

# **Non-Context-Free Languages, and Intro to Turing Machines**



# Announcements

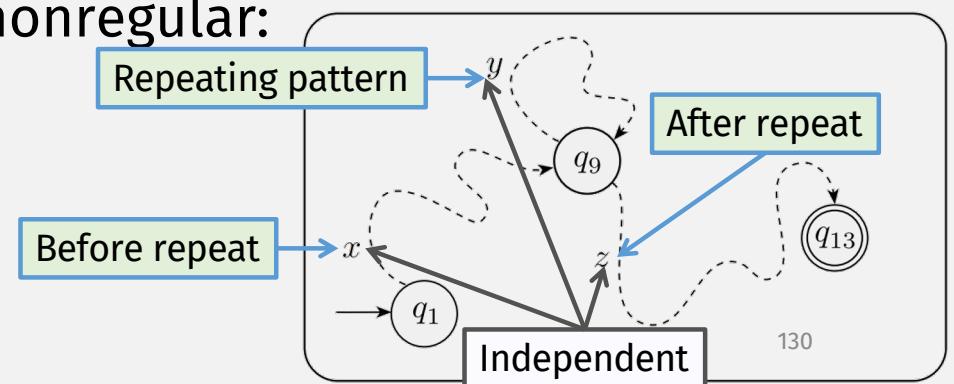
- HW 6 released
  - Due Fri 11/5 10:00pm
- Time + 
- Also, IMHO pretty easy

Next:

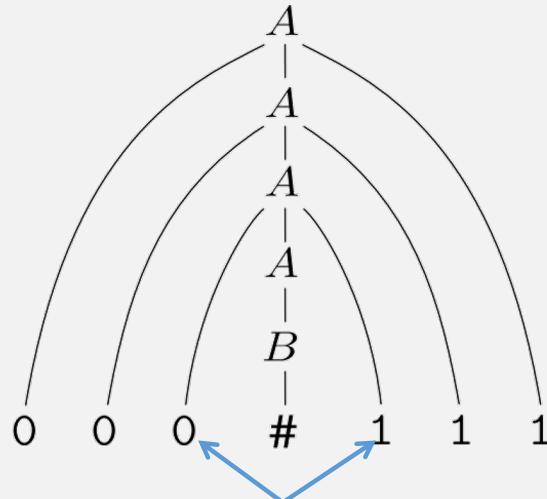
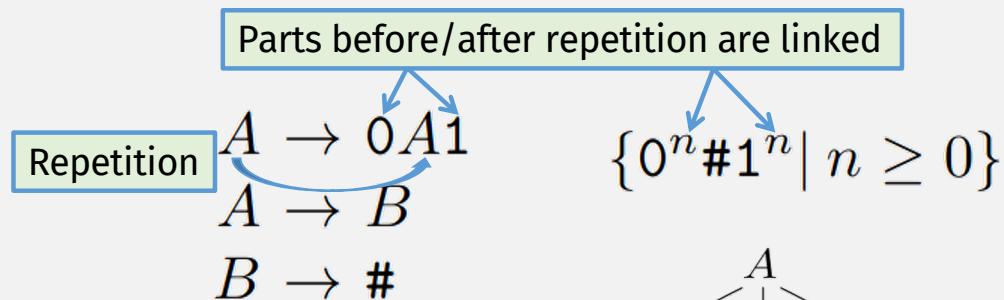


## Flashback: Pumping Lemma for Reg Langs

- The Pumping Lemma describes how strings repeat
- Strs in a regular lang can (only) repeat using Kleene pattern
  - Before/during/after parts are independent!
- Langs with dependencies are nonregular:
  - E.g.,  $\{0^n 1^n \mid n \geq 0\}$
- Today: How do CFLs repeat?



