Non-Context-Free Languages, <u>and</u> Intro to Turing Machines

Wednesday, March 10, 2021

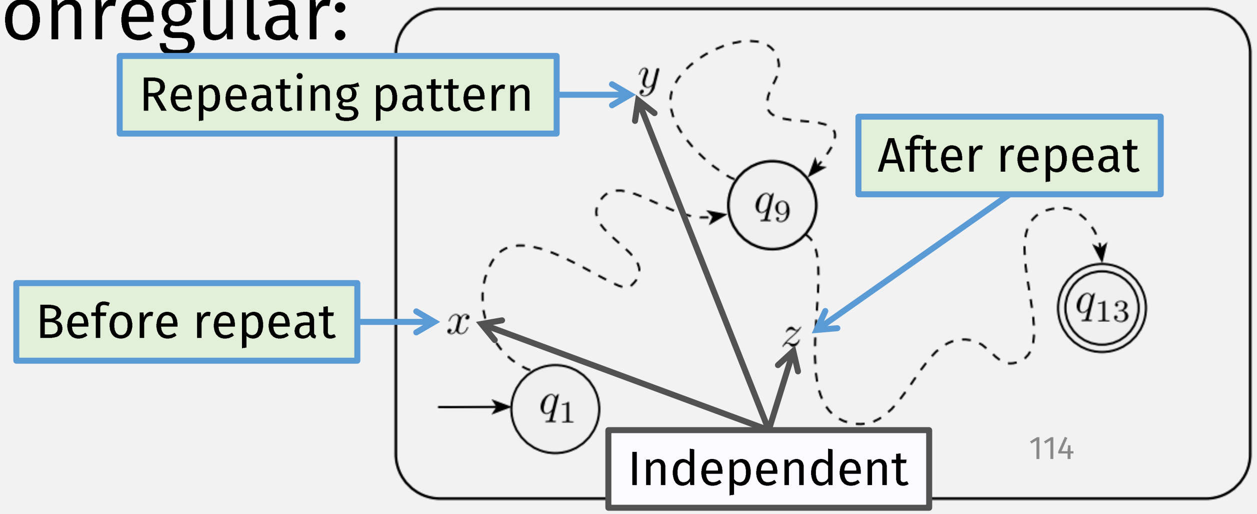
Announcements

- Reminder: no class next week (Spring Break)
  - 3/15 – 3/19
- HW 5 due tonight
  - 11:59pm EST
- HW 6 released
  - Due Sun 3/28 11:59pm EST (after break)

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., $\{0^n 1^n | \ n \geq 0\}$
- Today: How do CFLs repeat?



