Note: binary one is just $= \{10^n \mid n \geq 0\}$ - regular!

unary version is non-regular.

TM can try to repeatedly divide # of 1's by two...

(How?)

Def: An input-output TM is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{halt})$, Computation is same as regular TM (start with $q_0 w$ configuration for input w) but if/when switch to q_{halt}, halt and the output is what's on tape (upto ∞-many blanks)

Def: A function $f: \Sigma^* \to \Sigma^*$ is computable if there exists an input output TM M s.t. $\forall w \in \Sigma^*$, M on input w halts with $f(w)$ written on tape.

We will use this to construct helper functions we can call when designing algorithms / TMs.

Examples of Computable functions: → Exercise: Prove this (construct TM) for a couple of these

- on input n in unary, output $n+1$

  i.e. for $\Sigma = \{1\}$, on input 1^n, output 1^{n+1}

- On input n in binary output $n+1$

  ie for $\Sigma = \{0,1\}$ on input x, output $x+1$ (in binary)

- on input $w \in \{0, \ldots, 9\}^*$ output $\#w$

  (i.e., shift input to right and insert #)

- addition in unary:

  on input $1^n \# 1^m$ output $\# 1^{n+m}$

- addition in binary:

  on input $x \# y$ output $\# x+y$ (in binary)

- multiplication in unary or binary

  on input $1^n \# 1^m$ ← ↙ on input $x \# y$

  output $\# 1^{n \cdot m}$ output $x \cdot y$ (in binary)

- On input $w_1 \# w_2$

  output $w_1 \# w_2 \# w_2$ (can be useful) eg. for unary multiplication)

⋮

Next: variants of TM that seem stronger/offer more flexibility, but turn out to be equivalent!

Def: We say two machines M_1, M_2 are **equivalent** if they behave the same on every input, that is: $\forall w \in \Sigma^*$

M_1 accepts $w \iff M_2$ accepts w

M_1 rejects $w \iff M_2$ rejects w

M_1 runs forever on $w \iff M_2$ runs forever on w

Two models of computation are **equivalent** if for $\exists$ machine in one model $\iff \exists$ equivalent machine in the other

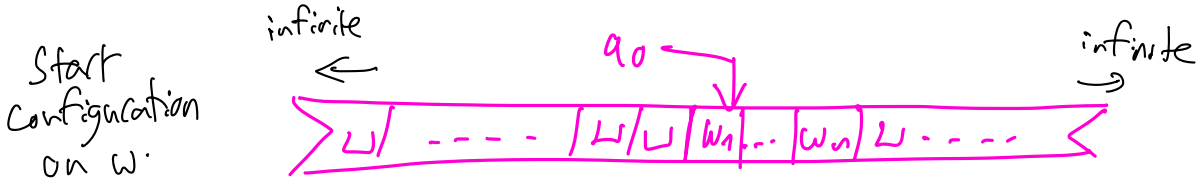
To show models equiv, need to show how to simulate a machine in one with other, and vice versa.

Here are some equivalent variants of TM:

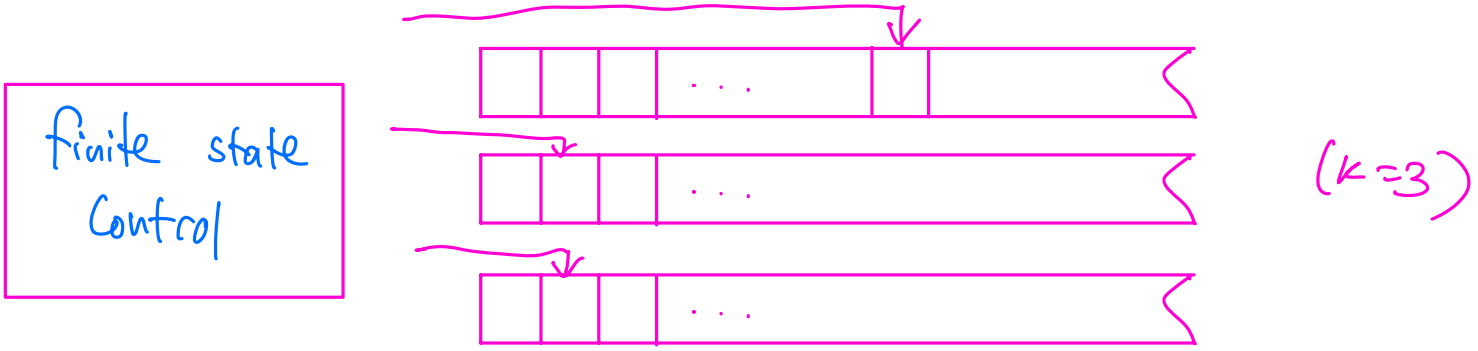
(1) TM where head can move right, left, or stay put

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

(2) A TM with a doubly-infinite tape



(3) Multi-tape TM: has a fixed k number of tapes and read/write heads:



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

initial configuration:

- heads all point to leftmost cell on their tape
- input written on first tape
- $\sqcup$ everywhere else

In class we discussed how to simulate/prove equivalence for (1), (2) with some details left as exercise. Here is what we said with details filled in after class.

you don't have to know all these details, just the main point!