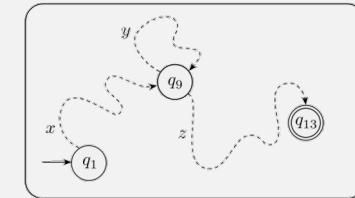
# Repetition and Dependency in CFLs



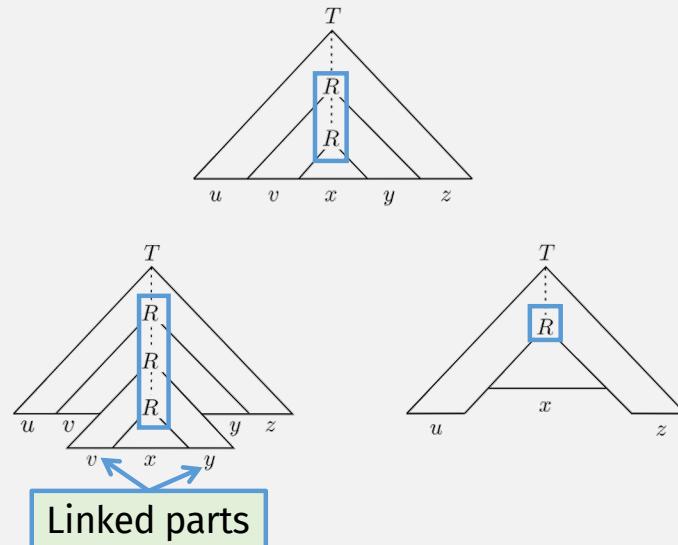
Parts before/after repetition are linked

# How Can Strings in CFLs Repeat?

- Strings in regular languages repeat states



- Strings in CFLs repeat subtrees in the parse tree

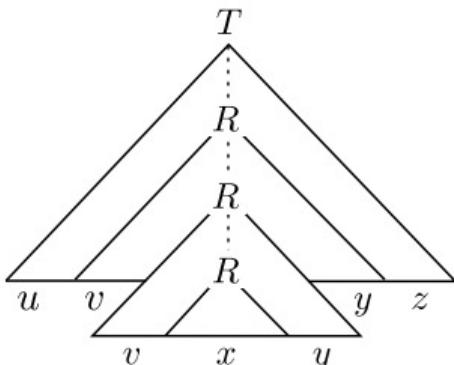


# Pumping

**Pumping lemma** If  $A$  is a context-free language, then there is a number  $p$  (the pumping length) where, if  $s$  is any string in  $A$  of length at least  $p$ , then  $s$  may be divided into five pieces  $s = uvxyz$  satisfying the conditions

Now there are two pumpable parts.  
But they must be pumped together!

1. for each  $i \geq 0$ ,  $uv^i xy^i z \in A$ ,
2.  $|vy| > 0$ , and
3.  $|vxy| \leq p$ .



# CFLS

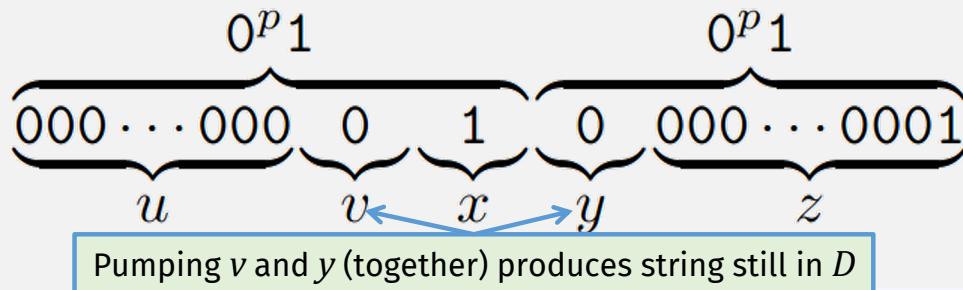
**languages** If  $A$  is a context-free language, then there is a number  $p$  (the pumping length) where, if  $s$  is any string in  $A$  of length at least  $p$ , then  $s$  may be divided into five pieces  $s = uvxyz$  satisfying the following conditions:

**Pumping lemma** If  $A$  is a regular language, then there is a number  $p$  (the pumping length) where if  $s$  is any string in  $A$  of length at least  $p$ , then  $s$  may be divided into three pieces,  $s = xyz$ , satisfying the following conditions:

1. for each  $i \geq 0$ ,  $xy^i z \in A$ ,
2.  $|y| > 0$ , and
3.  $|xy| \leq p$ .

Non CFL example:  $D = \{ww \mid w \in \{0,1\}^*\}$

- Previous: Showed  $D$  is nonregular w. unpumpable string  $s$ :  $0^p 1 0^p 1$
- Now: this  $s$  **can** be pumped according to CFL pumping lemma:



- CFL Pumping Lemma conditions:
  - 1. for each  $i \geq 0$ ,  $uv^i xy^i z \in A$ ,
  - 2.  $|vy| > 0$ , and
  - 3.  $|vxy| \leq p$ .

