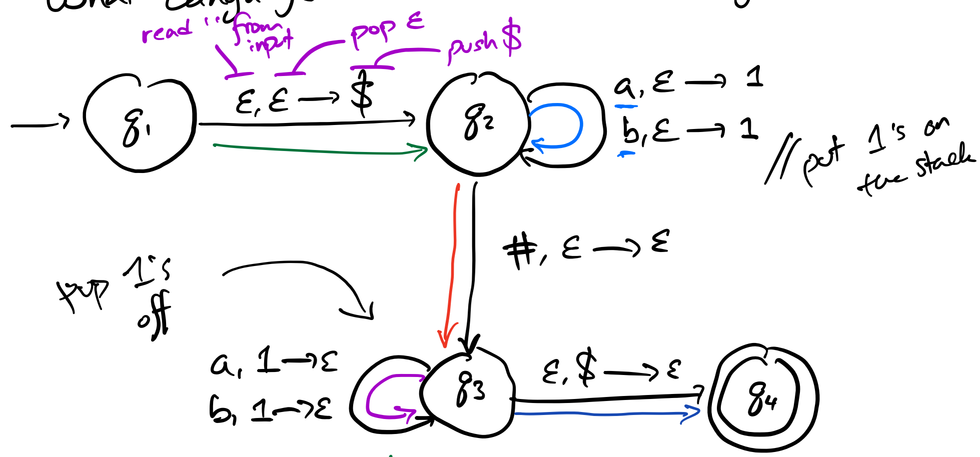
COMS W3261 — Lecture 7

- Equivalence of CFGs and PDAs.
- Non context-free languages (new Pumping Lemma!)

Teaser: What language does this PDA recognize?



Obs: all accepting strings must go $q_1 \to q_2 \to q_3 \to q_4$.

What does an accepting computation look like?

(1) Push $ onto the stack

(2) In state q_2, read in some (could be 0) a's and b's and push a 1 onto the stack for each a or b we read.

(3) Read a #.

(4) In state q_3, read in some a's and b's and pop 1's off the stack.

(5) Pop $ when my stack is exhausted, then accept if there are no more input symbols.

$$\{ (a \cup b)^k \# (a \cup b)^k \mid k \geq 0 \}$$

Announcements: HW #4 due Monday, 7/26/21 @ 11:59 PM EST.

If median grade on all HWks is low (<85), we'll curve some up. None will curve down.

Possible extra credit on HW #5, #6.

Readings: Sipser 2.2 (PDA = CFGs)
Sipser 2.3 (Non-context free languages)

Today: 1. Quick PDA review
2. PDAs recognize exactly the CFLs
   2.1) CFG $\to$ PDA
   2.2) PDA $\to$ CFG

3. Some languages are not context-free using Context-Free Pumping Lemma.

1. PDA review.

Def. A PDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

Q is a finite set of states, — if designing: can be whatever is convenient.

Σ and Γ are finite input and stack alphabets.

$$\delta : \underbrace{Q}_{\text{state}} \times \underbrace{\Sigma_\varepsilon}_{\text{input symbol (or }\varepsilon)} \times \underbrace{\Gamma_\varepsilon}_{\text{stack symbol pop}} \longrightarrow \underbrace{\mathcal{P}}_{\text{"all subsets of"}}(\underbrace{Q}_{\text{state}} \times \underbrace{\Gamma_\varepsilon}_{\text{stack symbol to push}})$$

$q_0 \in Q$ is the start state, and

$F \subseteq Q$ is the set of accept states.

