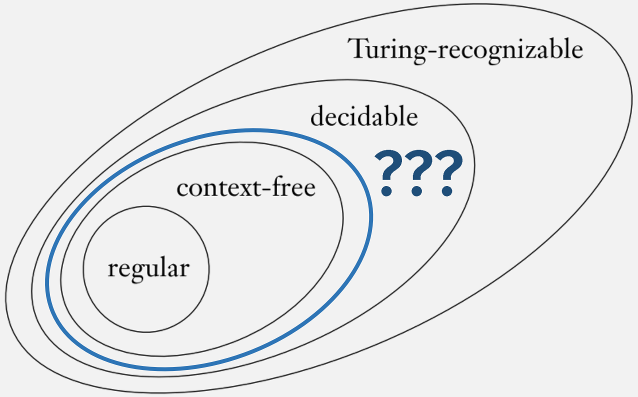
UMB CS420

Context-Sensitive Languages

Wednesday, March 30, 2022



Announcements

- HW 8 due 4/3 11:59pm

Last Time: Rice's Theorem

All languages that look like:

$ANYTHING_{\mathsf{TM}} = \{\langle M \rangle \mid M$ is a TM and "… **anything** …" about $L(M)\}$

… are **undecidable**!

This means: **There's <u>no algorithm</u>** **to compute anything about the** **language** (behavior) **of TMs,** i.e., **about programs!**

Last Time:

Langs about non-TMs: **Decidable**
Langs about TMs: **Undecidable**

- $A_{\mathsf{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ — Decidable
- $A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ — Decidable
- $A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ — **Undecidable**
- $E_{\mathsf{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$ — Decidable
- $E_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ — Decidable
- $E_{\mathsf{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$ — **Undecidable**
- $EQ_{\mathsf{DFA}} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$ — Decidable
- $EQ_{\mathsf{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$ — **Undecidable**
- $EQ_{\mathsf{TM}} = \{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$ — **Undecidable**

What's going on here?

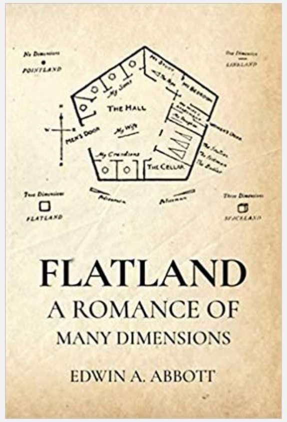
It breaks the pattern?

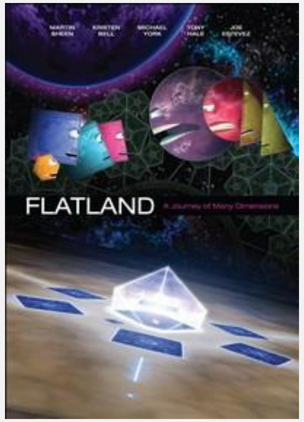
Last Time:

$A_{\mathsf{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts input string } w\}$ Decidable