This doesn't prove that the language is a CFL!  
It only means the counterexample doesn't work to prove it's non context-free.

Non CFL example:  $D = \{ww \mid w \in \{0,1\}^*\}$

- Choose another string  $s$ :

If  $vyx$  is contained in first or second half, then  
any pumping will break the match  $\times$

$\overbrace{0^p} \overbrace{1^p} \overbrace{0^p} \overbrace{1^p}$

So  $vyx$  must straddle the middle  $\times$   
But any pumping still breaks the match because order is wrong

- CFL Pumping Lemma conditions:
  1. for each  $i \geq 0$ ,  $uv^i xy^i z \in A$ ,
  2.  $|vy| > 0$ , and
  3.  $|vxy| \leq p$ .

Non CFL example:  $D = \{ww \mid w \in \{0,1\}^*\}$

- Previously: Showed  $D$  is not regular
- Just Now:  $D$  is not context-free either!

# XML Again ...

- We previously said XML sort of looks like the CFL:  $\{0^n 1^n \mid n \geq 0\}$ 
  - ELEMENT → <TAG>CONTENT</TAG>
  - Where TAG is any string
- But these arbitrary TAG strings must match!
- So XML also looks like this non-CFL:  $D = \{ww \mid w \in \{0,1\}^*\}$
- This means XML is not context-free!
  - Note: HTML is context-free because ...
  - ... there are only a finite number of tags,
  - so they can be embedded into a finite number of rules.
- In practice:
  - XML is parsed as a CFL, with a CFG
  - Then matching tags checked in a 2<sup>nd</sup> pass with a more powerful machine ...

## A More Powerful Machine ... Let's say ...

Can move to arbitrary memory locations, and read/write to it

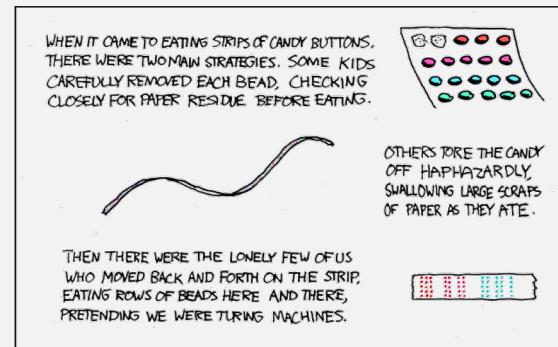
$M_1$  accepts its input if it is in language:  $B = \{w\#w \mid w \in \{0,1\}^*\}$

$M_1$  = “On input string  $w$ :

Infinite memory, initially starts with input

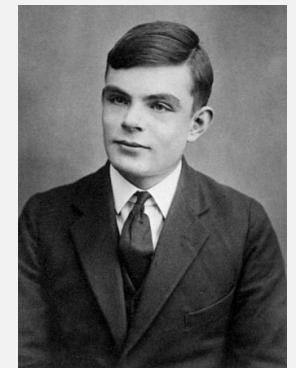
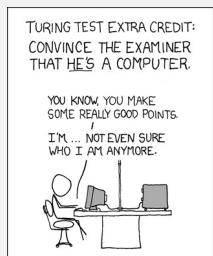
1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.

# Turing Machines (TMs)



# Alan Turing

- (One of the) first to formalize models of computation that we are studying
- Worked as codebreaker during WW2
  - Credited for saving hundreds of thousands of lives
- Also studied AI
  - Turing Test





Jason Screamin' 🎃

@jhemann

...

I just wanted to show a synchronous controller. Cold,  
emacs.