A PDA accepts the input

$$w = w_1 w_2 \cdots w_m, \quad (w_i \in \Sigma_\varepsilon),$$

if there exist states and strings

$r_0, r_1 \cdots r_m \in Q,$

$s_0, s_1 \cdots s_m \in \Gamma^*$, such that

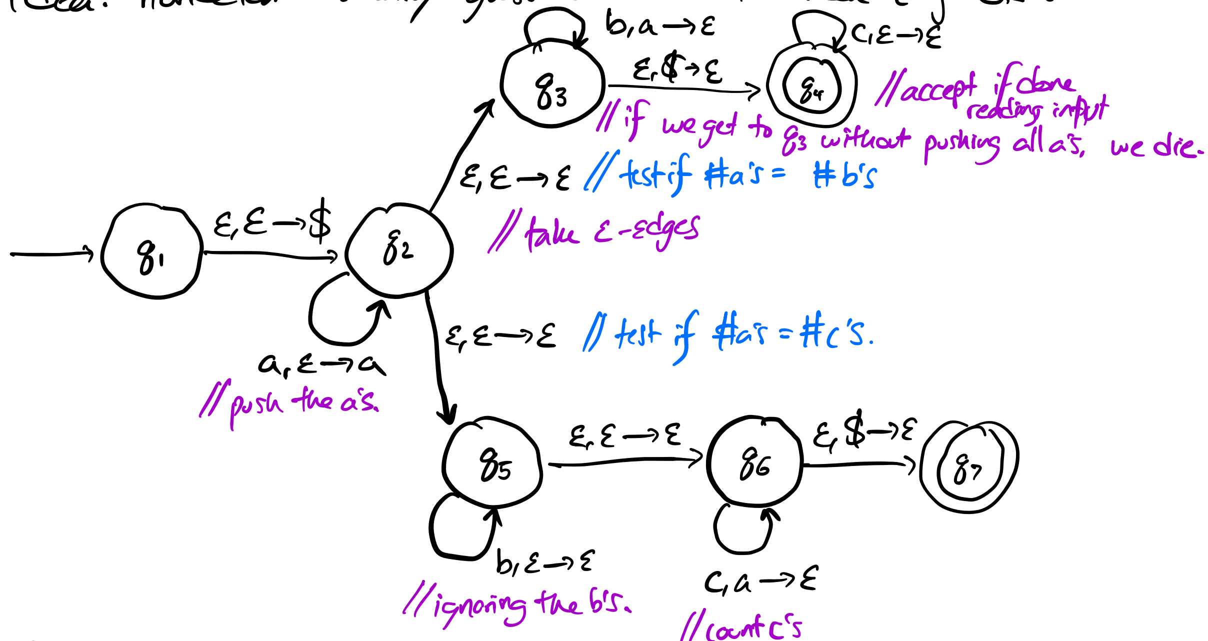
(1) $r_0 = q_0$, $r_m \in F$, $s_0 \in \varepsilon$,

(2) $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, for all $i = 0, 1, \cdots m-1$,

and $s_i = at$, $s_{i+1} = bt$ for some $a, b \in \Gamma$, and $t \in \Gamma^*$.

Example. We'll build a PDA for $L = \{a^i b^j c^k \mid i,j,k \geq 0, \text{ and } i=j \text{ OR } i=k\}$

Idea: nondeterministically guess whether to check $i=j$ OR $i=k$.



Remember: because δ maps to $\mathcal{P}(Q \times \Gamma_\varepsilon)$, every branch may have a different stack state.

2. PDAs recognize CFLs

We will show

Theorem: (PDA = CFL.) A language is context-free (:= described by some CFG) if and only if some Pushdown Automaton recognizes it.

This follows from two lemmas:

Lemma 1 (CFG $\to$ PDA). If a language is CF, some PDA recognizes it.

Lemma 2. (PDA $\to$ CFG). If a PDA recognizes some language, it is context-free.

Proof of L1 idea: CFGs derive every string by a series of substitutions. Given some CFG G, we'll show how to build a PDA P that nondeterministically derives all strings according to all substitution rules. Meanwhile, we'll check if any derivation matches the input string.

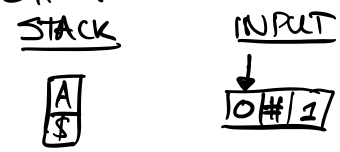
Example of PDA operation.

Consider G : $A \longrightarrow 0A1 \mid \#$

G recognizes $L = \{0^n \# 1^n \mid n \geq 0\}$.

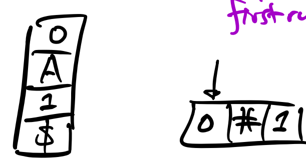
- What will our equivalent PDA do on input $0\#1$?

(1) Push \$ and the start symbol.

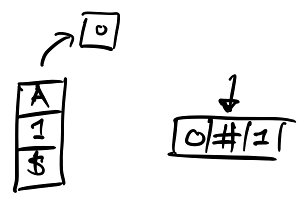


(consider branch picking first rule.)

