# Harvard CS 121 and CSCI E-121

# Lecture 11: General Context-Free Recognition

Harry Lewis

October 10, 2013

**Reading:** Sipser, Section 2.3 and Section 2.1 (material on Chomsky Normal Form).

# Pumping Lemma for CFLs

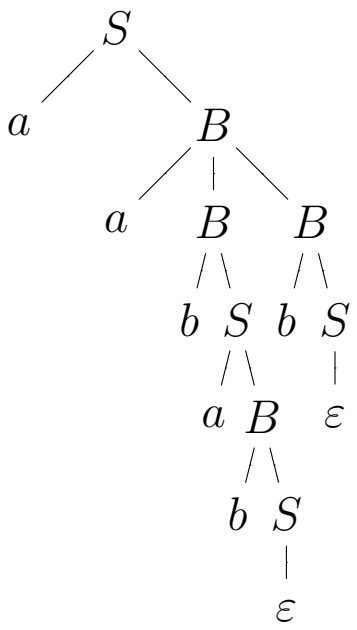
**Lemma:** If $L$ is context-free, then there is a number $p$ (the pumping length) such that any $s \in L$ of length at least $p$ can be divided into $s = uvxyz$, where

1. $uv^ixy^iz \in L$ for every $i \geq 0$,
2. $v \neq \varepsilon$ or $y \neq \varepsilon$, and
3. $|vxy| \leq p$.

- **Proposition:** $\{a^n b^n c^n : n \geq 0\}$ is not CF.
- **Corollary:** CFLs not closed under intersection (why?)
- **Corollary:** CFLs not closed under complement (why?)

# Recall: Parse Trees

$$
\begin{aligned}
S &\rightarrow aB \mid bA \mid \varepsilon \\
A &\rightarrow aS \mid bAA \\
B &\rightarrow bS \mid aBB
\end{aligned}
$$

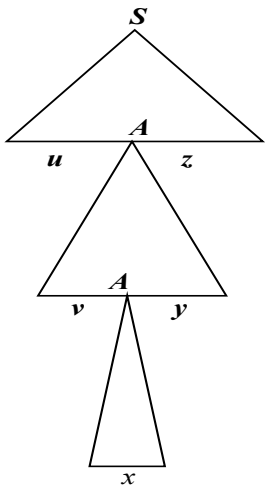



Parse tree for $aabab$, the "yield" of the tree

Height = max length path from $S$ to a terminal symbol = 6 in above example.

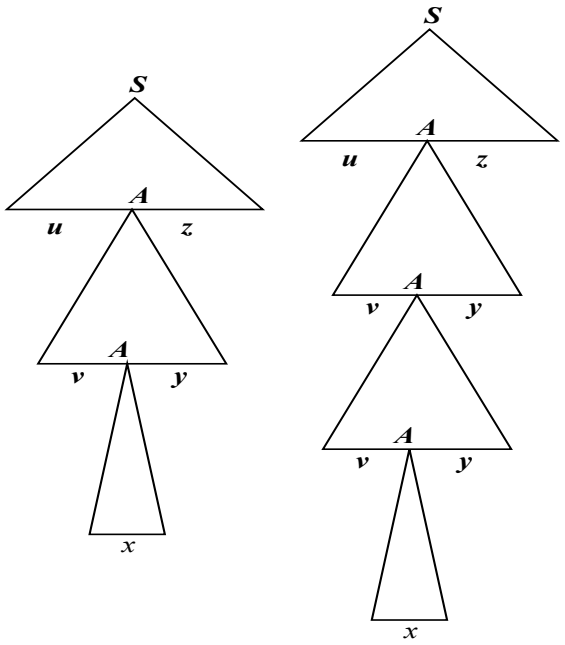
# Proof Idea for Pumping Theorem

Show that there exists a $p$ such that any string $s$ of length $\geq p$ has a parse tree of the form:



