COMS W3261
Computer Science Theory
Lecture 10
CNF, Pumping Lemma, CYK Algorithm

Alexander Roth

2014 – 09 – 29

## Outline

1. Eliminating $\epsilon$-productions from a CFG
2. Eliminating unit productions from a CFG
3. Putting a CFG into Chomsky normal form
4. Pumping lemma for CFL's
5. Cocke-Younger-Kasami algorithm

# 1 Eliminating $\epsilon$-productions from a CFG

- If a language $L$ has a CFG, then $L - \{\epsilon\}$ has a CFG without any $\epsilon$-productions.
- A nonterminal $A$ in a grammar is *nullable* if $A \overset{*}{\Rightarrow} \epsilon$.
- The nullable nonterminals can be determined iteratively.
- We can eliminate all $\epsilon$-productions in a grammar as follows:
  - Eliminate all productions with $\epsilon$ bodies.
  - Suppose $A \to X_1X_2 \ldots X_k$ is a production and $m$ of the $kX_i$'s are nullable. Then add the $2^m$ versions of this production where the nullable $X_i$'s are present or absent. (But if all symbols are nullable, do not add an $\epsilon$-production.)

- Let us eliminate the $\epsilon$-productions from the grammar $G$

$$\begin{aligned} S &\to AB \\ A &\to aAA \mid \epsilon \\ B &\to bBB \mid \epsilon \end{aligned}$$

$S$, $A$, and $B$ are nullable.
For the production $S \to AB$ we add the productions $S \to A \mid B$
For the production $A \to aAA$ we add the productions $A \to aA \mid a$
For the production $B \to bBB$ we add the productions $B \to bB \mid b$
The resulting grammar $H$ with no $\epsilon$-production is

$$\begin{aligned} S &\to AB \mid A \mid B \\ A &\to aAA \mid aA \mid a \\ B &\to bBB \mid bB \mid b \end{aligned}$$

We can prove that $L(H) = L(G) - \{\epsilon\}$.

## 2 Eliminating Unit Productions from a CFG

- A *unit* production is one of the form $A \to B$ where both $A$ and $B$ are nonterminals.
- Let us assume we are given a grammar $G$ with no $\epsilon$-productions.
- From $G$ we can create an equivalent grammar $H$ with no unit productions as follows.
  - Define $(A, B)$ to be a unit pair if $A \overset{*}{\Rightarrow} B$ in $G$.
  - We can inductively construct all unit pairs for $G$.
  - For each unit pair $(A, B)$ in $G$, we add to $H$ the productions $A \to \alpha$ where $B \to \alpha$ is a nonunit production of $G$.
- Consider the standard grammar $G$ for arithmetic expressions:

$$\begin{aligned} E &\to E + T \mid T \\ T &\to T * F \mid F \\ F &\to (E) \mid a \end{aligned}$$

The unit pairs are $(E, E)$, $(E, T)$, $(E, F)$, $(T, T)$, $(T, F)$, $(F, F)$.
The equivalent grammar $H$ with no unit productions is:

$$\begin{aligned} E &\to E + T \mid T * F \mid (E) \mid a \\ T &\to T * F \mid (E) \mid a \\ F &\to (E) \mid a \end{aligned}$$

# 3 Putting a CFG into Chomsky Normal Form

- A grammar $G$ is in Chomsky Normal Form if each production in $G$ is one of two forms:

  1. $A \to BC$ where $A$, $B$, and $C$ are nonterminals, or
  2. $A \to a$ where $a$ is a terminal.

- We will further assume $G$ has no useless symbols.

- Every context-free language without $\epsilon$ can be generated by a Chomsky Normal Form grammar.

- Let us assume we have a CFG $G$ with no useless symbols, $\epsilon$- productions, or unit productions. We can transform $G$ into an equivalent Chomsky Normal Form grammar as follows:

  – Arrange that all bodies of length two or more consist only of nonterminals.

  – Replace bodies of length three or more with a cascade of fproductions, each with a body of two nonterminals.

- Applying these two transformations to the grammar $H$ above, we get:

$$
\begin{aligned}
E &\to EA \,|\, TB \,|\, LC \,|a \\
A &\to PT \\
P &\to + \\
B &\to MF \\
M &\to * \\
L &\to ( \\
C &\to ER \\
R &\to ) \\
T &\to TB \,|\, LC \,|\, a \\
F &\to LC \,|\, a
\end{aligned}
$$

# 4 Pumping Lemma for CFL's

- For every nonfinite context-free language $L$, there exists a constant $n$ that depends on $L$ such that for all $z$ in $L$ with $|z| \geq n$, we can write $z$ as $uvwxy$ where

  1. $vx \neq \epsilon$,
  2. $|vwx| \leq n$, and

3. for all $i \geq 0$, the string $uv^iwx^iy$ is in $L$.

- Proof: See HMU, pp 281 – 282.
- One important use of the pumping lemma is to prove certain languages are not context free.
- Example: The language $L = \{a^nb^nc^n \mid n \geq 0\}$ is not context free.
  - The proof will be by contradiction. Assume $L$ is context free. Then by the pumping lemma there is a constant $n$ associated with $L$ such that for all $z$ in $L$ with $|z| \geq n$, $z$ can be written as $uvwxy$ such that
    1. $vx \neq \epsilon$,
    2. $|vwx| \leq n$, and
    3. for all $i \geq 0$, the string $uv^iwx^iy$ is in $L$.
  - Consider the string $z = a^nb^nc^n$.
  - From condition (2), $vwx$ cannot contain both $a$'s and $c$'s.
  - Two cases arise:
    1. $vwx$ has no $c$'s. But then $uwy$ cannot be in $L$ since at least one of $v$ or $x$ is nonempty.
    2. $vwx$ has no $a$'s. Again, $uwy$ cannot be in $L$.

    In both cases we have a contradiction, so we must conclude $L$ cannot be context free. The details of the proof can be found in HMU, p. 284.

# 5 Cocke-Younger-Kasami Algorithm for Testing Membership in a CFL

- Input: a Chomsky normal form CFG $G = (V, T, P, S)$ and a string $w = a_1a_2 \ldots a_n$ in $T^*$.
- Output: "yes" if $w$ is in $L(G)$, "no" otherwise.
- Method: The CYK algorithm is a dynamic programming algorithm that fils in a triangular table $x_{ij}$ with nonterminals $A$ such that $A \overset{*}{\Rightarrow} a_ia_i + j \ldots a_j$.
- The algorithm adds nonterminal $A$ to $x_{ij}$ iff there is a production $A \rightarrow BC$ in $P$ where $B \overset{*}{\Rightarrow} a_ia_{i+1} \ldots a_k$ and $C \overset{*}{\Rightarrow} a_{k+1}a_{k+2} \ldots a_j$.
- To compute entry $x_{ij}$, we examine at most $n$ pairs of entries: $(x_{ii}, x_{i+1,j})$,$(x_{i,i+1}, x_{i+2,j})$, and so on until $(x_{i,j-1}, x_{j,j})$.
- The running time of the CYK algorithm is $O(n^3)$.

```
for i = 1 to n do
  if A → a_i is in P then
    add A to X_ii
fill in the table, row-by-row, from row 2 to row n
  fill in the cells in each row from left-to-right
    if (A → BC is in P) and for some i ≤ k < j
      (B is in X_ik) and (C is in X_k+1,j) then
        add A to X_ij
if S is in X_1n then
  output "yes"
else
  output "no"
```

Figure 1: The CYK Algorithm in Pseudocode