(2) If the top of the stack is a variable, (nondeterministically) choose a rule and substitute.



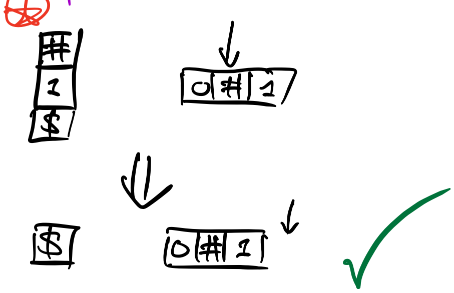
(3) If the top of my stack is a terminal, read a character from the input. If it matches the top of the stack, pop the stack. If not, die.



(idea: if 0 on the left, this will be true at end of derivation.)

(pick second rule, sub A)

(4) Repeat steps 2 and 3 until the branch dies or \$ appears on the stack. If the \$ appears, accept if all input has been read.



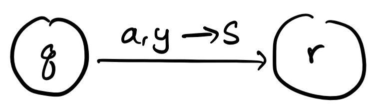
Lemma 1. If a language is context-free, some PDA recognizes it.

Proof. Let $G = (V, \Sigma, R, S)$ be a grammar. We'll show how to build an equivalent PDA $P = (Q, \Sigma, \Gamma, \delta, q_{start}, F)$.

[Shorthand: pushing whole strings. We write, for $s \in \Gamma^*$

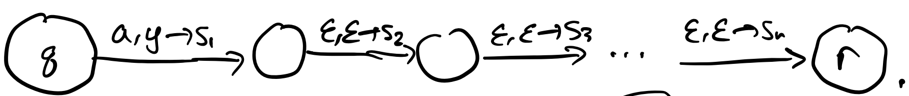
$(r, s) \in \delta(q, a, y)$ to mean

"If we are at state q, see input a, and we pop $y \in \Gamma$; push the entire string s and move to state r."



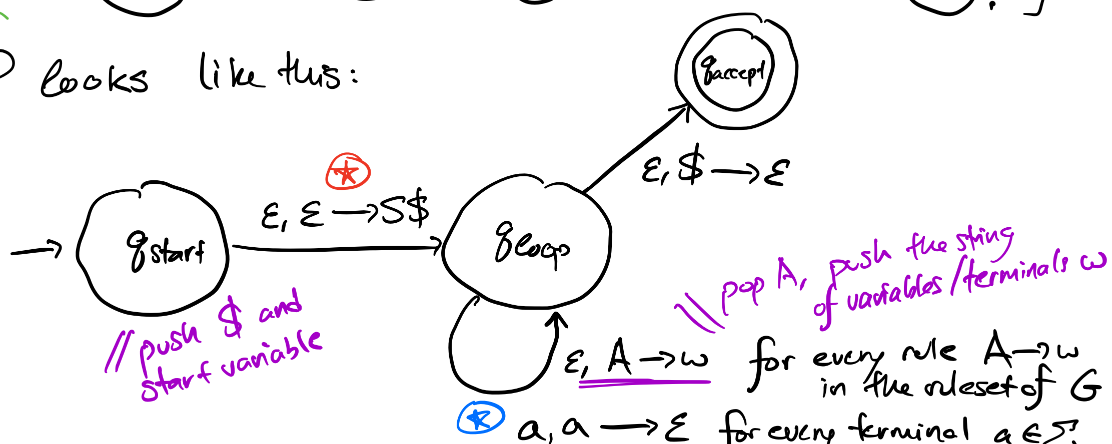
$S = S_1 S_2 \cdots S_n$.

shorthand for



.]

P looks like this:



Why does this accept iff our input string can be derived from S?

If $S \overset{*}{\Rightarrow}$ input, then some leftmost derivation exactly matches a branch of computation.

If some branch accepts, then we must have substituted all variables for terminals and matched all terminals with the input string. Thus there exists a derivation for the input.