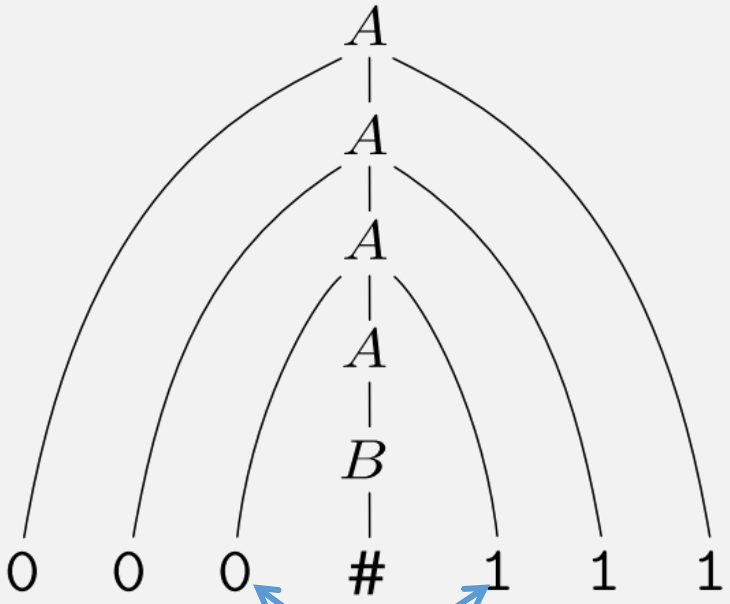
Repetition and Dependency in CFLs

Parts before/after repetition are linked

Repetition

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

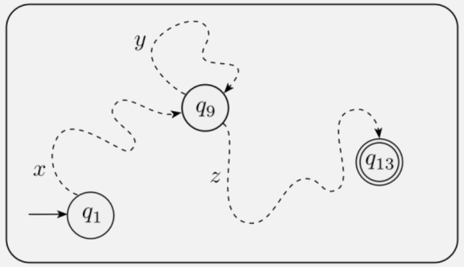
$$\{0^n\#1^n \mid n \geq 0\}$$



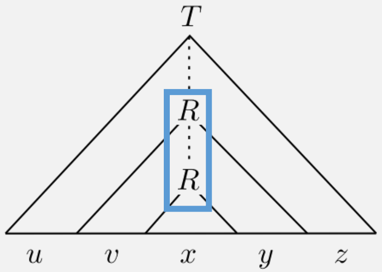
Parts before/after repetition are linked

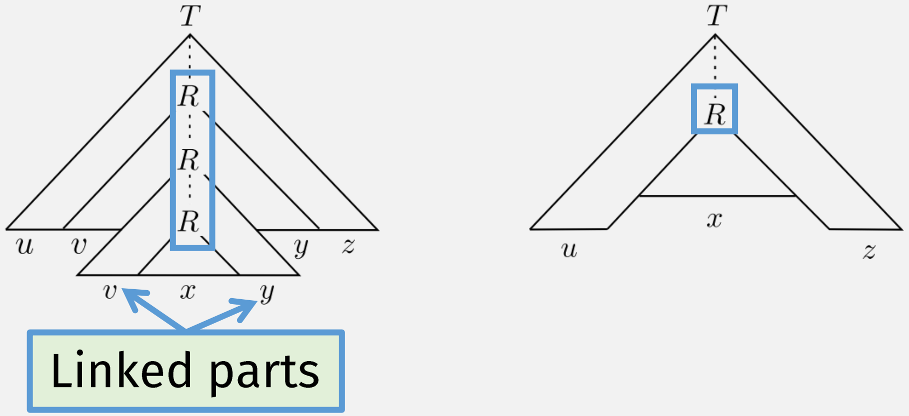
How Can Strings in CFLs Repeat?

- Strings in regular languages repeat states

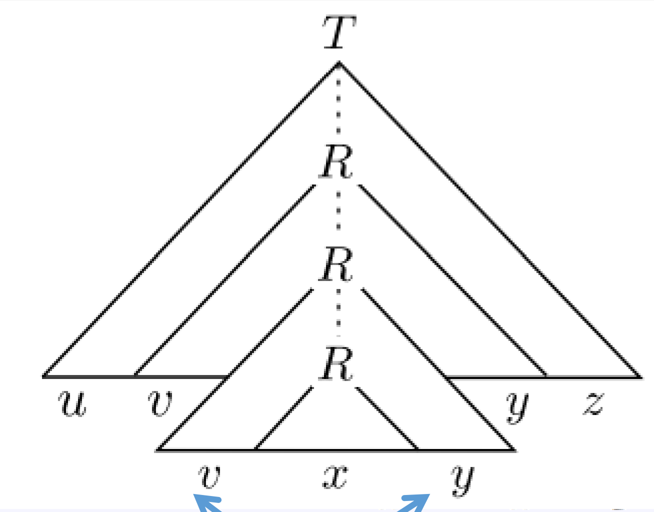


- Strings in CFLs <u>repeat subtrees</u> in the parse tree





Pumping … CFLS



Pumping lemma … guages If A is a context-free language, then there is a nu … 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 $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

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 $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

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

- <u>Previous</u>: Showed D is <u>nonregular</u> w. unpumpable string s: 0^p10^p1
- <u>Now</u>: this s **can** be pumped according to <u>CFL pumping lemma</u>:

$$\overbrace{\underbrace{000\cdots000}_{u}\;\underbrace{0}_{v}\;\underbrace{1}_{x}}^{0^p1}\;\overbrace{\underbrace{0}_{y}\;\underbrace{000\cdots0001}_{z}}^{0^p1}$$

Pumping v and y (together) produces string still in D

- CFL Pumping Lemma conditions:
  - ☑ **1.** for each $i \geq 0$, $uv^ixy^iz \in A$,
  - ☑ **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

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

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

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

- CFL Pumping Lemma conditions:

1. for each $i \geq 0$, $uv^ixy^iz \in A$,
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: $\{0^n 1^n | \, n \geq 0\}$
  - ELEMENT → <TAG>CONTENT</TAG>
  - Where TAG is any string

But these arbitrary TAG strings must match!