$A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ Decidable

$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$ **Undecidable**



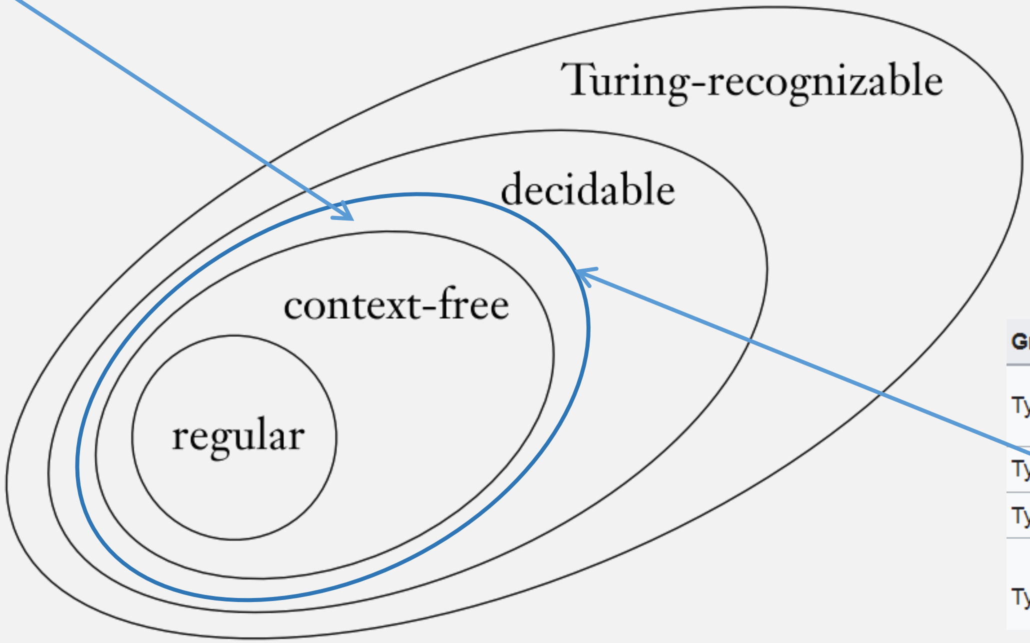


- In hindsight, **it makes sense that:**
  - A restricted TM (a **decider**) …
  - … can't simulate an unrestricted TM (a **recognizer**)

- But what about:
  - A restricted TM (a **decider**) …
  - … simulating an even more restricted TM (a **???**)

Context-Sensitive Languages

context-sensitive languages (CSL)
- generated by: **context-sensitive grammars** (CSG)
- recognized by: **linear bounded automata** (LBA)



Grammar	Languages	Automaton
Type-0	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine
Type-2	Context-free	Non-deterministic pushdown automaton
Type-3	Regular	Finite state automaton

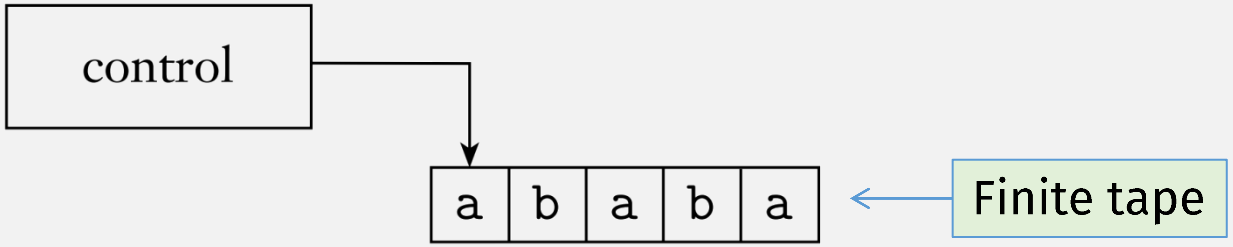
Chomsky hierarchy

From Wikipedia, the free encyclopedia

In formal language theory, computer science and linguistics, the **Chomsky hierarchy** (also referred to as the **Chomsky–Schützenberger hierarchy**)[1] is a containment hierarchy of classes of formal grammars.

Linear Bounded Automata

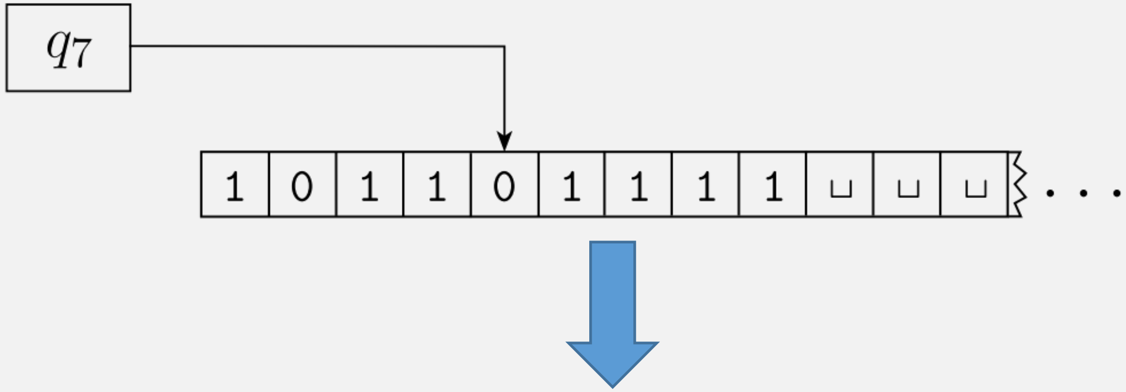
A ***linear bounded automaton*** is a restricted type of Turing machine wherein the tape head isn't permitted to move off the portion of the tape containing the input. If the machine tries to move its head off either end of the input, the head stays where it is—in the same way that the head will not move off the left-hand end of an ordinary Turing machine's tape.



