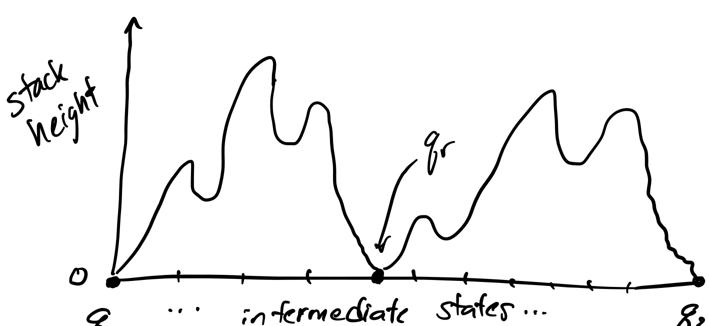
COMS W3261 — Lecture 7, part 2:

- PDA → CFG
- PL for Context-Free Languages.

<u>Lemma 2</u>: If a PDA recognizes some language, then that language is context-free. (Full proof: Section 2.2 of Sipser)

Picture: consider some computational path that takes us from state q_1 to state q_2, with an empty stack at the beginning and end.



Idea: suppose we have a CFG variable $A_{q_1 q_2}$ that generates all strings that take us from q_1 to q_2, with empty stacks. Then any particular string can be subdivided into strings that take us between intermediate states.

Say $A_{q_1 q_2}$ generates a string s, takes us $q_1 \to q_2$ then (in the picture above), s is generated by

$$A_{q_1 q_r} A_{q_r q_2}$$

<u>Proof sketch</u>. Given a PDA P, we'd like to create a CFG G that recognizes the same language.

Step 1) Simplify P so that several assumptions hold:

(1) Exactly one accept state, q_{accept}.

main thing: can simplify any PDA to be "nice."

(To do this: add ε-edges from the old accept states.)

(2) The stack is empty when we accept.

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

(3) All transitions either push <u>or</u> pop but not both.

(Convert transitions that do both $\longrightarrow$ two separate transitions with a new state)

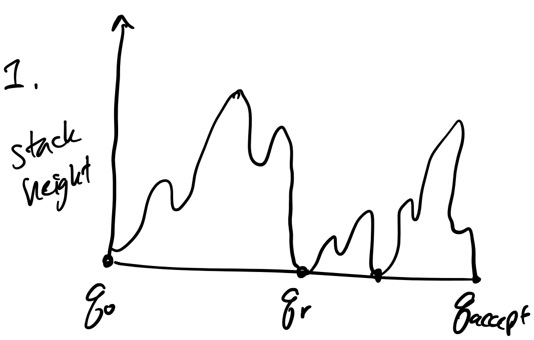
(Convert transitions that do neither $\longrightarrow$ two separate transitions that push, pop a meaningless symbol.)

Now: consider any accepting computational path from q_0 to q_{accept}, corresponding to some string s. We will create a grammar G with a new variable $A_{q_0\, q_{accept}}$ designed to derive every string that takes us from $q_0 \longrightarrow q_{accept}$ with an empty stack at the end.

(This is all accepting strings.)

Two cases on s:

Case 1.



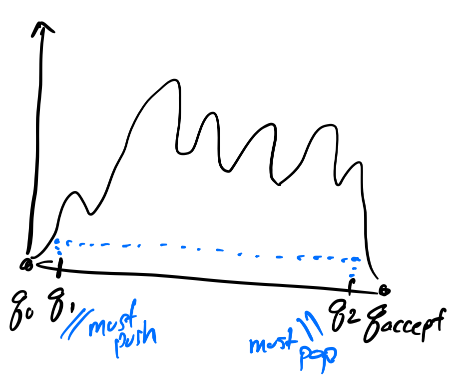
stack empty at some intermediate point.

So: create rule

$$A_{q_0 q_{accept}} \longrightarrow \underline{A_{q_0 q_r} A_{q_r q_{accept}}}$$

these variables will generate all strings between their states w/ an ending empty stack.

Case 2.



stack never empties completely.

So: create a rule

computation $q_1 \to q_2$

$$A_{q_0 q_{accept}} = a \underline{A_{q_1 q_2}} b,$$

where a and b are the input symbols read on the first and last step.

Claim. (unproved.) These two types of rules allow $A_{q_0 q_{start}}$ to generate all strings $q_0 \to q_{accept}$ with the stack empty at both ends.

(Also: add a base case $A_{qq} \to \varepsilon$.)

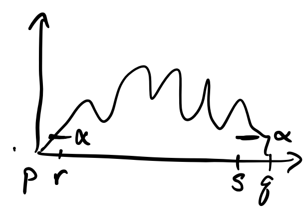
Construction. Say we have $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$. Construct $G = (V, \Sigma, R, S)$ as follows.

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

// goal: A_{pq} to derive all strings that take us $p \to q$, with stack empty at the end.

Rules R:

1) For $p, q, r \in Q$, add the rule $A_{pq} \to A_{pr} A_{rq}$.
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 the rule $A_{pq} \to a A_{rs} b$.



