Last Lecture

We began to show CFL = PDA

- **Theorem 1**. Every context-free language is accepted by some PDA.
- **Theorem 2**. For every PDA M, the language L(M) is context-free.

We showed how a PDA could be constructed from a CFL. Given a CFG $G=(V,T,P,S)$, we define a PDA $M=(\{q\},T, T \cup V, \delta,q,S)$, with δ given by

– If $A \in V$, then $\delta(q,\Lambda,A) = \{ (q,\alpha) \mid A \to \alpha \text{ is in } P\}$
– If $a \in T$, then $\delta(q,a,a) = \{ (q,\Lambda) \}$

1. The stack symbols of the new PDA contain all the terminal and non-terminals of the CFG
2. There is only 1 state in the new PDA
3. Add transitions on Λ, one for each production
4. Add transitions on $a \in T$, one for each terminal.

Transitions simulate left-most derivation

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$$

1.	$\delta(q, \Lambda, S) = (q, SS)$		$S \to SS$
2.	$\delta(q, \Lambda, S) = (q, (S))$		$S \to (S)$
3.	$\delta(q, \Lambda, S) = (q, \Lambda)$		$S \to \Lambda$
4.	$\delta(q, (, () = (q, \Lambda)$		
5.	$\delta(q,),)) = (q, \Lambda)$		

```
(q, "(())()" ,S      )   |- [1]
(q, "(())()" ,SS     )   |- [2]
(q, "(())()" ,(S)S   )   |- [4]
(q, "())()"  ,S)S    )   |- [4]
(q, "())()"  ,(S))S  )   |- [4]
(q, "))()"   ,S))S   )   |- [3]
(q, "))()"   ,))S    )   |- [5]
(q, ")()"    ,)S     )   |- [5]
(q, "()"     ,S      )   |- [2]
(q, "()"     ,(S)    )   |- [4]
(q, ")"      ,S)     )   |- [3]
(q, ")"      ,)      )   |- [5]
(q, ε        ,ε      )
```

Note there is an entry in δ for each terminal and non-terminal symbol. The stack operations mimic a top down parse, replacing Non-terminals with the rhs of a production.

Proof Outline

To prove that every string of L(G) is accepted by the PDA M, prove the following more general fact:

If $S \Rightarrow_{\text{left-most}}^{*} \alpha$ then (q,uv,S) |-* (q,v,β)

where $\alpha = u\beta$ is the "leftmost factorization" of α (u is the longest prefix of α that belongs to T^*, i.e. all terminals).

For example: if α = abcWdXa then u = abc, and β = WdXa, since the next symbol after abc is W$\in$V (a non-terminal or Λ)

$S \Rightarrow_{lm}^{*}$ abcW... then (q, abcV,S) |-* (q,V, W...)

The proof is by induction on the length of the derivation of α.

We also need to prove that every string accepted by M belongs to L(G). Again, to make induction work, we need to prove a slightly more general fact:

If $(q,w,A) \vdash^{*} (q, \Lambda, \Lambda)$, then $A \Rightarrow^{*} w$

For all Stacks A, letting A = Start we have our proof.

This time we induct on the length of execution of M that leads from the ID (q,w,A) to (q, Λ , Λ).

A Grammar from a PDA

Assume the $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0)$ is given, and that it accepts by empty stack. Consider execution of M on an accepted input string.

If at some point of the execution of M the stack is $Z\zeta$ (Z is on top, ζ is the rest of stack)

In terms of instantaneous descriptions

$(\text{state}_i, \text{input}, Z\zeta) \vdash \ldots$

Then we know that eventually the stack will be ζ.

Why? Because we assume the input is accepted, and M accepts by empty stack, so eventually Z must be removed from the stack

$$(state_i,\ \alpha X,\ \ Z\zeta) \ \ \vdash^* \ \ (state_j,\ X,\ \ \zeta)$$

The sequence of moves between these two instants is the “net popping” of Z from the stack.

During this sequence of moves, the stack may grow and shrink several times, some input will be consumed (the α), and M will pass through a sequence of states, from $state_i$ to $state_j$.

Net Popping

Net popping is fundamental for the construction of a CFG G equivalent to M.

We will have a variable (Non-terminal) [qZp] in the CFG G for every triple in (q,Z,p) $\in Q\times\Gamma\times Q$ from the PDA. Recall

1. Q is the set of states
2. Γ Is the set of stack symbols

We want the rhs of a production whose lhs is [qZp] to generate precisely those strings w $\in \Sigma^*$ such that M can move from q to p while reading the input w and doing the net popping of Z. A production like [qZp] -> ?