# Proof Idea for Pumping Theorem

Show that there exists a $p$ such that any string $s$ of length $\geq p$ has a parse tree of the form:



# Finding "Repetition" in a big parse tree

- Since RHS of rules have bounded length, long strings must have tall parse trees.
- A tall parse tree must have a path with a repeated nonterminal.
- Let $p = b^m + 1$, where:

  $b =$ max length of RHS of a rule

  $m =$ # of variables

- Suppose $T$ is the smallest parse tree for a string $s \in L$ of length at least $p$. Then

  Let $h =$ height of $T$. Then $b^h \geq p = b^m + 1$,

  $\Rightarrow h > m$,

  $\Rightarrow$ Path of length $h$ in $T$ has a repeated variable.

# Final annoying details

- **Q:** Why is $v$ or $y$ nonempty?
- **Q:** How to ensure $|vxy| \leq p$?

# The converse of the CF Pumping Lemma is False

Some non-context-free languages satisfy conclusion of Pumping Lemma, e.g. ?

# Some Other CF Closure Properties

Let $L_1/L_2 = \{w : wx \in L_1 \text{ for some } x \in L_2\}$. Then

- If $L$ is CF and $R$ is regular then $L/R$ is CF

# Context-Free Recognition

- **Goal:** Given CFG $G$ and string $w$ to determine if $w \in L(G)$
- **First attempt:** Construct a PDA $M$ from $G$ and run $M$ on $w$.
- **Brute-Force Method:**

  Check all parse trees of height up to some upper limit depending on $G$ and $|w|$

  **Exponentially costly**

- **Better:**

1. Transform $G$ into Chomsky normal form (CNF) (once for $G$)
2. Apply a special algorithm for CNF grammars

   (once for each $w$)

# Chomsky Normal Form

**Def:** A grammar is in Chomsky normal form if

- the only possible rule with $\varepsilon$ as the RHS is $S \to \varepsilon$

  (Of course, this rule occurs iff $\varepsilon \in L(G)$)

- Every other rule is of the form

1) $X \to YZ$

   where $X, Y, Z$ are variables, and $Y, Z \neq S$

2) $X \to \sigma$

   where $X$ is variable and $\sigma$ is a single terminal symbol

# Transforming a CFG into Chomsky Normal Form

**Definitions**:

- $\varepsilon$-rule: one of the form $X \to \varepsilon$
- Long Rule: one of the form $X \to \alpha$ where $|\alpha| > 2$.
- Unit Rule : One of the form $X \to Y$

  where $X, Y \in V$
- Terminal-Generating Rule: one of the form $X \to \alpha$

  where $\alpha \notin V^*$ and $|\alpha| > 1$ ($\alpha$ has at least one terminal)

# Eliminate non-Chomsky-Normal-Form Rules In Order:

1. All $\varepsilon$-rules, except maybe $S \to \varepsilon$

2. All unit rules

3. All long rules

4. All terminal-generating rules

- While eliminating rules of type $j$, we make sure not to reintroduce rules of type $i < j$.

# Eliminating $\varepsilon$-Rules

0. Ensure start variable does not appear on the RHS of any rule (by adding new start variable if necessary).

1. To eliminate $\varepsilon$-rules, repeatedly do the following:

   a. Pick a $\varepsilon$-rule $Y \to \varepsilon$ and remove it.

   b. Given a rule $X \to \alpha$ where $\alpha$ contains $n$ occurrences of $Y$, replace it with $2^n$ rules in which $0, \ldots, n$ occurrences are replaced by $\varepsilon$. (Do not add $X \to \varepsilon$ if previously removed.)

   e.g.

$$X \to aYZbY \qquad \Rightarrow$$

(Why does this terminate?)

# Eliminating Unit and Long Rules

2. To eliminate unit rules, repeatedly do the following:
   a. Pick a unit rule $A \to B$ and remove it.
   b. For every rule $B \to u$, add rule $A \to u$ unless this is a unit rule that was previously removed.