Theorem: A_{LBA} is decidable

$A_{\mathsf{LBA}} = \{\langle M, w\rangle \mid M \text{ is an LBA that accepts string } w\}$

Flashback: TM Configuration = State + Head + Tape



$$1011q_7 01111$$

Textual representation of "configuration"

1st char after state is current head position

How Many Possible Configurations …

$$1011q_7 01111$$

- Does an LBA have?
  - q states
  - g tape alphabet chars
  - tape of length n

- # of possible ways to fill the tape:
  - g^n
- # of possible head positions:
  - n
- Total Possible Configurations = $\boxed{qng^n}$

It's finite!

Theorem: A_{LBA} is decidable

$A_{\text{LBA}} = \{\langle M, w \rangle \mid M \text{ is an LBA that accepts string } w\}$

Proof: Create decider for A_{LBA}

On input <M, w>:

- Simulate M on w:
  - If M accepts w, then accept <M, w>
  - If M rejects w, then reject <M, w>
- If M runs for more than qng^n steps …
- … then we are in a loop so halt and reject!

Termination argument?

Theorem: E_{LBA} is undecidable

$E_{\mathsf{LBA}} = \{\langle M \rangle \mid M \text{ is an } \mathsf{LBA} \text{ where } L(M) = \emptyset\}$

Flashback: TM Configuration Sequences

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

Single-step

head

Next config

(Right) $\alpha q_1 \mathbf{a} \beta \vdash \alpha \mathbf{x} q_2 \beta$

write

if $q_1, q_2 \in Q$

$\delta(q_1, \mathbf{a}) = (q_2, \mathbf{x}, \mathrm{R})$

read

$\mathbf{a}, \mathbf{x} \in \Gamma \quad \alpha, \beta \in \Gamma^*$

(Left) $\alpha b q_1 \mathbf{a} \beta \vdash \alpha q_2 b \mathbf{x} \beta$

if $\delta(q_1, \mathbf{a}) = (q_2, \mathbf{x}, \mathrm{L})$

Extended

- Base Case

$$\boxed{I \overset{*}{\vdash} I \text{ for any ID } I}$$

- Recursive Case

$\boxed{I \overset{*}{\vdash} J}$ if there exists some ID K

such that $I \vdash K$ and $K \overset{*}{\vdash} J$

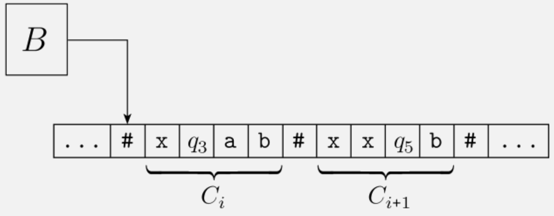
Theorem: E_{LBA} is undecidable

$E_{\text{LBA}} = \{\langle M\rangle \mid M \text{ is an LBA where } L(M) = \emptyset\}$

Proof, by contradiction:

- Assume E_{LBA} has *decider* R; use to create decider for A_{TM}:
- On input <*M*, *w*>, where $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$:
  - Construct LBA *B*, whose input is a sequence of *M* configs:
    - *B* accepts sequences of *M* configurations where *M* accepts *w*, i.e.,
      - **First configuration** is $q_0 w_1 w_2 \cdots w_n$
      - **Last configuration** has state q_{accept}
      - Each pair of **adjacent configs** is valid according to *M*'s δ
  - Run *R* with *B* as input:
    - If *R* accepts *B*, then *B*'s language is empty
      - So *M* has **no** sequence of configs that accepts *w*; so reject!
    - If *R* rejects *B*, then *B*'s language is not empty
      - So *M* has **a** sequence of configs that accepts *w*; so accept!