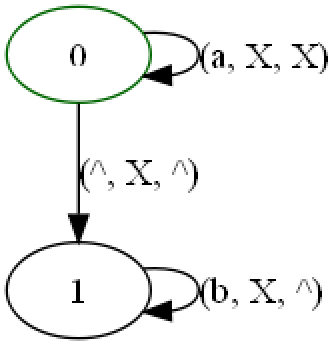
This can be also expressed as (q,w,Z) |-* (p, Λ , Λ)

Productions of G correspond to transitions of M.

If $(p,\zeta) \in \delta(q,a,Z)$, then there is one or more corresponding productions, depending on complexity of ζ.

1. If $\zeta = \Lambda$, we have $[qZp] \rightarrow a$
2. If $\zeta = Y$, we have $[qZr] \rightarrow a[pYr]$ for every state r
3. If $\zeta = YY'$ we have $[qZs] \rightarrow a[pYr][rY's]$, for every pair of states r and s.
4. You can guess the rule for longer ζ.

Example



$Q = \{0,1\}$

$S = \{a,b\}$

$\Gamma = \{X\}$

$\delta(0,a,X) = \{\ (0,X)\ \}$

$\delta\ (0,\Lambda,X) = \{\ (1,\Lambda)\ \}$

$\delta\ (1,b,X) = \{\ (1,\Lambda)\ \}$

$Q_{0\ =\ 0}$

$Z_0 = X$

$F = \{\}$, accepts by empty stack

Non-terminals

$(q,Z,p) \in Q\times\Gamma\times Q$

(0,'X',0)

(0,'X',1)

(1,'X',0)

(1,'X',1)

Productions, At least one from each element in delta

$(p,z) \in \delta(q,a,Z)$

(0,a,X,0,X)

$(1,b,X,1,\Lambda)$

$(0,\Lambda,X,1,\Lambda)]$

0X0 -> a 0X0

0X1 -> a 0X1

1X1 -> b

$0X1 \text{ -> } \Lambda$

CFL Pumping Lemma

A *CFL pump* consists of two non-overlapping substrings that can be pumped simultaneously while staying in the language.

Precisely, two substrings u and v constitute a CFL pump for a string w of L when

1. $uv \neq \Lambda$ (which means that at least one of u or v is not empty)
2. And we can write $w=xuyvz$, so that for every $i \geq 0$
3. $xu^iyv^iz \in L$

Pumping Lemma

Let L be a CFL. Then there exists a number n (depending on L) such that every string w in L of length greater than n contains a CFL pump.

Moreover, there exists a CFL pump such that (with the notation as above), $|uyv| \leq n$.

For example, take $L = \{0^i 1^i \mid i \geq 0\}$: there are no (RE) pumps in any of its strings, but there are plenty of CFL pumps.

The pumping Lemma Game

We want to prove L is not context-free. For a proof, it suffices to give a winning strategy for this game.

1. The demon first plays n.
2. We respond with $w \in L$ such that $|w| \geq n$.
3. The demon factors w into five substrings, $w=xuyvz$, with the proviso that $uv \neq \Lambda$ and $|uyv| \leq n$
4. Finally, we play an integer $i \geq 0$, and we win if $xu^iyv^iz \notin L$.

Example 1

We prove that $L = \{0^i 1^i 2^i \mid i \geq 0\}$ is not context-free.

In response to the demon's n, we play $w=0^n1^n2^n$.

The middle segment uyv of the demon's factorization of w = xuyvz, cannot have an occurrence of both 0 and 2 (because we can assume $|uyv| \leq n$).

Suppose 2 does not occur in uyv (the other case is similar).

1. We play i = 0.
2. Then the total number of 0's and 1's in w_0=xyz will be smaller than 2n,
3. while the number of 2's in w_0 will be n.
4. Thus, $w_0 \notin L$.

Example 2

Let L be the set of all strings over {0,1} whose length is a perfect square.

1. The demon plays n
2. We respond with $w = 0^{n \wedge 2}$
3. The demon plays a factorization $0^{n \wedge 2} = xuyvz$ with $1 \leq |uyv| \leq n$.
4. We play i=2.
5. The length of the resulting string $w_2 = xu^2yv^2z$ is between n^2+1 and n^2+n.
6. In that interval, there are no perfect squares, so $w_2 \notin L$.