3. To eliminate long rules, repeatedly do the following:
   a. Remove a long rule $A \to u_1 u_2 \cdots u_k$, where each $u_i \in V \cup \Sigma$ and $k \geq 3$.
   b. Replace with rules $A \to u_1 A_1, A_1 \to u_2 A_2, \ldots, A_{k-2} \to u_{k-1} u_k$, where $A_1, \ldots, A_{k-2}$ are newly introduced variables used only in these rules.

# Eliminating Terminal-Generating Rules

4. To eliminate terminal-generating rules:

   a. For each terminal $a$ introduce a new nonterminal $A$.

   b. Add the rules $A \rightarrow a$

   c. "Capitalize" existing rules, e.g.

      replace $X \rightarrow aY$
      with $X \rightarrow AY$

# Example of Transformation to Chomsky Normal Form

Starting grammar:

$$S \to XX$$
$$X \to aXb|\varepsilon$$

# Benefit of CNF for Deciding if $w \in L(G)$

- **Observation:** If $S \Rightarrow XY \Rightarrow^* w$, then $w = uv$, $X \Rightarrow^* u$, $Y \Rightarrow^* v$ where $u, v$ are *strictly shorter* than $w$.

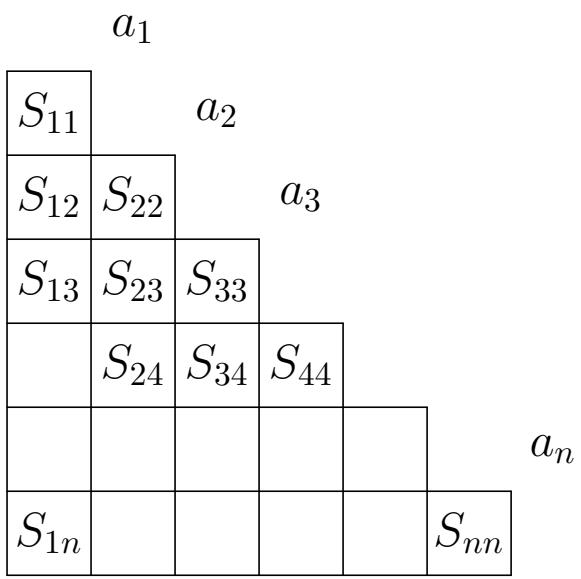
- **Divide and Conquer:** can decide whether $S$ yields $w$ by recursively determining which variables yield substrings of $w$.

- **Dynamic Programming:** record answers to all subproblems to avoid repeating work.

# Determining $w \in L(G)$, for $G$ in CNF

Let $w = a_1 \ldots a_n$, $a_i \in \Sigma$.
Determine sets $S_{ij}$ $(1 \leq i \leq j \leq n)$:

$$S_{ij} = \{X : X \stackrel{*}{\Rightarrow} a_i \ldots a_j, X \text{ variable of } G\}$$




- $w \in L(G)$ iff start symbol $\in S_{1n}$

# Filling in the Matrix

- Calculate $S_{ij}$ by induction on $j - i$

$(j - i = 0)$ $S_{ii}$ = $\{X : X \to a_i$ is a rule of $G\}$

$(j - i > 0)$ $X \in S_{ij}$ iff $\exists$ rule $X \to YZ$

$$\exists k : i \leq k < j$$

such that $Y \in S_{ik}$

$$Z \in S_{k+1,j}$$

e.g. $w = abaabb$

# The Chomsky Normal Form Parsing Algorithm

for $i \leftarrow 1$ to $n$ do

$S_{ii} = \{X : X \rightarrow a_i \text{ is a rule } \}$

for $d \leftarrow 1$ to $n-1$ do

for $i \leftarrow 1$ to $n-d$ do

$$S_{i,i+d} \leftarrow \bigcup_{j=i}^{i+d-1} \left\{ \begin{array}{l} X : X \rightarrow YZ \text{ is a rule,} \\ Y \in S_{ij}, Z \in S_{j+1,i+d} \end{array} \right\}$$

Complexity: $\mathcal{O}(n^3)$.

