Lecture 11:

* Quiz 1: grades out (regrade soon) ← solution sketch
* Hw 2: due yesterday
* Handout 2
* Challenge 2 (candy AND sticker)

Last time: Turing machines

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

Q: set of states

Σ: input alphat

Γ: tape alphabet (contains Σ, $\sqcup$)

δ: transition function : $Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

q_0 = start state

q_{acc}, q_{rej} = accept / reject states

when we enter q_{start} / q_{rej} => HALT

On an input w: a TM can:

- accept
- reject
- run forever

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

M is a **decider** if on all inputs w: M halts

not all TMs are deciders!

L is **TM-recognizable** if $\exists$ a TM M with $L(M) = L$

If $x \in L \Rightarrow M$ accepts x
If $x \notin L \Rightarrow M$ does not accept x

L is **TM-decidable** if $\exists$ a TM M with $L(M) = L$ and M is a decider

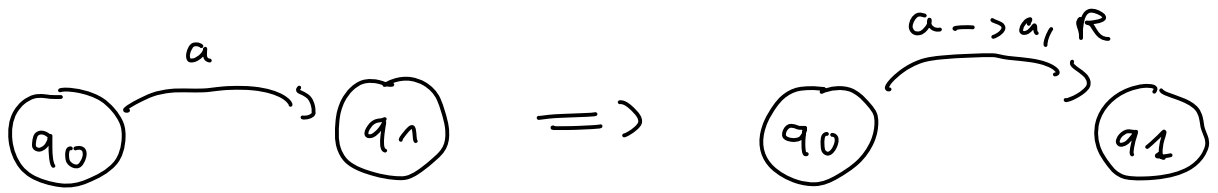
If $x \in L \Rightarrow M$ accepts x
If $x \notin L \Rightarrow M$ rejects x

Regular $\subseteq$ decidable $\subseteq$ Recognizable $\subseteq$ all languages

Regular languages — why ? — decidable languages — by definition — Recognizable languages — by def — all languages

We can implement a DFA using a TM

=> just go right, never write anything



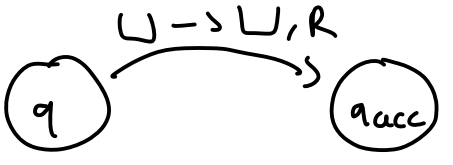
when we see the first ␣ we accept / reject

based on the DFA:

If q was reject



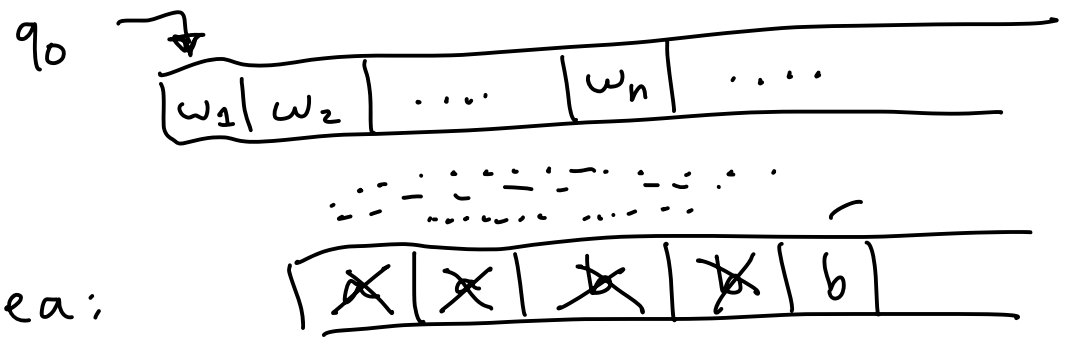
If q was accept in the DFA



TMs can also decide non-regular languages

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

TM decider for L?