Input to *R*



So checking configuration sequences is a key capability!

So any language that can be used to check configuration sequences must be **undecidable**!

Theorem: ALL_{CFG} is undecidable

$ALL_{\mathsf{CFG}} = \{\langle G \rangle |\ G \text{ is a CFG and } L(G) = \Sigma^*\}$

Proof, by contradiction

- Assume ALL_{CFG} has a decider R. Use it to create decider for A_{TM}:

On input <M, w>:

Can a PDA do this?

- Construct a PDA P that rejects sequences of M configs that accept w
- Convert P to a CFG G (previous class)
- Give G to R:
  - If R accepts, then M has **no** accepting config sequences for w, so reject
  - If R rejects, then M has **an** accepting config sequence for w, so accept

A PDA That Rejects TM M Config Sequences

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

$\#\underbrace{\longrightarrow}_{C_1}\#\underbrace{\longleftarrow}_{C_2^{\mathcal{R}}}\#\underbrace{\longrightarrow}_{C_3}\#\underbrace{\longleftarrow}_{C_4^{\mathcal{R}}}\#\ \cdots\ \#\underbrace{\qquad}_{C_l}\#$

On input $\#\underbrace{\qquad}_{C_1}\#\underbrace{\qquad}_{C_2}\#\underbrace{\qquad}_{C_3}\#\cdots\#\underbrace{\qquad}_{C_l}\#$, nondeterministically:

- Reject if C_1 is not $q_0 w_1 w_2 \cdots w_n$
- Reject if C_l does not have q_{accept}
- Reject if any C_i and C_{i+1} is invalid according to δ:
  1. Push C_i onto the stack
  2. Compare C_i with C_{i+1} (reversed):
     a) Check that initial chars match
     b) On first non-matching char:
        - check that next 3 chars is valid according to δ
        - Each possible δ can be hard-coded since δ is finite
     c) Continue checking remaining chars
     d) Reject whenever anything is invalid

Example
C_i config (on the stack): q_0`0101`
Stack (C_i backwards): `1010`q_0
Next C_{i+1} config (rev): `1011`q_1
To validate:
- read, pop, check δ

Why **reject** accepting configuration sequences?

Could we create a PDA that **accepts** accepting configuration sequences?

But that would mean E_{CFG} (dual to ALL_{CFG}) is undecidable??

$E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$

We already proved this is **decidable!**

Algorithms For CFLs

- $A_{\text{CFG}} = \{\langle G, w\rangle |\ G \text{ is a CFG that generates string } w\}$ Decidable
- $E_{\text{CFG}} = \{\langle G\rangle |\ G \text{ is a CFG and } L(G) = \emptyset\}$ ← Already proved this is **decidable** Decidable
- $ALL_{\text{CFG}} = \{\langle G\rangle |\ G \text{ is a CFG and } L(G) = \Sigma^*\}$ ← Just proved this is **undecidable** **Undecidable**

These two languages tell us something about the **threshold of decidability**

Exploring the Limits of CFLs

- A CFL: $\{w_1\#w_2 \mid w_1 \neq w_2\}$
  - PDA nondeterministically checks matching positions in $1^{st}/2^{nd}$ parts
  - And rejects if **<u>any</u>** are not the same
  - I.e., Each computation branch is **independent**, i.e. "**context free**"

This is similar to the <u>config-rejecting PDA</u>

- <u>Not</u> a CFL: $\{w_1\#w_2 \mid w_1 = w_2\}$
  - Can nondeterministically check matching positions
  - But needs to accept only if **<u>all</u>** branches match
  - I.e., Each branch is **not independent**

This is similar to the *ww* language (not pumpable)

An <u>config-accepting PDA</u> would be like this language … i.e., not a CFL!

