Pumping Lemma: Given a regular language L, there exists some pos. int. m such that, for every $w \in L$, $|w| \geq m$, there is a way to write $w = xyz$, $|xy| \leq m$, $y \neq \lambda$ so that $xy^i z \in L$ for all i.

We use the Pumping Lemma to show languages are not regular: if we show that for every m, there is some $w \in L$, so that, for every way we write $w = xyz$, $|xy| \leq m$, $y \neq \lambda$, there is some i with $xy^i z \notin L$, then L must be not regular.

$L = \{a^n b^n \mid n \geq 1\}$ is not regular

Pf: Given m, let $w = a^m b^m = aa\cdots abb\cdots b$. Suppose $w = xyz$, $|xy| \leq m$, $y \neq \lambda$. Since $|xy| \leq m$, $xy = a\cdots a$, so $y = a^j$ for some j, $1 \leq j \leq m$.

Then $xy^0z = a^{m-j}b^m$. Since $m-j \neq m$,

~~xy^0z~~ $xz \notin L$.

Construct a PDA for

$$L = \{ w \mid 2n_a(w) \leq n_b(w) \leq 3n_a(w) \}$$

$(q_0, a, z) \longrightarrow (q_0, AAz)$

$(q_0, a, z) \longrightarrow (q_0, AAAz)$

$(q_0, a, A) \longrightarrow (q_0, AAA)$

$(q_0, a, A) \longrightarrow (q_0, AAAA)$

$(q_0, b, A) \longrightarrow (q_0, \lambda)$

$(q_0, \lambda, z) \longrightarrow (q_f, \lambda)$

Idea: # of A's on stack count how many ~~we need to have~~ b's we are owed.

Attempt 1 - this doesn't work if b's come first - we need to figure out what to do if we have a b w/ z on stack.

We have a problem - one b doesn't cause us to be owed an entire A - we have to think about being owed $\frac{1}{2}$ an a or $\frac{1}{3}$ of an a. - we only have a finite # of possibilities - $\frac{1}{2}$, $\frac{1}{3}$, or $\frac{2}{3}$

q_0 - owed no fractions of A's

$q_{\frac{1}{2}}$ - owed $\frac{1}{2}$ A

$q_{\frac{1}{3}}$ owed $\frac{1}{3}$ A

$q_{\frac{2}{3}}$ owed $\frac{2}{3}$ A

Also, on the stack, we can put B's to count how many full a's we are owed.

$(q_0, b, z) \longrightarrow (q_{\frac{1}{2}}, z)$

~~$(q_0, b, A) \longrightarrow (q_{\frac{1}{2}}, A)$~~

$(q_0, b, B) \longrightarrow (q_{\frac{1}{2}}, B)$

$(q_0, b, z) \longrightarrow (q_{\frac{1}{3}}, z)$

$(q_0, b, B) \longrightarrow (q_{\frac{1}{3}}, B)$

we're arranging this so that we never have both A's + B's on stack, and, if we have A's on stack, we are in q_0.

$(q_{\frac{1}{2}}, b, z) \longrightarrow (q_0, Bz)$

$(q_{\frac{1}{2}}, b, B) \longrightarrow (q_0, BB)$

$(q_{\frac{1}{2}}, a, z) \longrightarrow (q_0, Az)$

owed $\frac{1}{2}$A, get 1a, now owe 1 b

$\rightarrow (q_{\frac{1}{2}}, a, B) \longrightarrow (q_{\frac{1}{2}}, \lambda)$

owed $n+\frac{1}{2}$a, get 1a, now owed $(n-1)+\frac{1}{2}$a

stuff for $q_{\frac{1}{3}}$ + $q_{\frac{2}{3}}$

getting a CFG from a PDA:

~~Plan~~ Idea: [redacted]

variables look like

[state, stack symb, state] or [q, s, q']

and the strings that can be generated from [q, s, q'] are the strings that the PDA can eat going from q to q' eating s from stack.

What are the productions corresponding to a transition

$$(p, \ell, t) \longrightarrow (p', uv)$$

This tells us, for any states a, b

$$[p, t, a] \longrightarrow \ell [p', u, b][b, v, a]$$

go from p to a eating t from the stack by first using the transition, which eats ℓ from string, now you're at p' → you need to get from p' to a eating uv from stack

you go from p' to b eating u & b to a eating v.