(1) TM where head can move right, left, or stay put

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

→ given a standard TM, obviously can simulate with "stay-put" TM (special case: just run normally).

→ given a "stay-put" TM, how to simulate with standard one?

- transitions with R or L don't change.
- transition with C → move right then left:

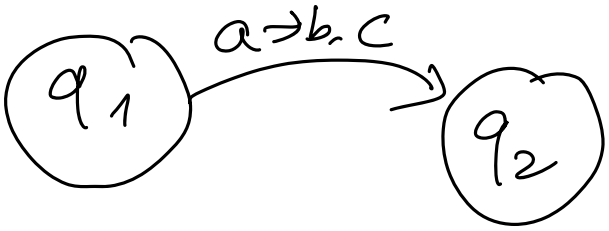
for any $(q_1, a) \to (q_2, b, C)$

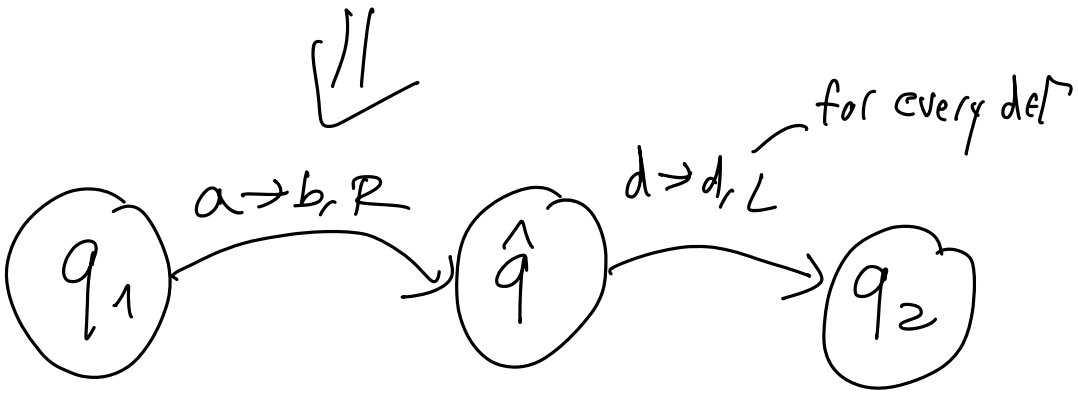
make a new state $\hat{q}$ and add

transitions $(q_1, a) \to (\hat{q}, b, R)$

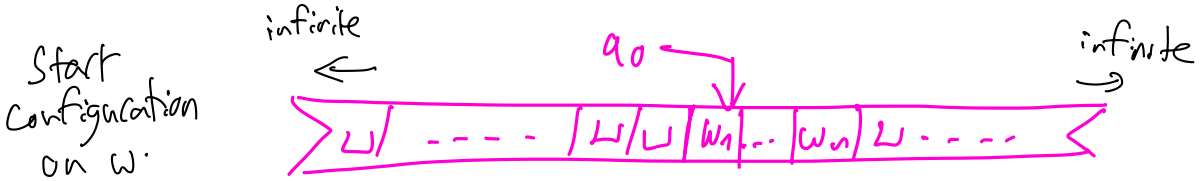
$\forall d \in \Gamma$: $(\hat{q}, d) \to (q_2, d, L)$

i.e: every





(2) A TM with a doubly-infinite tape

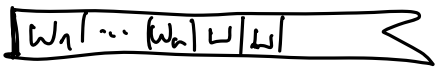


→ given a standard TM, how to simulate with 2-sided TM?

Idea: just add a special symbol to the left of input and whenever you hit it go back right (to "first" cell)