A screenshot of a terminal window titled "doctor". The window contains a conversation between a user and a bot named "doctor". The user types messages, and the bot responds with a question or comment. The user's messages are in white text on a black background, while the bot's responses are in blue text.

```
I am the psychotherapist. Please, describe your problems. Each time
you are finished talking, type RET twice.

I'm having a problem with my code.

Why do you say you are having a problem with your code?

My program isn't running correctly.

Is it because of your plans that you say your program is not running
correctly?

I had hoped to be finished with my homework by this evening so I could
go on a run.

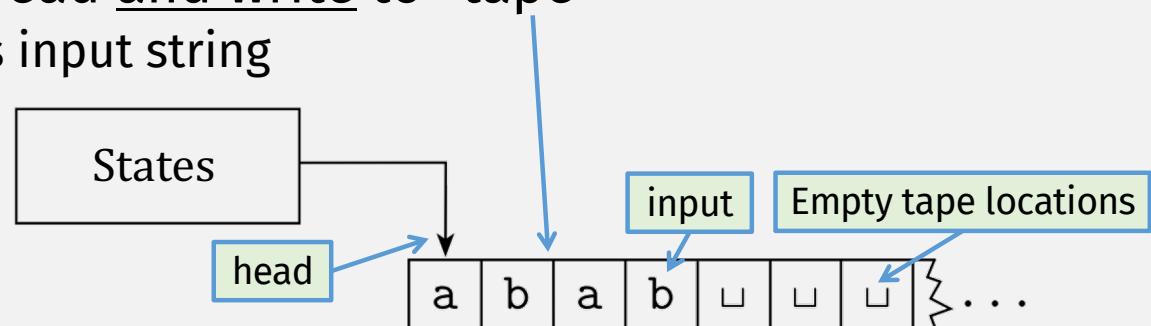
Maybe your life has something to do with this.
```

12:16 PM · May 13, 2020 · Twitter Web App

# Other Automata vs Turing Machines

- Turing Machines can read and write to “tape”
  - Tape initially contains input string

- The tape is infinite



- Each step: “head” can move left or right

- A Turing Machine can accept/reject at any time

---

**DEFINITION 3.5**

Call a language **Turing-recognizable** if some Turing machine recognizes it.

# Turing Machine Example

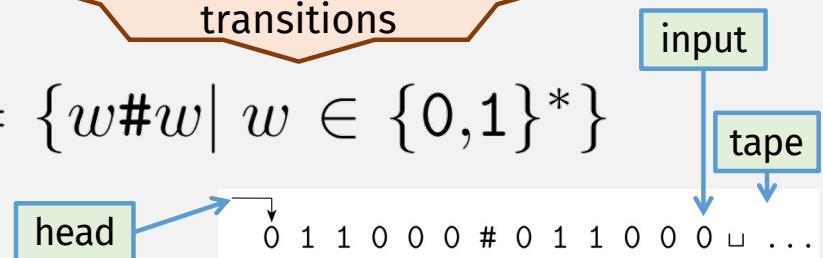
This is an informal  
TM description.  
One step = multiple  
transitions

$M_1$  accepts inputs in language  $B = \{w\#w \mid w \in \{0,1\}^*\}$

$M_1$  = “On input string  $w$ :

1. Zig-zag across the tape to corresponding positions on either side of the  $\#$  symbol to check whether these positions contain the same symbol. If they do not, or if no  $\#$  is found, *reject*.  
*Cross off symbols* as they are checked to keep track of which symbols correspond.

“Cross off” =  
write “x” char



# Turing Machine Example

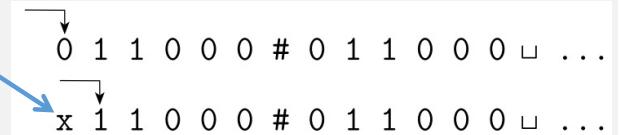
$M_1$  accepts inputs in language  $B = \{w\#w \mid w \in \{0,1\}^*\}$

$M_1$  = “On input string  $w$ :

“Cross off” = write “x” char

1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, *reject*.  
Cross off symbols as they are checked to keep track of which symbols correspond.

“Cross off” =  
write “x” char



# Turing Machine Example

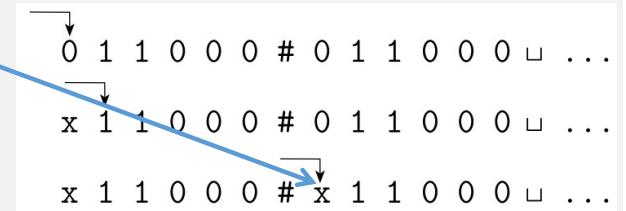
$M_1$  accepts inputs in language  $B = \{w\#w \mid w \in \{0,1\}^*\}$

$M_1$  = “On input string  $w$ :

“Cross off” = write “x” char

1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, *reject*.  
Cross off symbols as they are checked to keep track of which symbols correspond.