Proof of the pumping lemma

Strategy in several steps

1. Define fanout
2. Define height yield
3. Prove a lemma about height yield
4. Apply the lemma to prove pumping lemma

Fanout

Let fanout(G) denote the maximal length of the rhs of any production in the grammar G.

E.g. For the Grammar

$S \rightarrow S\ S$

$S \rightarrow (\ S\ )$

$S \rightarrow \varepsilon$

The fanout is 3

Height Yield

The proof of Pumping Lemma depends on this simple fact about parse trees.

The *height* of a tree is the maximal length of any path from the root to any leaf.

Lemma. If a parse tree of G has height h, than its yield has length at most $\text{fanout}(G)^h$

Proof. Induction on h

qed

The actual Proof

The constant n for the grammar G is $\text{fanout}(G)^{|V|}$ where V is the set of variables of G.

Suppose $w \in L(G)$ and $|w| \geq n$.

Take a parse tree of w with the smallest possible number of nodes.

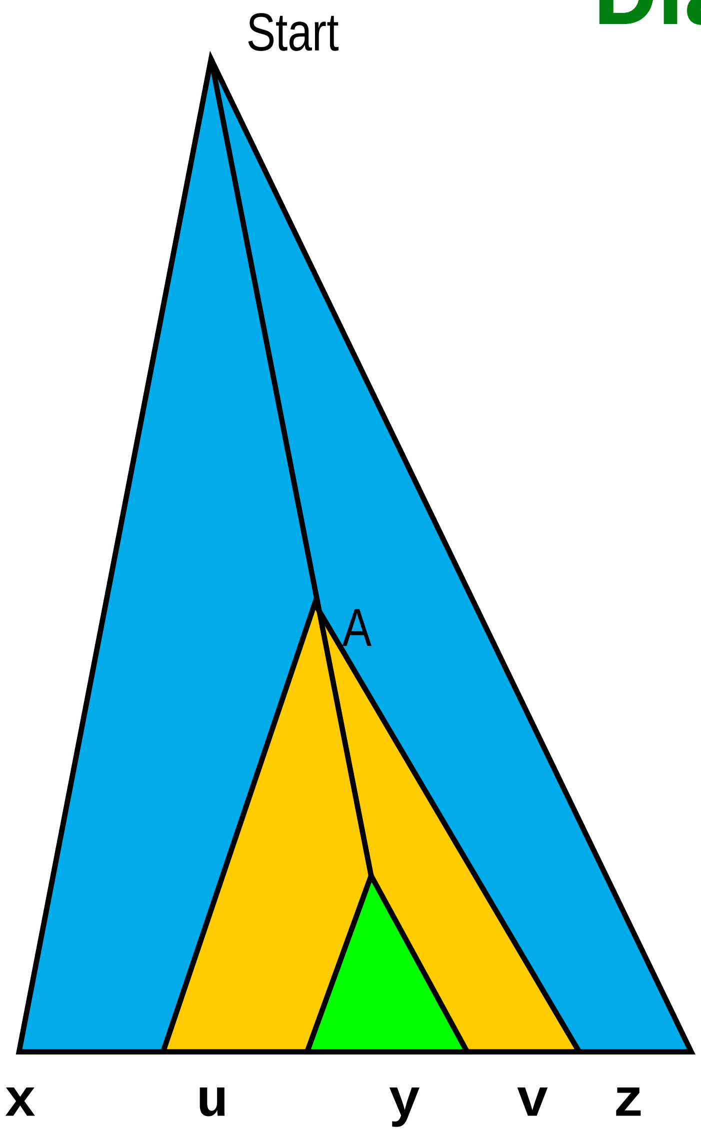
By the Height-Yield Lemma, any parse tree of w must have height $\geq |V|$.

Therefore, there must be two occurrences of the same variable on a path from root to a leaf.

Consider the last two occurrences of the same variable (say A) on that path.

They determine a factorization of the yield w=xuyvz as in the picture on the next slide

Diagram



We have

$S \Rightarrow^* xAz$

$A \Rightarrow^* uAv$

$A \Rightarrow^* y$

so clearly $S \Rightarrow^* xu^iyv^iz$ for any $i \geq 0$.

We also need to check that $uv \neq \Lambda$. Indeed, if $uv = \Lambda$, we can get a smaller parse tree for the same w by ignoring the productions “between the two As”. But we have chosen the smallest possible parse tree for w! Which leads to a Contradiction.

Finally, we need to check that $|uyv| \leq n$. This follows from the Height-Yield Lemma because the nodes on our chosen path from the first depicted occurrence of A, onward, are labeled with necessarily distinct variables.

qed