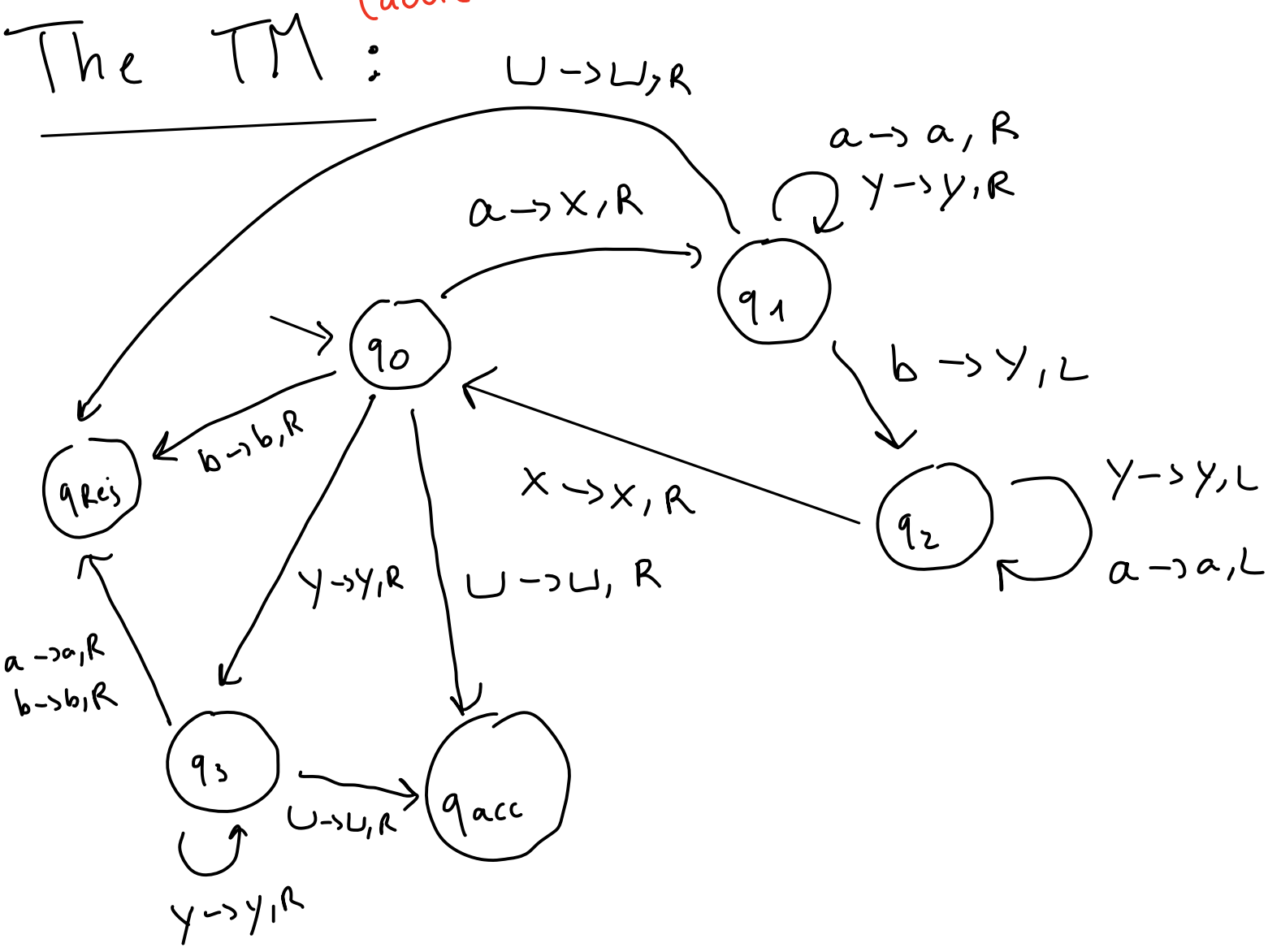
Idea:

Replace the first a with X, go right to the first b:

- If no b ~> reject
- Else replace b with Y
  and go left to see an X
- If there's still unmarked a's
  => repeat

Else => go right to check there's no b's

The TM: (added after class)



Ex: write a TM for $\{1^{2^n} \mid n \geq 0\}$ (→ unary)

over $\Sigma = \{1\}$

- write a TM for $\{x \in \{0,1\}^* \mid$ $x = 2^n, n \geq 0$ in binary$\}$

(→ regular)

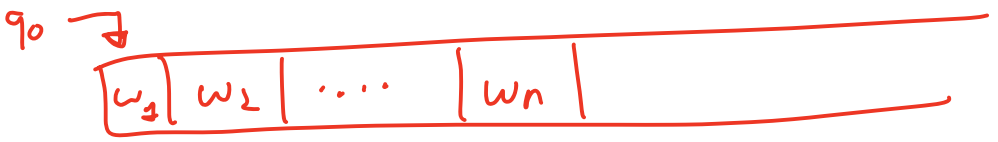
(→ 1, 10, 100, 1000)

