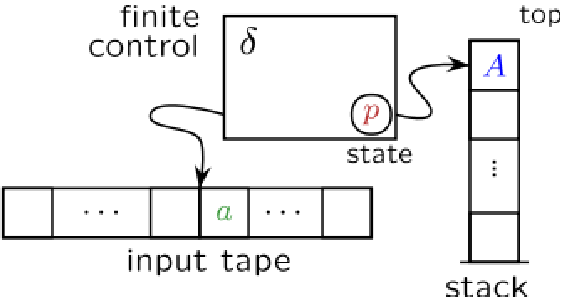
Formal Languages, Automata and Computation

Pushdown Automata

Properties of CFLs

PUSHDOWN AUTOMATA-SUMMARY

- Pushdown automata (PDA) are abstract automata that accept all context-free languages.
- PDAs are essentially NFAs with an additional infinite stack memory.
  - (Or NFAs are PDAs with no additional memory!)



PDA TO CFG

LEMMA

If a PDA recognizes some language, then it is context free.

PROOF IDEA

Create from P a CFG G that generates all strings that P accepts, i.e., G generates a string if that string takes PDA from the start state to some accepting state.

PDA TO CFG–PRELIMINARIES

Let us modify the PDA P slightly

- The PDA has a single accept state q_{accept}
  - Easy – use additional $\epsilon, \epsilon \to \epsilon$ transitions.
- The PDA empties its stack before accepting.
  - Easy – add an additional loop to flush the stack.

PDA to CFG-Preliminaries

More modifications to the PDA P:

- Each transition either pushes a symbol to the stack or pops a symbol from the stack, but not both!.
  1. Replace each transition with a pop-push, with a two-transition sequence.
     - For example replace $a, b \rightarrow c$ with $a, b \rightarrow \epsilon$ followed by $\epsilon, \epsilon \rightarrow c$, using an intermediate state.
  2. Replace each transition with no pop-push, with a transition that pops and pushes a random symbol.
     - For example, replace $a, \epsilon \rightarrow \epsilon$ with $a, \epsilon \rightarrow x$ followed by $\epsilon, x \rightarrow \epsilon$, using an intermediate state.

PDA TO CFG–PRELIMINARIES

- For each pair of states p and q in P, the grammar with have a variable A_{pq}.
  - A_{pq} generates all strings that take P from p with an empty stack, to q, leaving the stack empty.
  - A_{pq} also takes P from p to q, leaving the stack as it was before p!

PDA to CFG–Preliminaries

- Let x be a string that takes P from p to q with an empty stack.
  - First move of the PDA should involve a push! (Why?)
  - Last move of the PDA should involve a pop! (Why?)

PDA TO CFG–PRELIMINARIES

- There are two cases:
  1. Symbol pushed after p, is the same symbol popped just before q
  2. If not, that symbol should be popped at some point before! (Why?)
- First case can be simulated by rule $A_{pq} \rightarrow aA_{rs}b$
  - Read a, go to state r, then transit to state s somehow, and then read b.
- Second case can be simulated by rule $A_{pq} \rightarrow A_{pr}A_{rq}$
  - r is the state the stack becomes empty on the way from p to q

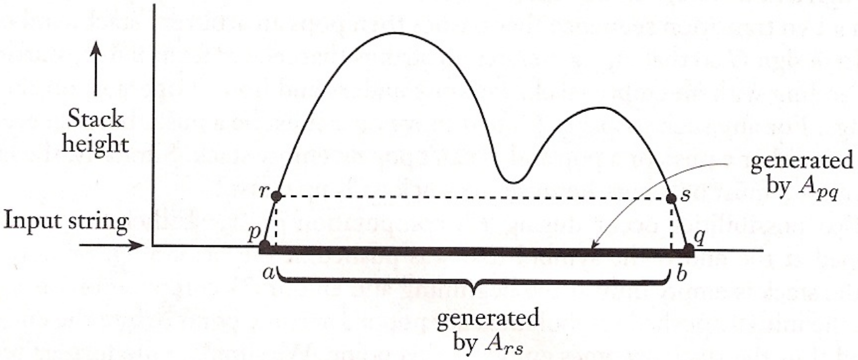
PDA TO CFG – PROOF

- Assume $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$.
- The variables of G are $\{A_{pq} \mid p, q \in Q\}$
- The start variable is $A_{q_0, q_{accept}}$
- The rules of G are as follows:
  - For each $p, q, r, s \in Q, t \in \Gamma$, and $a, b \in \Sigma_\epsilon$, if
    - $\delta(p, a, \epsilon)$ contains (r, t) and
    - $\delta(s, b, t)$ contains (q, ϵ)