(P, formally: $Q = \{q_{start}, q_{loop}, q_{accept}\} \cup \{$other states necessary to push strings$\}$

$\Sigma = \Sigma$

$\Gamma = V \cup \Sigma \cup \{\$\}$

$q_0 = q_{start}$

$F = \{q_{accept}\}$

δ contains the following:

$$\delta(q_{start}, \varepsilon, \varepsilon) = \{(q_{loop}, S\$)\}$$

$$\delta(q_{loop}, \varepsilon, A) = \{(q_{loop}, w) \mid A \to w \text{ is a rule in } R\}$$

$$\delta(q_{loop}, a, a) = \{(q_{loop}, \varepsilon)\} \text{ for all } a \in \Sigma$$

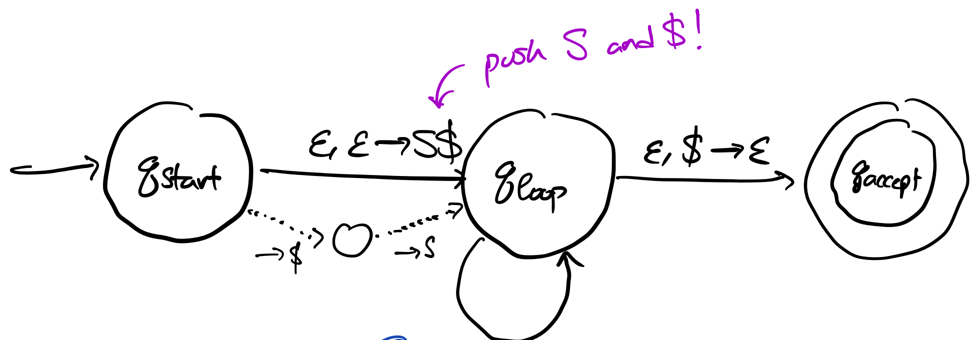
$$\delta(q_{loop}, \varepsilon, \$) = \{(q_{accept}, \varepsilon)\}.$$

(δ evaluates to $\emptyset$ for all other inputs.) ■

Example. CFG $\longrightarrow$ Equivalent PDA.

Consider G:

$$S \longrightarrow aTb \mid b$$
$$T \longrightarrow Ta \mid \varepsilon$$



$\varepsilon, S \longrightarrow aTb$

$\varepsilon, S \longrightarrow b$

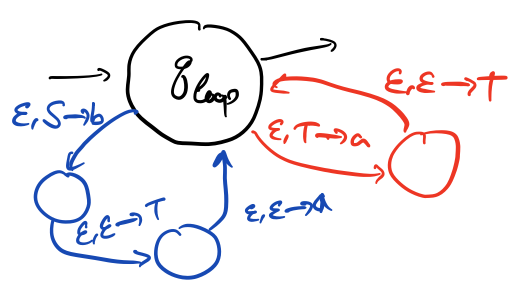
$\varepsilon, T \longrightarrow Ta$

$\varepsilon, T \longrightarrow \varepsilon$

$a, a \longrightarrow \varepsilon$

$b, b \longrightarrow \varepsilon$

zoom in on q_{loop}



CFG $\to$ PDA

NEXT.

Break.

Back at 11:40.

Lemma 2. If a PDA recognizes some language, that language is context-free.

Proof will be "sketch." Full proof is in Sipser 2.2.

Picture: some computational path in a PDA that takes us from q_1 to q_2 with empty stacks at the beginning and end.

Idea: create some CFG variable $A_{q_1 q_2}$ that will generate all the strings that might take us from q_1 to q_2 with the stack empty before and after.

Substitution rules will break $A_{q_1 q_2}$ down into pieces.

Example: $A_{q_1 q_2} \to A_{q_1 q_r} A_{q_r q_2}$.

Proof sketch: Given a PDA P, we want to create a CFG G that generates the language P recognizes.

Step 1: Simplify P without loss of generality. we make an assumption that doesn't limit the scope of our proof.

We can assume:

(1) P has exactly one accept state, q_{accept}.

(Can add ε-edges from old accept states to q_{accept}.)

(2) The stack is empty when we accept.

(Can add edges $\varepsilon, a \to \varepsilon$ to q_{accept} state for all $a \in \Gamma$.)

(3) All transitions either push or pop (but not both or neither.)

(Convert transitions that do both into two transitions, two states.