2^0 2^1 2^2 2^3

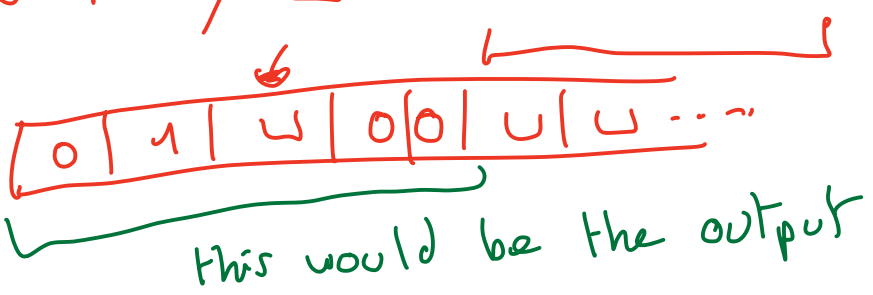
Def: an input-output TM is a 6 tuple

$(Q, \Sigma, \Gamma, \delta, q_0, q_{halt})$

The TM starts with the input w on the tape, followed by ∞ many blanks



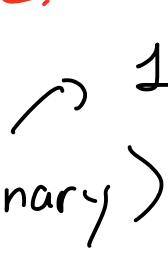
- If/when we enter q_{halt}, the TM stop and output whatever is on the tape followed by ∞ many ␣



We will use such TMs to build helper functions/ subroutines when constructing algos / TMs

Def: a function $f: \Sigma^* \to \Sigma^*$ is TM computable if there exists an input-output TM, M such that on all inputs x M halts with f(x) written on the tape.

Example: On input n (in unary) output n+1 in unary



On input n in binary, output n+1 in binary

↳ 1^{n+1}

- On input $w \in \{0,1,\dots,9\}$ output $\#w$
- On input $w \in \{0,1,\dots,9\}$ output $w\#w$
- On input $1^n \# 1^m \Rightarrow$ output $\# 1^{n+m}$
- Same thing in binary
- Multiplication: $1^n \# 1^m \Rightarrow$ output $\# 1^{n*m}$

$x \# y \Rightarrow$ output $\# x*y$

↑ binary

We defined TMs in a very narrow way

(pro) it makes it easier to prove TMs can't solve certain problems

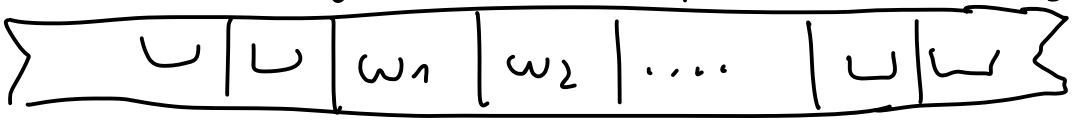
(con) they are hard to work with

We'll show a few equivalent models of TMs
⇒ easier to work with

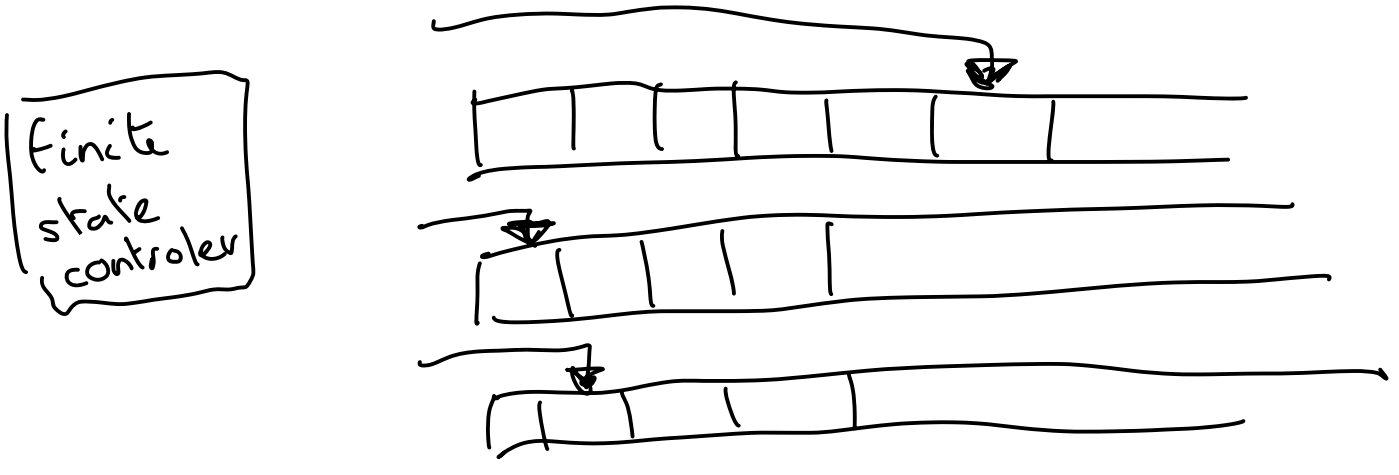
(1) TMs that can "stay put"

the TM can move its head L/R or
stay in the same cell

(2) The tape is ∞ to the right but
not the left: doubly infinite TMs
← infinite to the left and right →



(3) Multitape TMs: a TM with $k \geq 1$
tapes, one head for each tape



Def: We say 2 TMs, M_1 and M_2 are equivalent if they behave the same on every input:

- 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 $\forall$ machine in the first model there exists an equivalent machine in the second and vice versa.