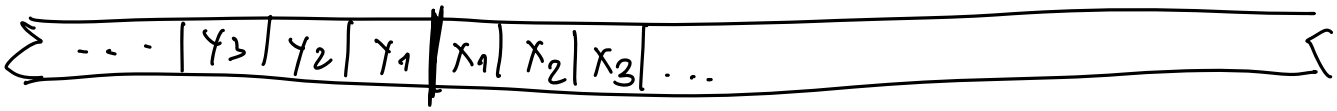


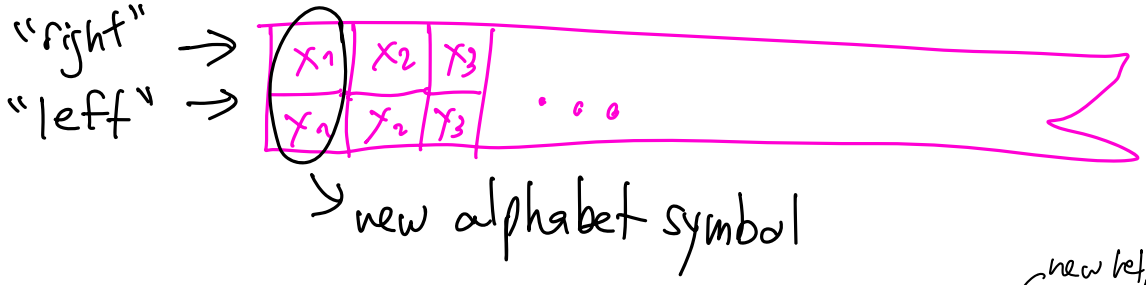
everything else remains the same

This behaves same as 

→ given a doubly infinite TM, how to simulate with standard TM:

imagine "folding" doubly infinite tape around initial location of read head:





formally, our new tape alphabet will be $\Sigma \cup \Gamma^2 \cup \{\sqcup, \#\}$ (new helper symbol to mark beginning of tape)

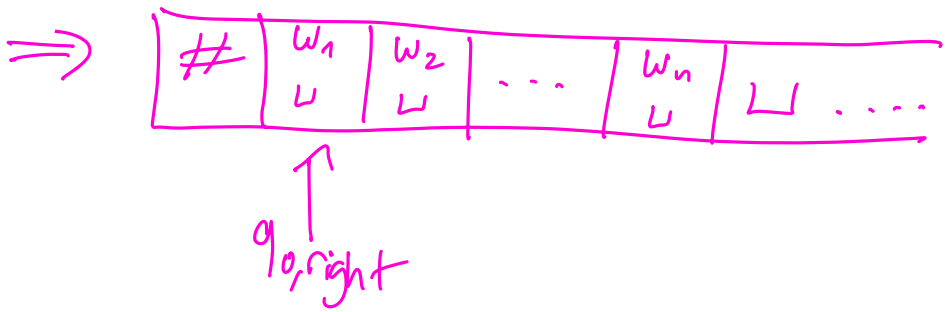
- new states: two versions of each state $q \to q_{right}, q_{left}$

one for "left" and one for "right":

↳ focus on bottom half of symbol, head moves in opposite direction.

↳ focus on top half of symbol, head moves in same direction as before.

- start by changing tape from $w_1 w_2 \ldots w_n \sqcup \ldots$ to



change δ to δ' that looks at correct part of symbol (depending if at $q_{i,right}$ or $q_{i,left}$) and adjust transitions

for example if two-sided tape had $\delta(q_7, a) = (q_8, b, R)$ we add the rules:

$$\forall c \in \Sigma, \quad \delta'(q_{7,right}, \begin{pmatrix} a \\ c \end{pmatrix}) = (q_{8,right}, \begin{pmatrix} b \\ c \end{pmatrix}, R)$$

$$\forall c \in \Sigma, \quad \delta'(q_{7,left}, \begin{pmatrix} c \\ a \end{pmatrix}) = (q_{8,left}, \begin{pmatrix} c \\ b \end{pmatrix}, L)$$

Also take care to switch between 'right' and 'left' when reach #: (meaning in 2-sided tape we moved from one half to the other)

$$\delta'(q_{i,right}, \#) = (q_{i,left}, \#, R) \quad \text{(for all states)}$$

$$\delta'(q_{i,left}, \#) = (q_{i,right}, \#, R)$$

(3) Multi-tape TM: has a fixed k number of tapes and read/write heads:

We will show equivalence next time)

now, a couple of exercise solutions filled after class:

Solutions to two of the exercises: (let me know if you find any errors!)

- **L has a DFA ⇒ L has a TM:**

(1) Write this construction formally:

Sol: Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$, construct a TM M that simulates it, as follows:

$M = (Q, \Sigma, \delta', \Gamma, q_0, q_{acc}, q_{rej})$ where δ' is defined as follows:

(Q: same as D; Γ: $\Sigma \cup \{\sqcup\}$; q_0: same as D; q_{acc}, q_{rej}: new states)