- So XML also looks like this non-CFL: $D = \{ww | \; w \in \{0,1\}^*\}$

- 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 2nd pass with a more powerful machine …

A More Powerful Machine …

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

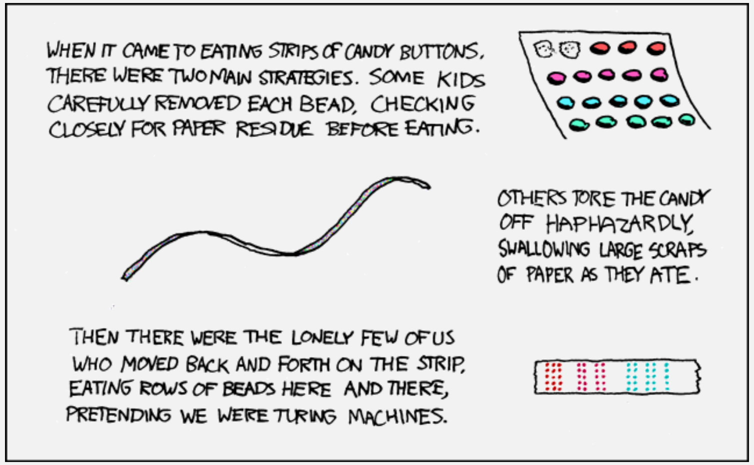
M_1 accepts its input if it is in language: $B = \{w\#w \mid w \in \{0,1\}^*\}$

Infinite memory, initially starts with input

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.

Turing Machines (TMs)

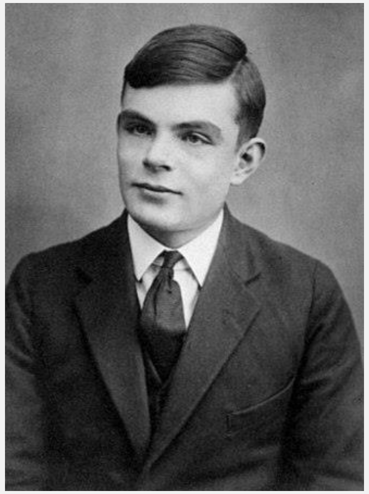


Alan Turing

- First to formalize models of computation that we are studying
  - I.e., he invented this course

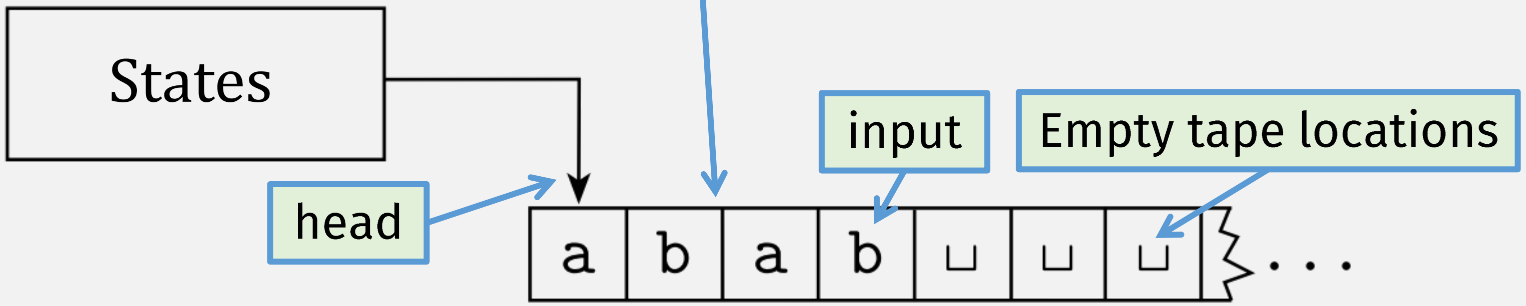
- Worked as codebreaker during WW2

- Also studied AI
  - Turing Test



Automata vs Turing Machines

- Turing Machines can read <u>and write</u> 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 $B = \{w\#w |\ w \in \{0,1\}^*\}$

input

tape

head

0 1 1 0 0 0 # 0 1 1 0 0 0 ␣ . . .