(1) Stay put TMs: $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R, C\}$

only how δ is defined is different.

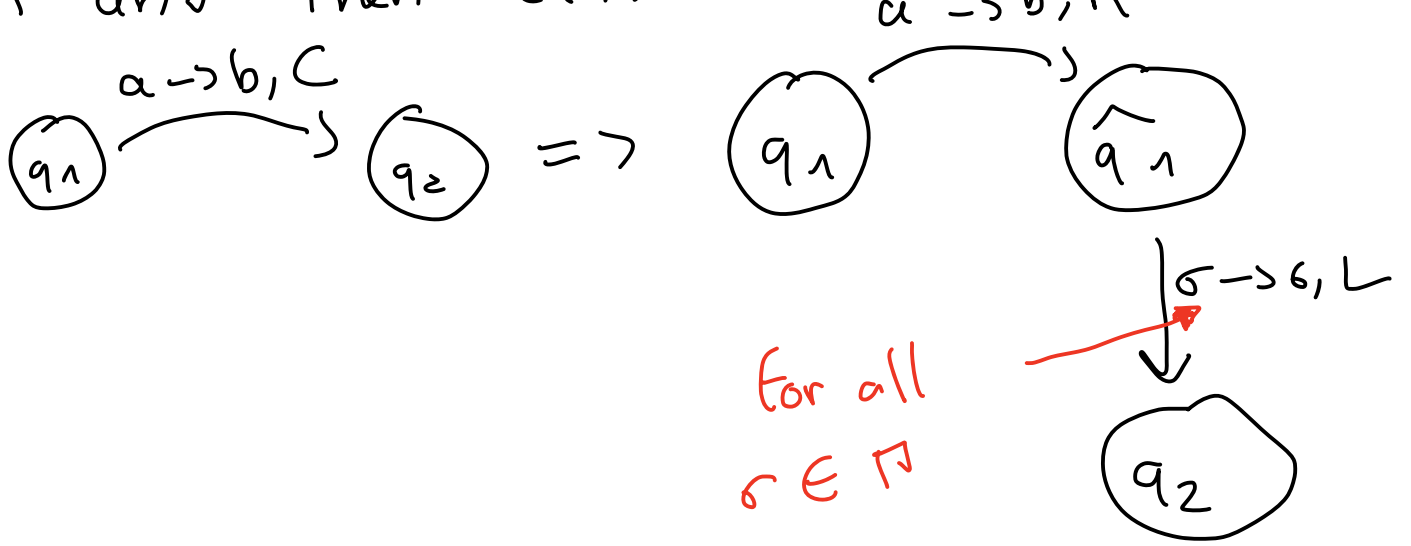
↑ stay in same cell

→ given a standard TM, how to simulate it with a "stay put" TM? By definition

Run the TM normally.

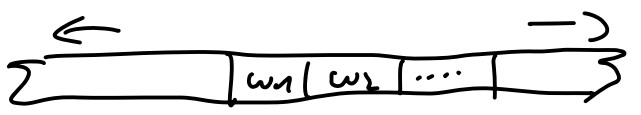
→ given a "stay put" TM, how to simulate it with a standard TM?

For each transition that "stays put", go right and then left.



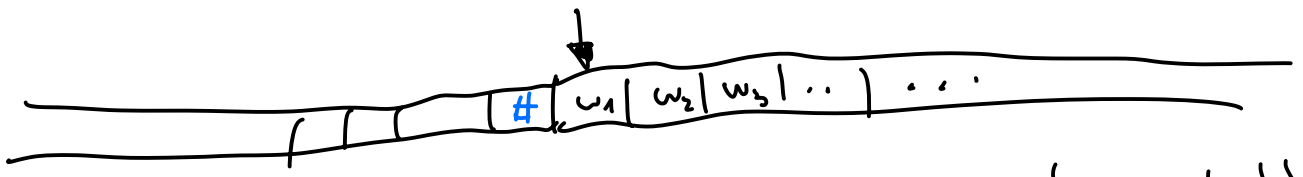
One new state for each transition with a C

- Doubly infinite TMs :



- Given a standard TM, how to simulate it with a doubly ∞ TM?

* we don't change anything



- initially add # (a new special symbol) to the left of the input
- whenever we move left => if we read the # we immediately go right

(everything else remains the same

Other direction: next time!!