“Cross off” =  
write “x” char



# Turing Machine Example

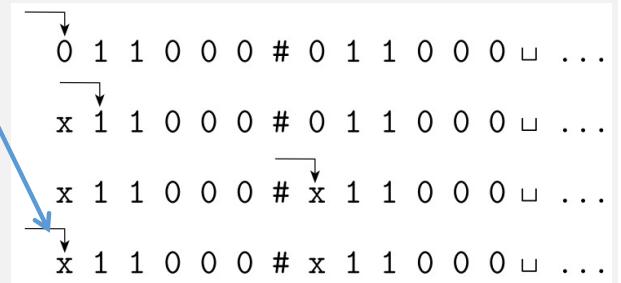
$M_1$  accepts inputs in language  $B = \{w\#w \mid w \in \{0,1\}^*\}$

$M_1$  = “On input string  $w$ :

1. Zig-zag across the tape to corresponding positions on either side of the  $\#$  symbol to check whether these positions contain the same symbol. If they do not, or if no  $\#$  is found, *reject*.  
Cross off symbols as they are checked to keep track of which symbols correspond.

“Cross off” =  
write “x” char

“zag” to start



# Turing Machine Example

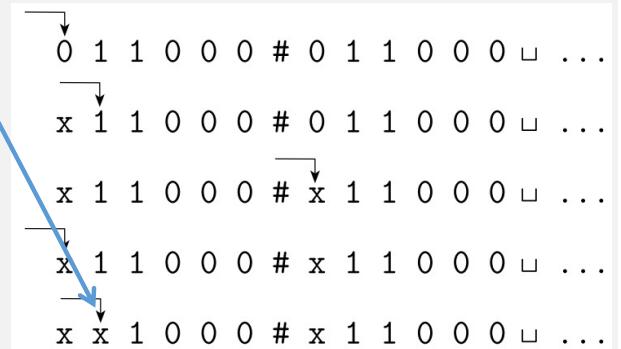
$M_1$  accepts inputs in language  $B = \{w\#w \mid w \in \{0,1\}^*\}$

$M_1$  = “On input string  $w$ :

1. Zig-zag across the tape to corresponding positions on either side of the  $\#$  symbol to check whether these positions contain the same symbol. If they do not, or if no  $\#$  is found, *reject*.  
Cross off symbols as they are checked to keep track of which symbols correspond.

“Cross off” =  
write “x” char

Continue crossing off

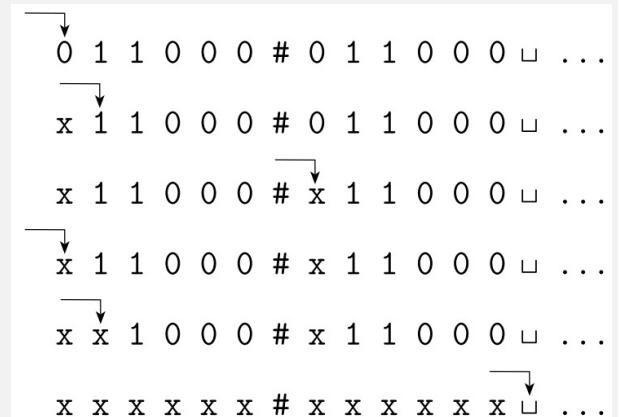


# Turing Machine Example

$M_1$  accepts inputs in language  $B = \{w\#w \mid w \in \{0,1\}^*\}$

$M_1$  = “On input string  $w$ :

1. Zig-zag across the tape to corresponding positions on either side of the  $\#$  symbol to check whether these positions contain the same symbol. If they do not, or if no  $\#$  is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
2. When all symbols to the left of the  $\#$  have been crossed off, check for any remaining symbols to the right of the  $\#$ . If any symbols remain, *reject*; otherwise, *accept*.”

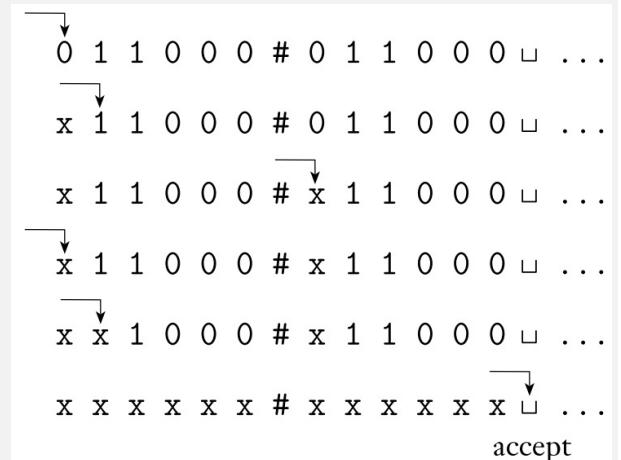


# Turing Machine Example

$M_1$  accepts inputs in language  $B = \{w\#w \mid w \in \{0,1\}^*\}$

$M_1$  = “On input string  $w$ :

1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
  2. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise, *accept*.”