$M_1 =$ "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:

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

0 1 1 0 0 0 # 0 1 1 0 0 0 ␣ . . .

x 1 1 0 0 0 # 0 1 1 0 0 0 ␣ . . .

"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

"Cross off" = write "x" char

```
0 1 1 0 0 0 # 0 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # 0 1 1 0 0 0 ␣ ...
x 1 1 0 0 0 # x 1 1 0 0 0 ␣ ...
```

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

0 1 1 0 0 0 # 0 1 1 0 0 0 ␣ . . .

x 1 1 0 0 0 # 0 1 1 0 0 0 ␣ . . .

x 1 1 0 0 0 # x 1 1 0 0 0 ␣ . . .

x 1 1 0 0 0 # x 1 1 0 0 0 ␣ . . .

Turing Machine Example

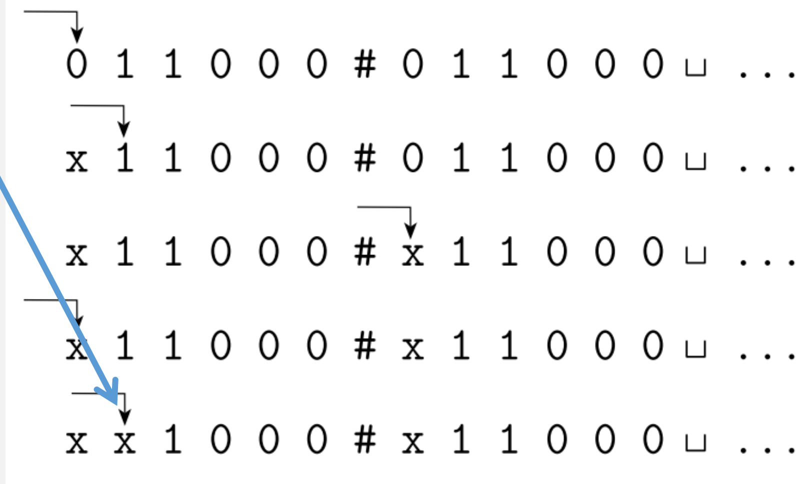
M_1 accepts inputs in language $B = \{w\#w \mid w \in \{0,1\}^*\}$

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



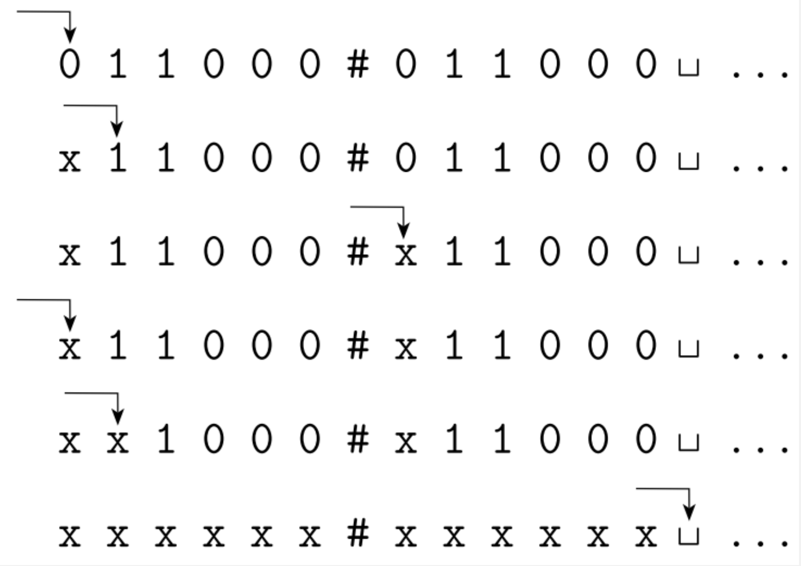
Turing Machine Example

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*."



