

# Harvard CS 121 and CSCI E-207

## Lecture 12: General Context-Free Recognition

Harry Lewis

October 13, 2009

**Reading:** Sipser, pp. 119-128.

## 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  $S \rightarrow \varepsilon$

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

- Every other rule is of the form

1)  $X \rightarrow YZ$

where  $X, Y, Z$  are variables

2)  $X \rightarrow \sigma$

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  $X \rightarrow \varepsilon$
- Long Rule: one of the form  $X \rightarrow \alpha$  where  $|\alpha| > 2$ .
- Unit Rule : One of the form  $X \rightarrow Y$   
where  $X, Y \in V$
- Terminal-Generating Rule: one of the form  $X \rightarrow \alpha$   
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  $S \rightarrow \varepsilon$
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  $Y \rightarrow \varepsilon$  and remove it.
  - b. Given a rule  $X \rightarrow \alpha$  where  $\alpha$  contains  $n$  occurrences of  $Y$ , replace it with  $2^n$  rules in which  $0, \dots, n$  occurrences are replaced by  $\varepsilon$ . (Do not add  $X \rightarrow \varepsilon$  if previously removed.)

e.g.

$$X \rightarrow aY ZbY \quad \Rightarrow$$

(Why does this terminate?)

## Eliminating Unit and Long Rules

2. To eliminate unit rules, repeatedly do the following:
  - a. Pick a unit rule  $A \rightarrow B$  and remove it.
  - b. For every rule  $B \rightarrow u$ , add rule  $A \rightarrow u$  unless this is a unit rule that was previously removed.
  
3. To eliminate long rules, repeatedly do the following:
  - a. Remove a long rule  $A \rightarrow u_1u_2 \cdots u_k$ , where each  $u_i \in V \cup \Sigma$  and  $k \geq 3$ .
  - b. Replace with rules  $A \rightarrow u_1A_1, A_1 \rightarrow u_2A_2, \dots, A_{k-2} \rightarrow u_{k-1}u_k$ , where  $A_1, \dots, A_{k-2}$  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:

$$S \rightarrow XX$$

$$X \rightarrow aXb \mid \varepsilon$$

## 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.



## Filling in the Matrix

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

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

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

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

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
- Strassen's algorithm requires  $O(n^{2.81})$  instead of  $O(n^3)$  multiplications

## 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.  $\{w c w^R : w \in \{a, b\}^*\}$  is a deterministic CFL but  $\{w w^R : w \in \{a, b\}^*\}$  (even palindromes) is not
- Methods used in compilers 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  $\alpha \rightarrow \beta$ , 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  $aB \rightarrow aDE$  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