Summary: the "**context freeness**" of CFLs has to do with <u>dependency</u> between non-deterministic computation branches

(This is also why **union is closed for CFLs** but **intersection is not**)

Algorithms For CFLs

- $A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$ Decidable
- $E_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$ ← Already proved this is **decidable** Decidable
- $ALL_{\text{CFG}} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) = \Sigma^*\}$ ← Just proved this is **undecidable** **Undecidable**
- $EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$ **Undecidable?**

(Still need to prove this is undecidable)

Theorem: EQ_{CFG} is undecidable

$EQ_{\text{CFG}} = \{\langle G, H\rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$

- Proof by contradiction: Assume EQ_{CFG} has a decider R
- Use R to create a decider for ALL_{CFG}:

On input <G>:
- Construct a CFG G_{ALL} which generates all possible strings
- Run R with G and G_{ALL}
- Accept G if R accepts, else reject

The Post Correspondence Problem (PCP)

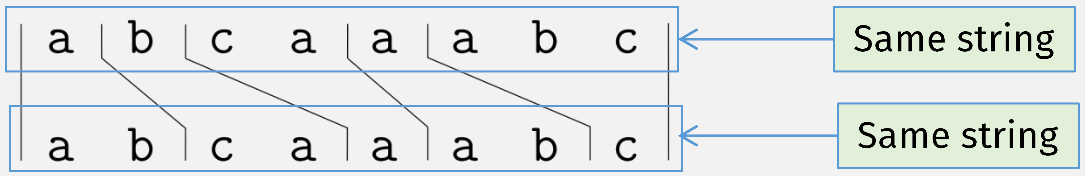
A unique undecidable problem

A Non-Formal Languages Undecidable Problem: *PCP*

- Let P be a set of "**dominos**" $\left\{ \left[\frac{t_1}{b_1}\right], \left[\frac{t_2}{b_2}\right], \ldots, \left[\frac{t_k}{b_k}\right] \right\}$
  - Where each t_i and b_i are strings
  - E.g., $P = \left\{ \left[\frac{\texttt{b}}{\texttt{ca}}\right], \left[\frac{\texttt{a}}{\texttt{ab}}\right], \left[\frac{\texttt{ca}}{\texttt{a}}\right], \left[\frac{\texttt{abc}}{\texttt{c}}\right] \right\}$
- A **match** is:
  - A sequence of dominos with the same top and bottom strings

Repeats allowed

  - E.g., $\left[\frac{\texttt{a}}{\texttt{ab}}\right]\left[\frac{\texttt{b}}{\texttt{ca}}\right]\left[\frac{\texttt{ca}}{\texttt{a}}\right]\left[\frac{\texttt{a}}{\texttt{ab}}\right]\left[\frac{\texttt{abc}}{\texttt{c}}\right] \Rightarrow$



- Then: $PCP = \{ \langle P \rangle \mid P \text{ is a set of dominos with a match} \}$

Theorem: PCP is undecidable

PCP = { <P> | P is a set of dominos with a match }

Proof by contradiction:

Assume PCP is decidable, has decider R; use it to create decider for A_{TM}:

On input <M, w>:
1. Construct a set of dominos P that has a **match** only when M accepts w
2. Run R with P as input
3. Accept if R accepts, else reject

The trick: P has M's TM configurations as its domino strings

So a match is a sequence of configs showing M accepting w!

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

PCP Dominos

- First domino: $\left[\dfrac{\#}{\# q_0 w_1 w_2 \cdots w_n \#}\right]$
- Key idea: add dominos representing valid TM steps:

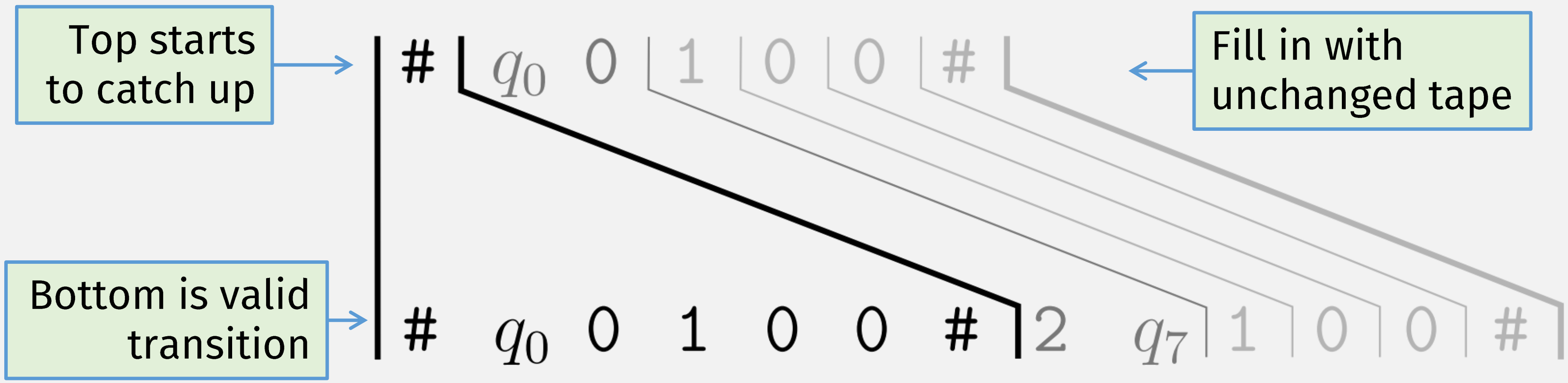
$$\text{if } \delta(q, a) = (r, b, \mathrm{R}),\ \text{put } \left[\frac{qa}{br}\right] \text{ into } P$$

$$\text{if } \delta(q, a) = (r, b, \mathrm{L}),\ \text{put } \left[\frac{cqa}{rcb}\right] \text{ into } P$$

- For the tape cells that don't change: $\text{put } \left[\frac{a}{a}\right] \text{ into } P$
- Top can only "catch up" if there is an accepting config sequence

PCP Example

- Let w = `0100` and $\delta(q_0, 0) = (q_7, 2, \mathrm{R})$ so $\left[\frac{q_0 0}{2q_7}\right]$ in P



PCP Dominos (accepting)

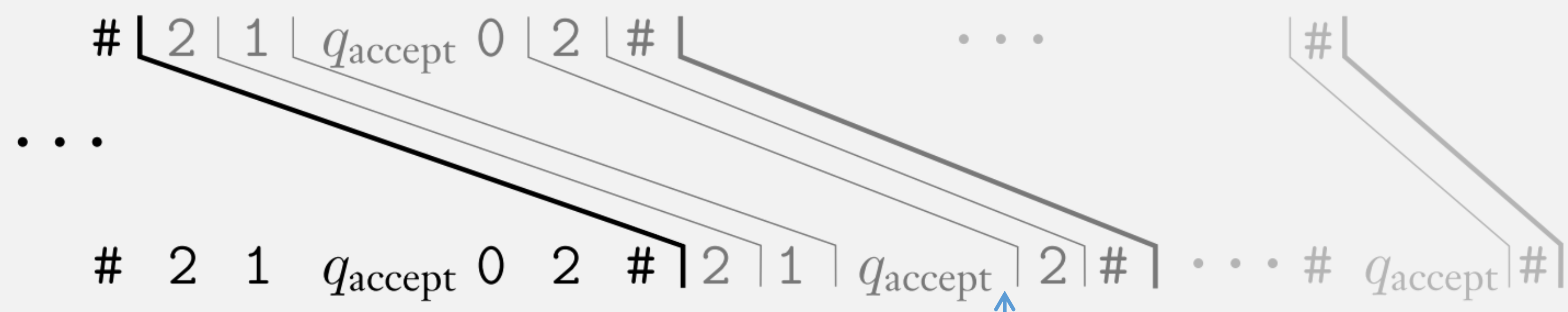
- When accept state reached, let top "catch" up:

For every $a \in \Gamma$,

put $\left[\frac{a\, q_{\text{accept}}}{q_{\text{accept}}}\right]$ and $\left[\frac{q_{\text{accept}}\; a}{q_{\text{accept}}}\right]$ into P

Bottom "eats" one char

Only possible match: **accepting sequence of TM configs**



"eat" one char

Check-in Quiz 3/30

On gradescope