    Add rule $A_{pq} \rightarrow aA_{rs}b$ to G.
  - For each $p, q, r \in Q$, add rule $A_{pq} \rightarrow A_{pr}A_{rq}$ to G.
  - For each, $p \in Q$, add the rule $A_{pp} \rightarrow \epsilon$ to G.

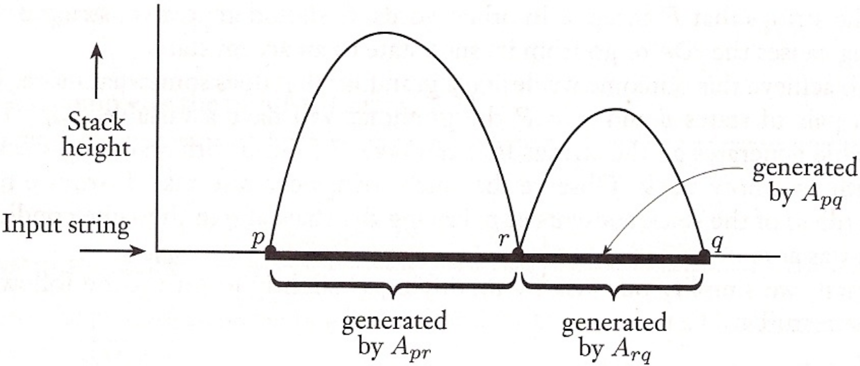
PDA to CFG Intuition

- PDA computation for $A_{pq} \to aA_{rs}b$



PDA to CFG Intuition

- PDA computation for $A_{pq} \to A_{pr}A_{rq}$



PDA to CFG Proof (cont'd)

Claim

If A_{pq} generates x, then x can bring P from p with empty stack to q with empty stack.

- Basis Case: Derivation has 1 step.
  - This can only be possible with a production of the sort $A_{pp} \rightarrow \epsilon$. We have such a rule!
- Assume true for derivations of length at most k, $k \geq 1$
  - Suppose that $A_{pq} \stackrel{*}{\Rightarrow} x$ with $k + 1$ steps. The first step in this derivation would either be $A_{pq} \rightarrow aA_{rs}b$ or $A_{pq} \rightarrow A_{pr}A_{rq}$
- We handle these cases separately.

PDA TO CFG PROOF (CONT'D)

Case $A_{pq} \to aA_{rs}b$:

- $A_{rs} \overset{*}{\Rightarrow} y$ in k steps where $x = ayb$ and by induction hypothesis, P can go from r to s with an empty stack.
- If P pushes t onto the stack after p, after processing y it will leave t back on stack.
- Reading b will have to pop the t to leave an empty stack.
- Thus, x can bring P from p to q with an empty stack.

PDA to CFG Proof (cont'd)

Case $A_{pq} \rightarrow A_{pr}A_{rq}$

- Suppose $A_{pr} \stackrel{*}{\Rightarrow} y$ and $A_{rq} \stackrel{*}{\Rightarrow} z$, where $x = yz$.
- Since these derivations are at most k steps, before p and after r we have empty stacks, and thus also after q.
- Thus x can bring P from p to q with an empty stack.

PDA to CFG Proof (cont'd)

Claim

If x can bring P from p to q with empty stack, $A_{pq} \stackrel{*}{\Rightarrow} x$.