δ':

same state control as DFA; just read (=rewrite with same); just go right

$$\forall a \in \Sigma \quad \forall q \in Q \quad \delta'(q,a) = (\delta(q,a), a, R)$$



$$\forall q \in F \quad \delta'(q,\sqcup) = (q_{acc}, \sqcup, L)$$

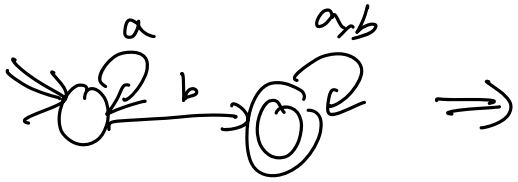


$$\forall q \in Q \setminus F \quad \delta'(q,\sqcup) = (q_{rej}, \sqcup, L)$$



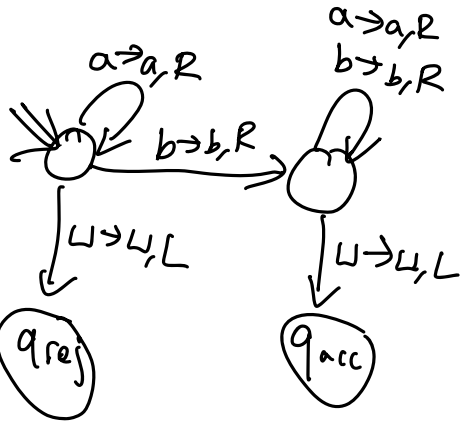
(2) Draw the resulting decider for $\{w \in \{a,b\}^* \mid w \text{ contain a 'b'}\}$

DFA:



⇒

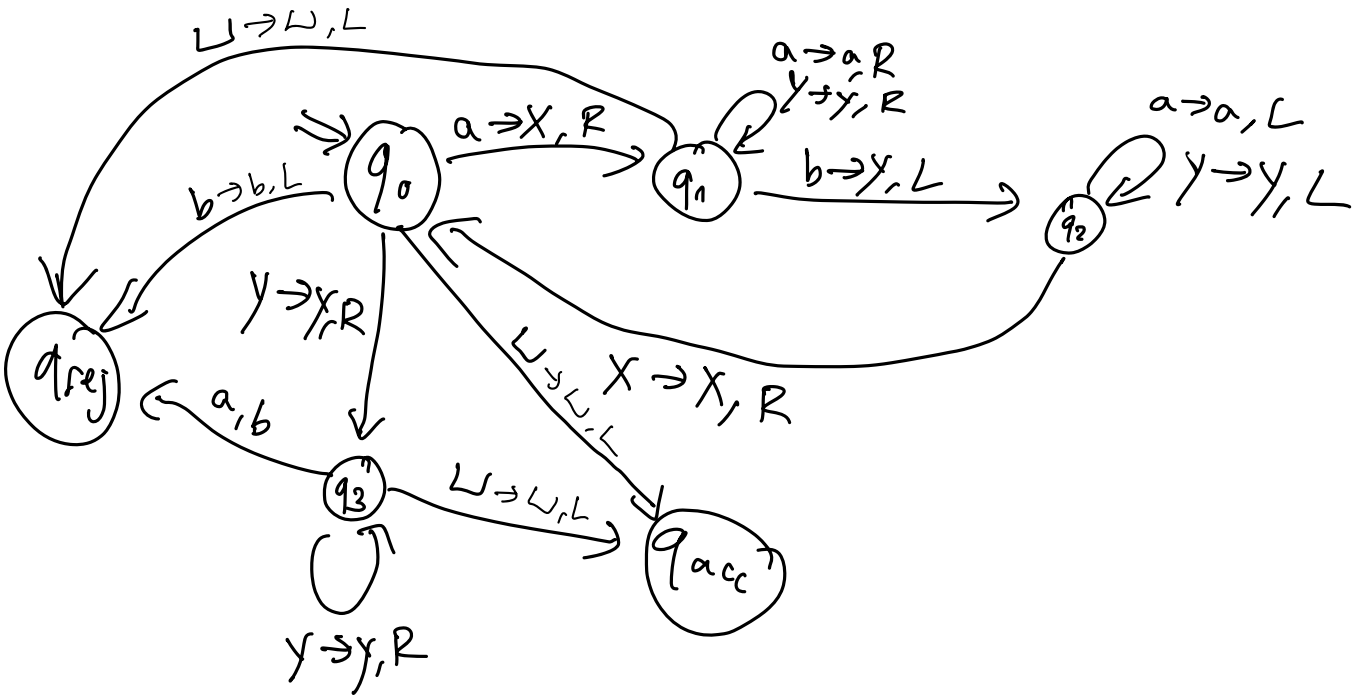
TM:



Aside: does this TM implement the same algorithm as the TM's we saw for this language last lecture?

• state diagram for TM deciding $\{a^n b^n \mid n \geq 0\}$:

Implementing second algorithm (first one similar-ish):



q_0: marked same # of a's and b's (all X's to my left, all a/b/y to my right.

q_1: marked one more a than b (going right searching 'b')

q_2: marked same # of a's and b's (going left to X)

q_3: string had a prefix $a^n b^n$ $(n \geq 1)$ which we finished marking off (it's now $X^n y^n$) and we're going right to the end of it (accept iff nothing more after).