$\bigcirc \xrightarrow{\varepsilon, a \to b} \bigcirc \implies \bigcirc \xrightarrow{\varepsilon, a \to \varepsilon} \bigcirc \xrightarrow{\varepsilon, \varepsilon \to b} \bigcirc$)

(Convert transitions that do neither by pushing and popping an extra symbol.

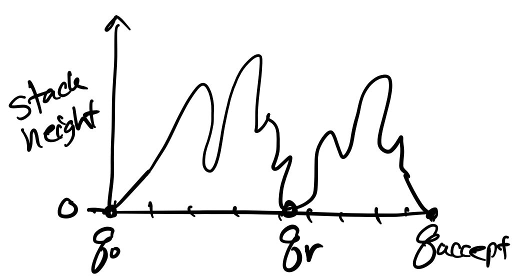
$\bigcirc \xrightarrow{a, \varepsilon \to \varepsilon} \bigcirc \implies \bigcirc \xrightarrow{a, \varepsilon \to ☺} \bigcirc \xrightarrow{\varepsilon, ☺ \to \varepsilon} \bigcirc$)

We will add the variable $A_{q_0 q_{accept}}$ to G. We want to make sure $A_{q_0 q_{accept}}$ derives every string in the language of P. (By our assumptions, this means P and G are equivalent.)

with empty stacks before/after.

Two cases for $A_{q_0 q_{accept}}$.

<u>Case 1</u>: strings s that go $q_0 \to q_{accept}$ and empty the stack in between.



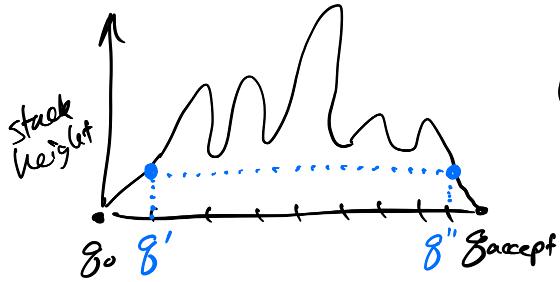
(on string s)

So: we can replace $A_{q_0 q_{accept}} \longrightarrow A_{q_0 q_r} A_{q_r q_{accept}}$, where these two variables generate (empty-stack) strings $q_0 \to q_r$ and $q_r \to q_{accept}$, respectively.

This rule will generate strings that take us from q_0, ~~to~~ q_r, to q_{accept}, with empty stacks at these three main points.

<u>Case 2</u>: strings s

that go $q_0 \to q_{accept}$ and don't empty the stack in between.



(on string s)

Any string in case 2 pushes some symbol α and goes to q'; and ends by popping α and going to q''.

So: we can replace $A_{q_0 q_{accept}}$ with $a A_{q'q''} b$, where a and b are the input symbols we read going from $q_0 \to q'$ and $q'' \to q_{accept}$. ($A_{q'q''}$ generates all strings that take us $q' \to q''$ with empty stacks on each end.)

<u>Claim.</u> (not proved). These two rule types capture all strings that from q_0 to q_{accept} with empty stacks at either end. Moreover, now we can <u>recurse</u> for all smaller A-variables!

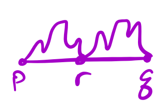
(will also add some rules $A_{qq} \to \varepsilon$.)

<u>Formal construction.</u> Say that $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, and construct $G = (V, \Sigma, R, S)$.

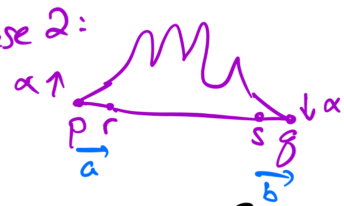
$$V = \{A_{pq} \mid p, q \in Q\}$$
$$S = A_{q_0 q_{accept}}.$$

<u>Ruleset:</u>

1) For $p, q, r \in Q$, $A_{pq} \longrightarrow A_{pr} A_{rq}$ // Case 1



2) For $p, q, r, s \in Q$, $\alpha \in \Gamma$, and $a, b \in \Sigma_\varepsilon$,
if $(r, \alpha) \in \delta(p, a, \varepsilon)$
$(q, \varepsilon) \in \delta(s, b, \alpha)$,
add rule $A_{pq} \longrightarrow a A_{rs} b$ // Case 2:



3) For each $p \in Q$, put the rule $A_{pp} \to \varepsilon$ in G.