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

Claim (unproved.) A_{pq} generates the string x if and only if x can bring P from p to q w/ empty stack at both ends. (Proof by induction.)

Thus $A_{q_0 q_{accept}}$ derives strings that take us from q_0 to q_{accept} w/ empty stack at the end. By our original simplifying assumptions, this is the language recognized by P. ∎

Take away: Lemma 1. CFG $\to$ PDA.
Lemma 2. PDA $\to$ CFG.

$\therefore$ A language is context-free if and only if some PDA recognizes it.

3. Non-context-free languages.

Yes – there is a pumping lemma.

<u>Idea</u>: same as before.

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

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

(1) for each $i \geq 0$, $uv^ixy^iz \in L$,

(2) $|vy| > 0$ // like the condition $|y|>0$,

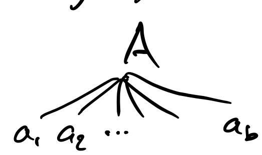
(3) $|vxy| \leq p$ // like $|xy| \leq p$ before.

<u>Proof idea</u>: CFLs have CFGs with a finite number of variables. Sufficiently long strings have derivations that always use some variable twice. If we repeat a variable, this creates a "loop" we can pump.

$$A \Rightarrow 0A1 \Rightarrow 01$$

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 0011.$$

<u>Proof.</u> (PL for CFLs). Let G be a CFG for a CFL A. Let b be the maximum number of symbols on the righthand side of a rule.



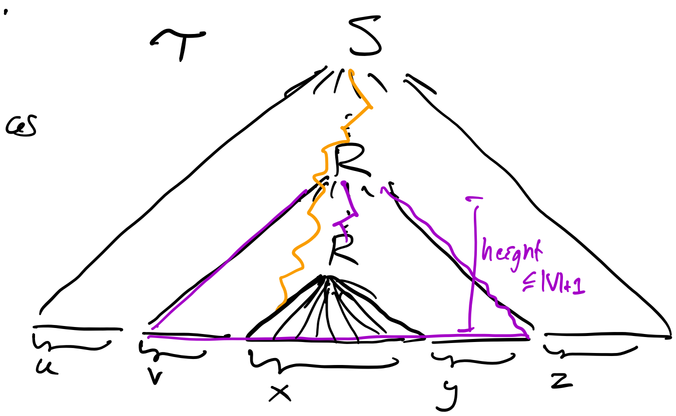
Thus any parse tree has at most b nodes at level 1, b^2 at level 2.

b^h nodes at level h, and so on.

Set our pumping length $p = b^{|V|+1}$, where $|V|$ is the number of variables. Any string $s \in A$ of length at least $p = b^{|V|+1}$ must have parse trees with height at least $|V|+1$. (Because parse trees with height at most $|V|$ have at most $b^{|V|}$ leaves.)

Let τ be some parse tree for s. Because τ has height at least $|V|+1$, there is some path of length $|V|+1$ w/ $|V|+2$ nodes.

Thus some variable appears twice. Call this variable R, let R be the first variable that repeats as we go from the bottom up.



Divide s into u, v, x, y, z according to this picture.

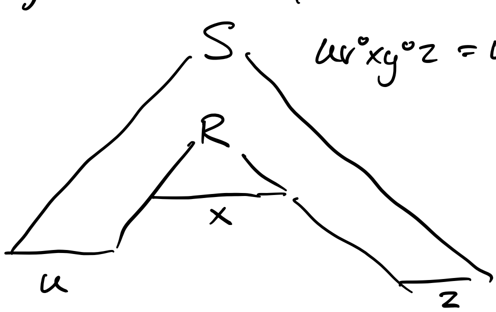
Now, I claim our three PL conditions hold.

(2) $|vy| > 0$. // holds because we can assume w.l.o.g. that τ is the shortest parse tree for s, and thus some symbol descends from the top R but not the lower R.

(3) $|vxy| \leq p$. // We find a repeated variable R in the bottom $|V|+1$ levels because there are only $|V|$ variables. Thus the height of the big tree rooted at R is at most $|V|+1$ and it has at most $p = b^{|V|+1}$ leaves.

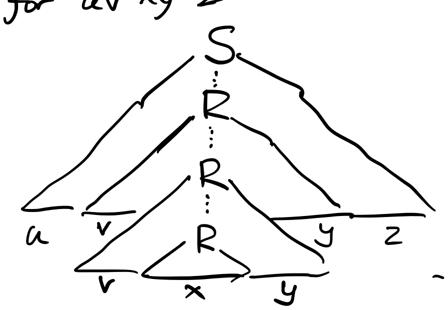
(1) $uv^ixy^iz \in A$ for all $i \geq 0$. Why?

If $i=0$, see the parse tree:



$uv^0xy^0z = uxz$

if $i=2$, this is the parse tree for uv^2xy^2z.



We can do this repetition to create a parse tree for uv^ixy^iz, $i \geq 0$. ∎

Next up: Turing Machines!

Recall: HW 4 due Monday, 11:59 PM EST. (7/26)

Reading: Sipser 2.2 (CFG = PDA)
Sipser 2.3 (CFPL).