# Of what does this triply nested loop remind you?

# Of what does this triply nested loop remind you?

- Matrix Multiplication
- In fact, better matrix multiplication algorithms yield (asymptotically) better general context free parsing algorithms
- Fastest known matrix multiplication algorithm uses $\mathcal{O}(n^{2.373})$ operations (Stothers '11 and Williams '11, improving Coppersmith & Winograd '89).

# CF Recognition in Practice

In compilers for programming languages, parsing is done via algorithms that correspond to *Deterministic* PDAs (DPDAs).

- What is the advantage over CNF algorithm?

Our CFG$\mapsto$PDA construction is highly nondeterministic.

- Constructs parse tree "top-down" from start variable; input might not be used until the very end.

A dual, "bottom-up" approach sometimes yields a DPDA.

- Construct parse tree starting from input string.
- Yields a DPDA if $G$ is a "DCFG".
- We can design programming languages to ensure the DCFG property. (DCFLs are a strict subset of CFLs.)

# The Top-Down CFG $\rightarrow$ PDA construction, revisited

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

  "Start by putting $S\$$ on the stack, & go to $q_{\text{loop}}$"

- for each $A \in V$,
  $\delta(q_{\text{loop}}, \varepsilon, A) = \{(q_{\text{loop}}, w) : A \rightarrow w \text{ is a rule of } R\}$

  "Remove a variable from the top of the stack and replace it with a corresponding righthand side"

- for each $\sigma \in \Sigma$, $\delta(q_{\text{loop}}, \sigma, \sigma) = \{(q_{\text{loop}}, \varepsilon)\}$

  "Pop a terminal symbol from the stack if it matches the next input symbol"

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

  "Go to accept state if stack contains only $\$$."

# The Dual Bottom-Up CFG → PDA Construction

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

  "Start by putting $\$$ on the stack, & go to $q_{\text{loop}}$"

- for each $\sigma \in \Sigma$ , $\delta(q_{\text{loop}}, \sigma, \varepsilon) = \{(q_{\text{loop}}, \sigma)\}$

  "Shift input symbols onto the stack"

- for each $A \in V$, $\delta(q_{\text{loop}}, \varepsilon, w^R) = \{(q_{\text{loop}}, A) : A \to w \in R\}$

  "Reduce right-hand sides on the stack to corresponding left-hand sides"

- $\delta(q_{\text{loop}}, \varepsilon, S\$) = \{(q_{\text{accept}}, \varepsilon)\}$

  "Accept if the stack consists just of $S$ above the bottom-marker"

# Summary of Context-Free Recognition

- CFL to PDA reduction yields nondeterministic automaton
- By use of Chomsky Normal Form and dynamic programming, there is a general $O(n^3)$ non-stack-based algorithm
- The deterministic CFLs are the languages recognizable by deterministic PDAs
- E.g. $\{wcw^R : w \in \{a, b\}^*\}$ is a deterministic CFL but $\{ww^R : w \in \{a, b\}^*\}$ (even palindromes) is not
- Methods used in compliers are deterministic stack-based algorithms, requiring that the source language be deterministic CF or a special type of deterministic CF (LR$(k)$, etc.).

# Beyond Context-Free Languages

- A **Context-Sensitive Grammar** allows rules of the form $\alpha \to \beta$, where $\alpha$ and $\beta$ are strings and $|\alpha| \leq |\beta|$, so long as $\alpha$ contains at least one nonterminal.

- The possibility of using rules such as $aB \to aDE$ makes the grammar "sensitive to context"

- Is there an algorithm for determining whether $w \in L(G)$ where $G$ is a CSG?

- But the field moved, and now we also move, from syntactic structures to computational difficulty.