- Basis Case: Suppose PDA takes 0 steps.
  - It should stay in the same state. Since we have a rule in the grammar $A_{pp} \rightarrow \epsilon$, $A_{pp} \stackrel{*}{\Rightarrow} \epsilon$.
- Assume true for all computations of P of length at most k, $k \geq 0$.
  - Suppose with x, P can go from p to q with an empty stack. Either the stack is empty only at the beginning and at the end, or it becomes empty elsewhere, too.
- We handle these two cases separately.

PDA to CFG Proof (cont'd)

Case: Stack is empty only at the beginning and at the end of a derivation of length $k + 1$

- Suppose $x = ayb$. a and b are consumed at the beginning and at the end of the computation (with t being pushed and popped).
- P takes $k - 2$ steps on y.
- By hypothesis, $A_{rs} \stackrel{*}{\Rightarrow} y$ where $(r, t) \in \delta(q, a, \epsilon)$ and $(q, \epsilon) \in \delta(s, b, t)$.
- Thus, using rule $A_{pq} \to aA_{rs}b$, $A_{pq} \stackrel{*}{\Rightarrow} x$.

PDA to CFG Proof (cont'd)

Case: Stack becomes empty at some intermediate stage in the computation of x

- Suppose $x = yz$, such that P has the stack empty after consuming y.
- By induction hypothesis $A_{pr} \stackrel{*}{\Rightarrow} y$ and $A_{rq} \stackrel{*}{\Rightarrow} z$ since P takes at most k steps on y and z.
- Since rule $A_{pq} \to A_{pr}A_{rq}$ is in the grammar, $A_{pq} \stackrel{*}{\Rightarrow} x$.

Regular Languages are Context Free

Corollary

Every regular language is context free.

Proof.

Since a regular language L is reconized by a DFA and every DFA is a PDA that ignores it stack, there is a CFG for L $\square$

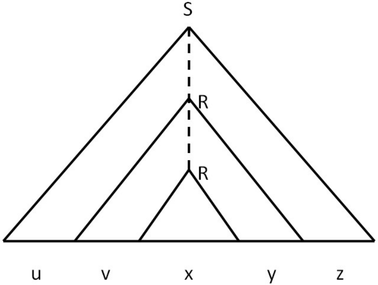
- Right-linear grammars
- Left-linear grammars

NON-CONTEXT-FREE LANGUAGES

- There are non-context-free languages.
- For example $L = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free.
  - Intuitively, once the PDA reads the a's and the matches the b's, it "forgets" what the n was, so can not properly check the c's.
- There is an analogue of the Pumping Lemma we studied earlier for regular languages.
  - It states that there is a pumping length, such that all longer strings can be pumped.
  - For regular languages, we related the pumping length to the number of states of the DFA.
  - For CFLs, we relate the pumping length to the properties of the grammar!.

Pumping Lemma for CFLs - Intuition

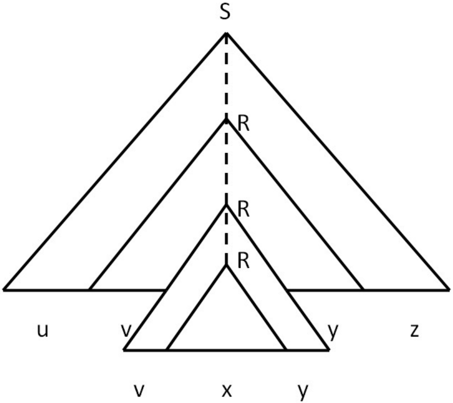
- Let s be a "sufficiently long" string in L.
- $s = uvxyz$ should have a parse tree of the following sort:



- Some variable R must repeat somewhere on the path from S to some leaf. (Why?)

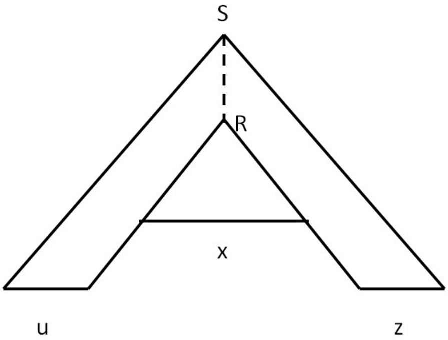
Pumping Lemma for CFLs - Intuition

- Then the string $s' = uvvxyyz$, should also be in the language.