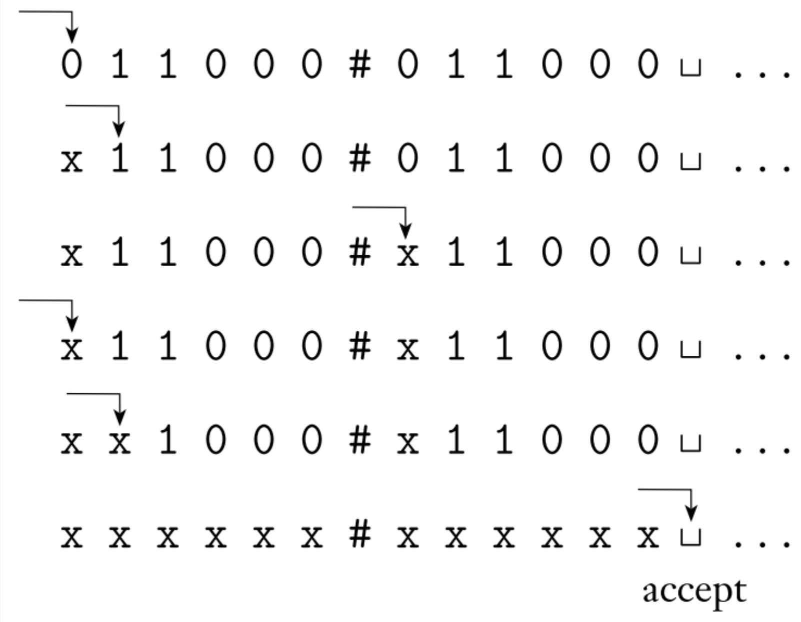
Turing Machine Example

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

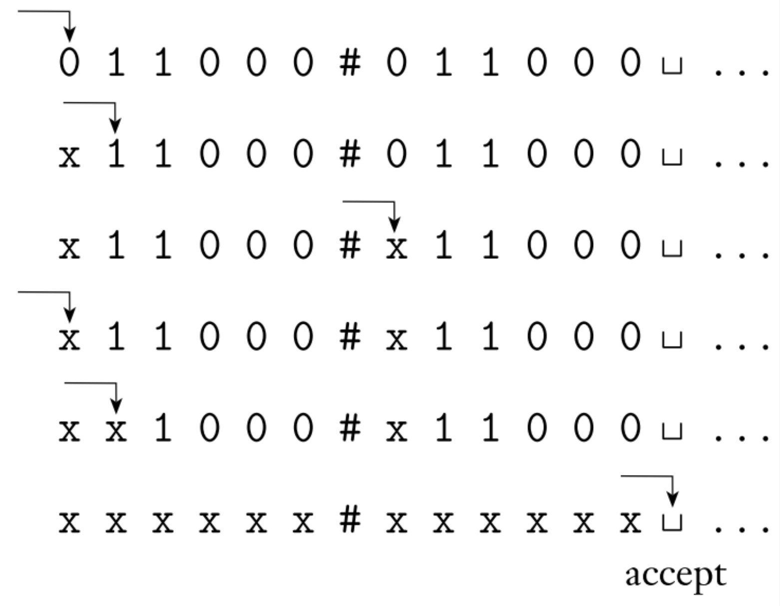
DEFINITION 3.3

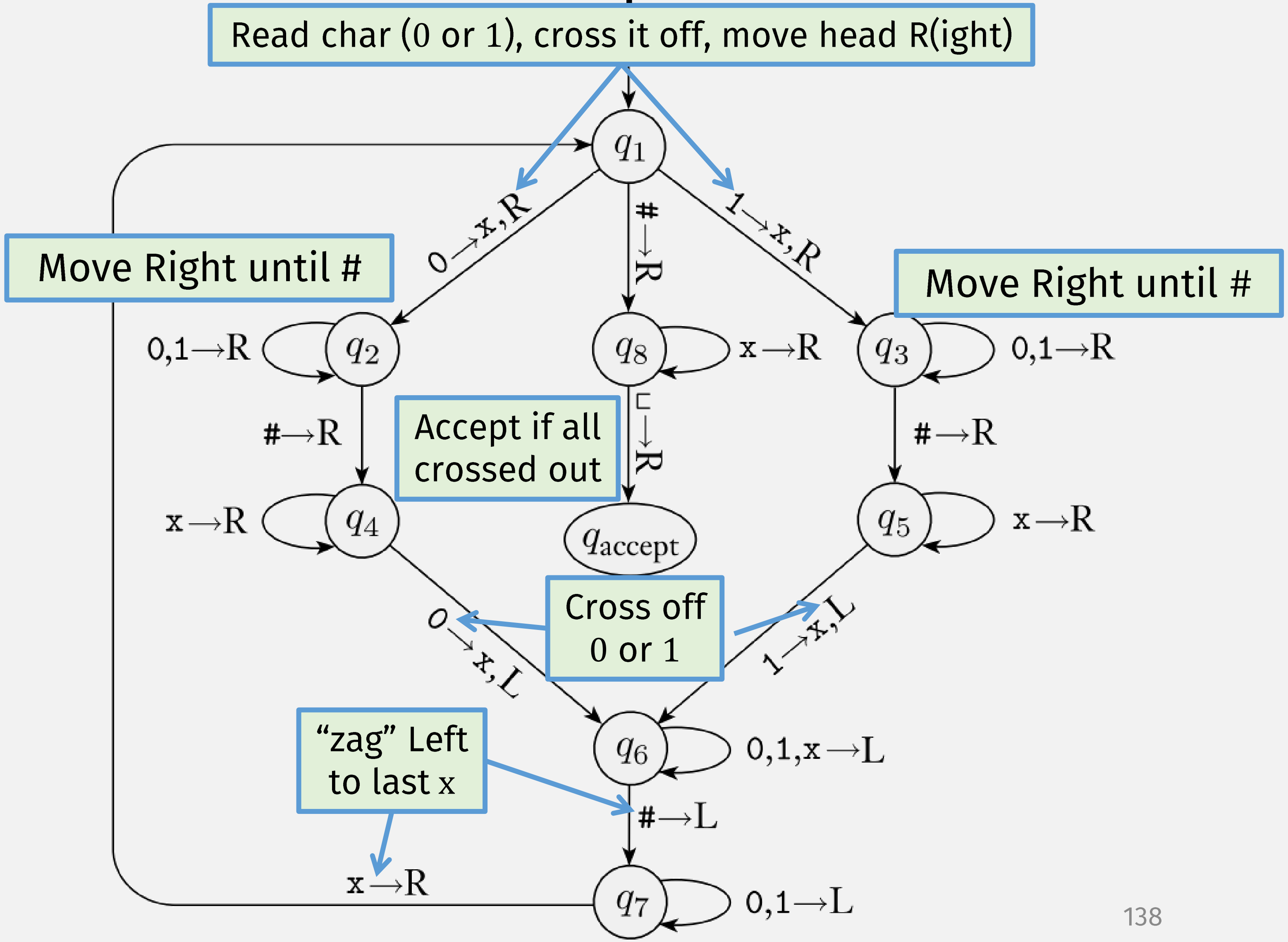
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\boxed{\sqcup} \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\boxed{\delta \colon Q \times \underbrace{\Gamma}_{\text{read}} \longrightarrow Q \times \underbrace{\Gamma}_{\text{write}} \times \underbrace{\{\mathrm{L}, \mathrm{R}\}}_{\text{move}}}$ 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 no[...] if no # is found, *reject*. Cross off symbols as they [...]p track of which symbols correspond.

2. When all symbols to [...]n crossed off, check for any remaining s[...]t of the #. If any symbols remain, *reject*; otherwise, [...]ccept."

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

TM Informal Description: Caveats

- TM informal descriptions are not a “do whatever” card
  - They must be sufficiently precise to communicate the formal tuple
- Input must be a string, written with chars from finite alphabet
- An informal “step” represents sequence of formal transitions
  - I.e., some **finite** number of transitions
  - It cannot run forever
  - E.g., can’t say “try all numbers” as a “step”

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 <u>run forever</u>
  - E.g., the head can move back and forth in a loop
- Thus, there are <u>two classes of Turing Machines</u>:
  - A <u>recognizer</u> is a Turing Machine that may run forever
  - A <u>decider</u> is a Turing Machine that always halts.