# Turing Machines: Formal Definition

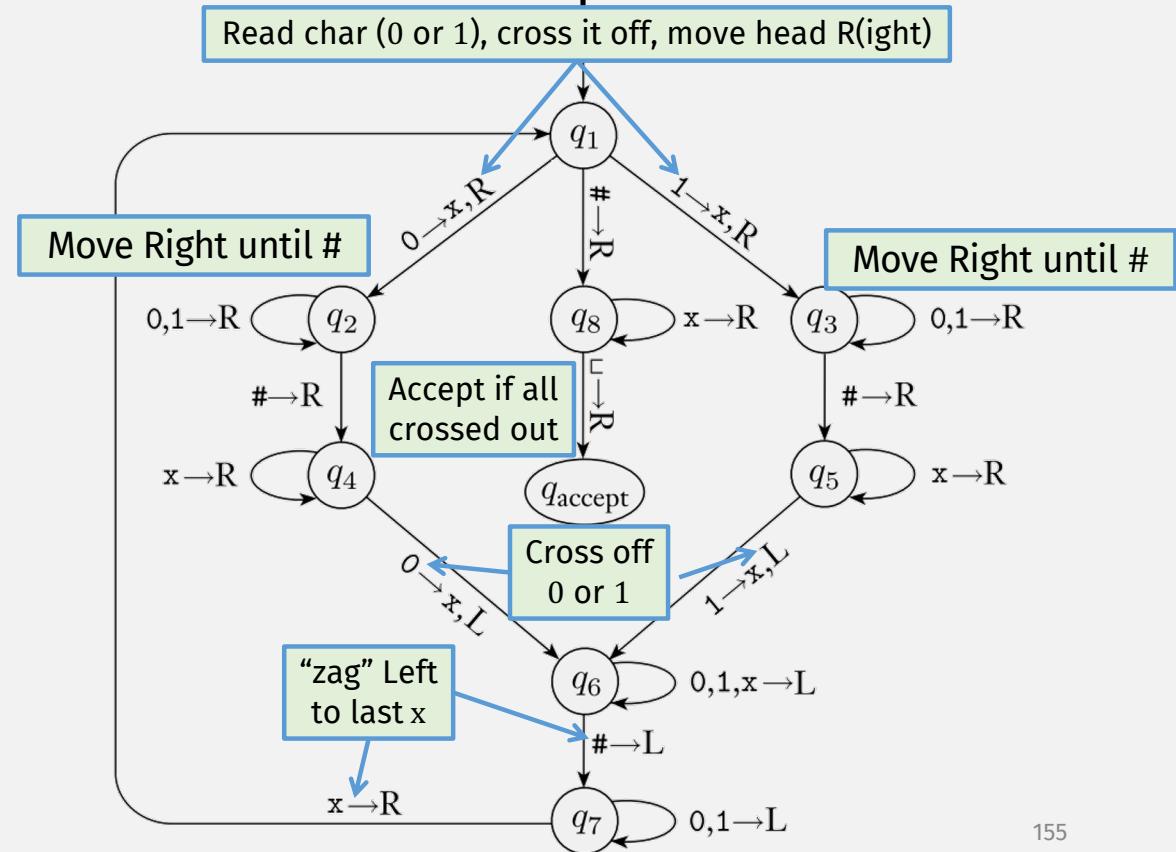
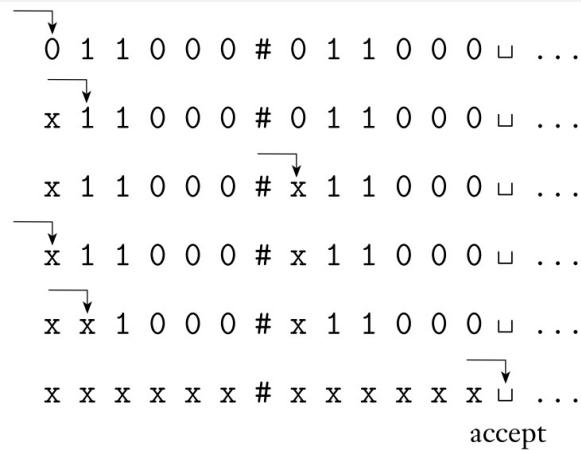
## DEFINITION 3.3