Pumping Lemma for CFLs - Intuition

- Also the string $s'' = uxz$, should also be in the language.



Pumping Lemma for CFLs - Intuition

Lemma

If L is a CFL, then there is a number p (the pumping length) such that if s is any string in L of length at least p, then s can be divided into 5 pieces $s = uvxyz$ satisfying the conditions:

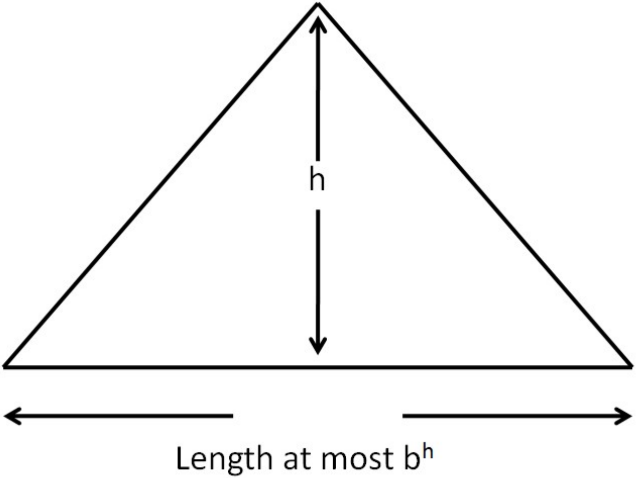
1. $|vy| > 0$
2. $|vxy| \leq p$
3. *for each* $i \geq 0$, $uv^ixy^iz \in L$

- Either v or y is not ϵ otherwise, it would be trivially true.

Proof – The Pumping Length

- Let G be the grammar for L. Let b be the maximum number of symbols of any rule in G.
  - Assume b is at least 2, that is every grammar has some rule with at least 2 symbols on the RHS.
- In any parse tree, a node can have at most b children.
  - At most b^h leaves are within h steps of the start variable.
- If the parse tree has height h, the length of the string generated is at most b^h.
- Conversely, if the string is at least $b^h + 1$ long, each of its parse trees must be at least $h + 1$ high.

PROOF – THE PUMPING LENGTH



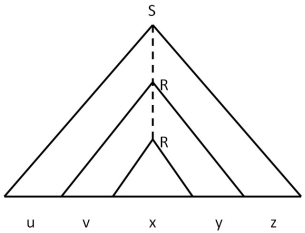
Proof - The Pumping Length

- Let $|V|$ be the number of variables in G.
- We set the pumping length $p = b^{|V|+1}$.
- If s is a string in L and $|s| \geq p$, its parse tree must be at least $|V| + 1$ high.
  - $b^{|V|+1} \geq b^{|V|} + 1$

Proof - How to Pump a String

- Let τ be the parse tree of s that has the smallest number of nodes. τ must be at least $|V| + 1$ high.
- This means some path from the root to some leaf has length at least $|V| + 1$.
- So the path has at least $|V| + 2$ nodes: 1 terminal and at least $|V| + 1$ variables.
- Some variable R must appear more than once on that path (Pigeons!)
    - Choose R as the variable that repeats among the lowest $|V| + 1$ variables on this path.

Proof - How to Choose a String



- We divide s into $uvxyz$ according to this figure.

- Upper R generates vxy while the lower R generates x.
- Since the same variable generates both subtrees, they are interchangeable!
- So all strings of the form uv^ixy^iz should also be in the language for $i \geq 0$.

Proof – How to Choose a String

- We must make sure both v and y are not both ϵ.
- If they were, then τ would not be smallest tree for s.
  - We could get a smaller tree for s by substituting the smaller tree!
- $R \stackrel{*}{\Rightarrow} vxy$.
- We chose R so that both its occurrences were within the last $|V| + 1$ variables on the path.
- We chose the longest path in the tree, so the subtree for $R \stackrel{*}{\Rightarrow} vxy$ is at most $|V| + 1$ high.
- A tree of this height can generate a string of length at most $b^{|V|+1} = p$.