DEFINITION 3.5

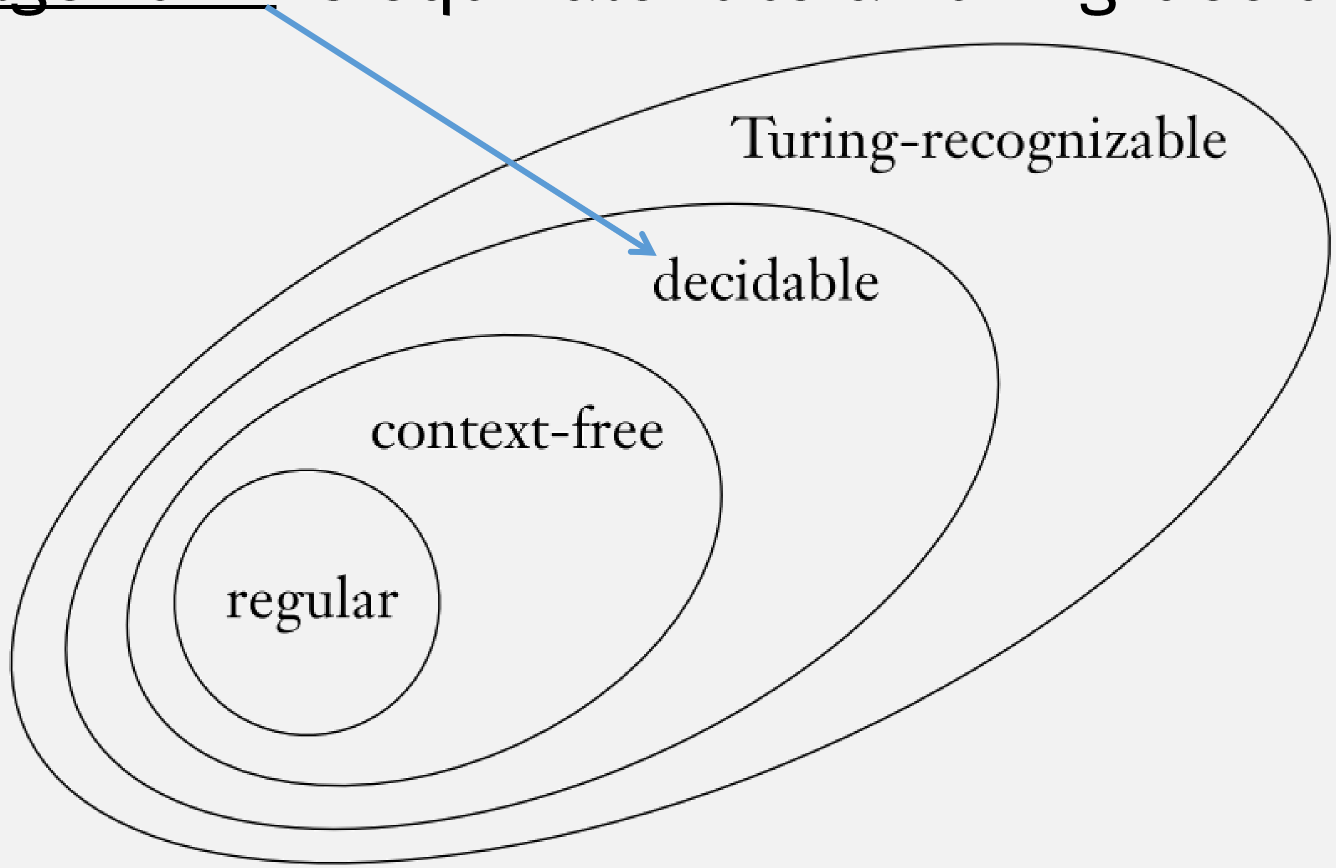
Call a language ***Turing-recognizable*** if some Turing machine recognizes it.

DEFINITION 3.6

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

Formal Definition of an "Algorithm"

- An <u>algorithm</u> is equivalent to a Turing-decidable Language



Check-in Quiz 3/10

On Gradescope