Claim (again improved). A_{pq} generates the string s if and only if s can take us from p to q in P, with empty stacks before and after. (Proof by induction.)

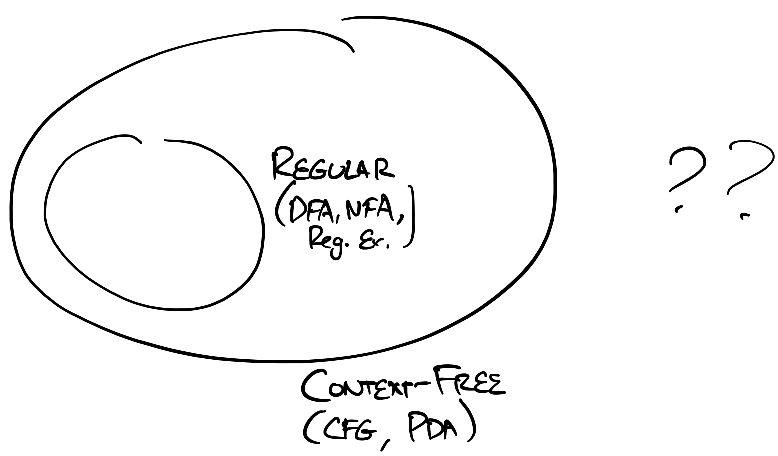
It follows that $A_{q_0 q_{accept}}$ generates all strings that take us from q_0 to q_{accept} on P with an empty stack at the end. By our assumptions, this the language of P. ■

Takeaway: CFG ⟶ PDA
PDA ⟶ CFG

Result: A language is context-free if and only if some PDA recognizes it. (Sipser p. 121–124.)

---

3. Non-context free languages.



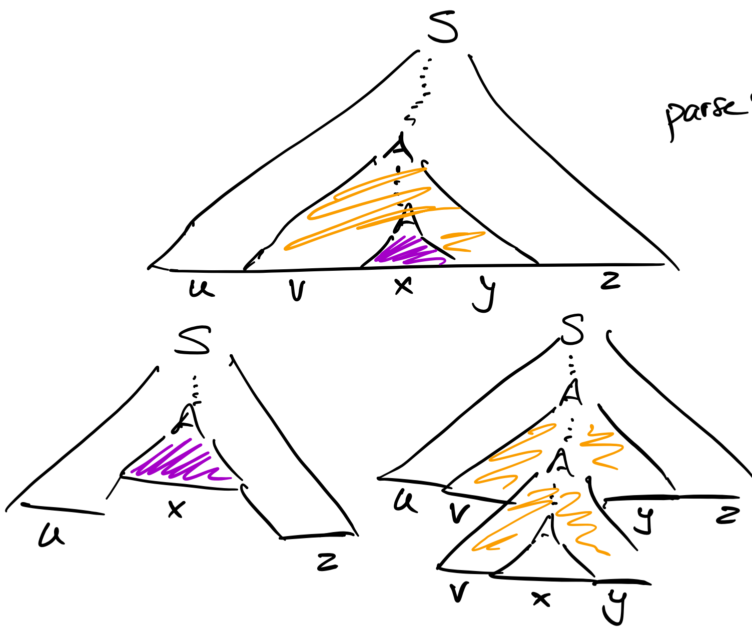
Context-Free Pumping Lemma:

- All CF languages have a certain property. (sufficiently long strings can be "pumped.")
- To show that a language is not context-free, assume that it satisfies the CFPL and show a contradiction.

Theorem. (PL for Context-Free languages). If L is a context-free language, there exists some "pumping length" p such that all strings $s \in L$, $|s| \geq p$ can be divided into five substrings $s = uvxyz$ such that

(1) for all $i \geq 0$, $uv^ixy^iz \in L$. // like $xy^iz \in L$
(2) $|vy| > 0$, // like $|y| > 0$
(3) $|vxy| \leq p$. // like $|xy| \leq p$

Idea: CFLs have CFGs with a finite number of variables. Sufficiently long strings must repeat a variable in any parse tree. This creates a "loop" in the derivation that we can pump.



parse tree for long strings s

also valid parse trees.

uv^2xy^2z.

Next time: CFPL, Turing Machines.
Reading: Sipser 2.2, 2.3