A *Turing machine* is a 7-tuple,  $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ , where  $Q, \Sigma, \Gamma$  are all finite sets and

1.  $Q$  is the set of states,
2.  $\Sigma$  is the input alphabet not containing the **blank symbol**  $\sqcup$ ,
3.  $\Gamma$  is the tape alphabet, where  $\sqcup \in \Gamma$  and  $\Sigma \subseteq \Gamma$ ,
4.  $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$  is the transition function,
5.  $q_0 \in Q$  is the start state,
6.  $q_{\text{accept}} \in Q$  is the accept state, and
7.  $q_{\text{reject}} \in Q$  is the reject state, where  $q_{\text{reject}} \neq q_{\text{accept}}$ .

# Formal Turing Machine Example

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



# Turing Machine: Informal Description

- $M_1$  accepts if input is in language  $B = \{w\#w \mid w \in \{0,1\}^*\}$

$M_1$  = “On input string  $w$ :

1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, if no # is found, reject. Cross off symbols as they are checked off, keeping track of which symbols correspond.
2. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, reject; otherwise, accept.”

We will (mostly) stick to informal descriptions of Turing machines, like this one

# Non-halting Turing Machines (TMs)

IMPORTANT SLIDE

- A DFA, NFA, or PDA\* always halts
  - Because the (finite) input is always read exactly once
- But a Turing Machine can run forever
  - E.g., the head can move back and forth in a loop
- Thus, there are two classes of Turing Machines:
  - A recognizer is a Turing Machine that may run forever
  - A decider is a Turing Machine that always halts.

\* For a CFL  
there **\*is\*** a

DEFINITION 3.5

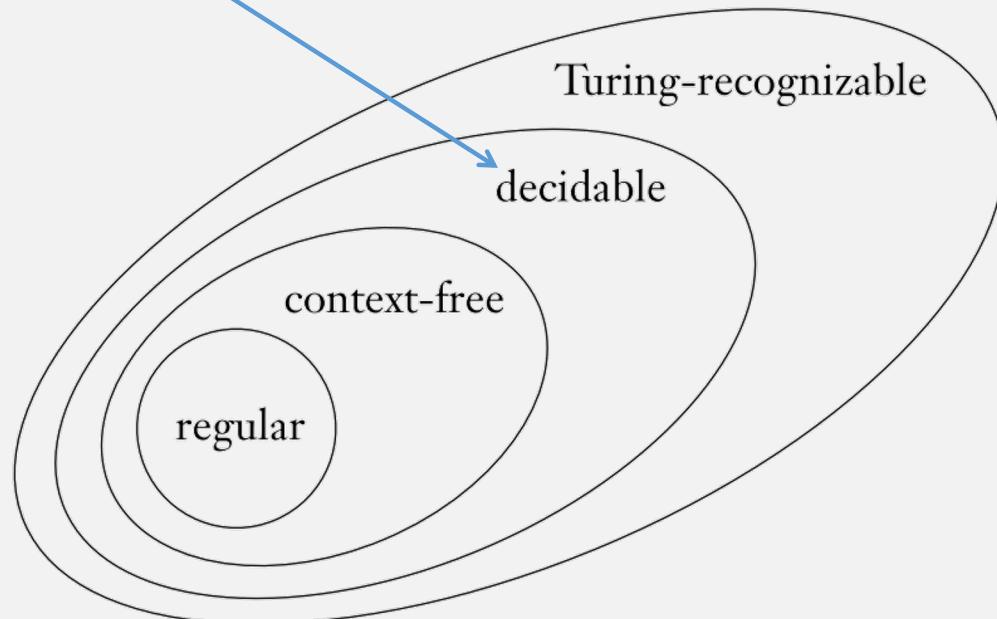
Call a language **Turing-recognizable** if some Turing machine recognizes it.

DEFINITION 3.6

Call a language **Turing-decidable** or simply **decidable** if some Turing machine decides it.

# Formal Definition of an “Algorithm”

- An algorithm is equivalent to a Turing-decidable Language



# **Check-in Quiz**

On Gradescope