Context Free Languages: Properties

Normal Forms.

Chomsky Normal Form.

All productions are of form $A \longrightarrow BC$ or $A \longrightarrow a$ (where $a \in T$ and $A, B, C \in V$).

Useless symbols: Symbols which do not appear in any derivation of a string from the start symbol. That is, $S \Rightarrow_G^* w$, for some $w \in T^*$.

We want to eliminate useless symbols.

A symbol is said to be useful if it appears as $S \Rightarrow_G^* \alpha A \beta \Rightarrow_G^* w$, for some $w \in T^*$.

1. We say that a symbol A is generating if $A \Rightarrow_G^* w$, for some $w \in T^*$.

2. We say that a symbol A is reachable if $S \Rightarrow_G^* \alpha A \beta$, for some $\alpha, \beta \in (V \cup T)^*$.

Surely a symbol is useful only if is reachable and generating (though vice-versa need not be the case).
What we will show is that if we get rid of non-generating symbols first and then the non-reachable symbols in the remaining grammar, then we will only be left with useful symbols.

Theorem: Suppose $G = (V, T, P, S)$ is a grammar which generates at least one string.
Then, if
1) First eliminate all symbols (and productions involving these symbols) which are non-generating. Let this grammar be $G_2 = (V_2, T, P_2, S)$.
2) Remove all non-reachable symbols (and corresponding productions for them) from the grammar G_2. Suppose the resulting grammar is G_3.
Then G_3 contains no useless symbols and generates the same language as G.

Generating Symbols:
Base Case: All symbols in T are generating.
Induction: If there is a production of form $A \longrightarrow \alpha$, where α consists only of generating symbols, then A is generating.
Iterate the above process until no more symbols can be added.

Reachable symbols:
Base Case: S is reachable.
Induction Case: If A is reachable, and $A \rightarrow \alpha$ is a production, then every symbol in α is reachable.

A symbol is non-reachable, iff it is not reachable.

Converting a Grammar into Chomsky Normal Form:

1. Eliminate ϵ productions.
2. Eliminate unit-productions.
3. Convert the productions to productions of length 2 (involving non-terminals on RHS) or productions of length 1 (involving terminal on RHS).

Eliminating ϵ productions.

1. We first find all nonterminals A such that $A \Rightarrow_G^* \epsilon$. These nonterminals are called nullable.

2. Then, we get rid of ϵ productions, and for each production $B \to \alpha$, we replace it with all possible productions, $B \to \alpha'$, where α' can be formed from α by possibly deleting some of the nonterminals which are nullable.

Note: If S is nullable, then our method only generates the language $L - \{\epsilon\}$.

Theorem: If we modify the grammar as above, then $L(G') = L(G) - \{\epsilon\}$.
Proof: We prove a more general statement:
For $w \in T^* - \{\epsilon\}$, $A \Rightarrow_G^* w$, iff $A \Rightarrow_{G'}^* w$.

Suppose $A \Rightarrow_{G'}^* w$. Then we claim that $A \Rightarrow_G^* w$.

Suppose $A \Rightarrow_G^* w$. Then we claim that $A \Rightarrow_{G'}^* w$.

Identifying nullable symbols.

Base: If $A \rightarrow \epsilon$, then A is nullable.
Induction: If $A \rightarrow \alpha$, and every symbol in α is nullable, then A is nullable.
Apply the induction step until no more nullable symbols can be found.

Eliminating Unit Productions

First determine for each pair of non-terminals A, B, if $A \Rightarrow^*_G B$.
Then we need to add $A \rightarrow \gamma$, for all non unit productions of form $B \rightarrow \gamma$.

Base: (A, A) is a unit pair.
Induction: If (A, B) is a unit pair, and $B \rightarrow C$, then (A, C) is a unit pair.
Do the induction step until no more new pairs can be added.

All productions of length ≥ 2 can be changed to (a set of) productions of length 2 (involving only non-terminals on RHS) or productions of length 1 (involving terminals on RHS) as follows:
Given Production: $A \rightarrow X_1 X_2 \ldots X_k$
is changed to the following set of productions:
$A \rightarrow Z_1 B_2$,
$B_2 \rightarrow Z_2 B_3$, ...,
$B_{k-1} \rightarrow Z_{k-1} Z_k$,
$Z_i \rightarrow X_i$, if $X_i \in T$,
$Z_i = X_i$, if X_i is a nonterminal,
where B_i (and possibly) Z_i are new non-terminals.

Size of Parse Tree

Theorem: Suppose we have a parse tree using a Chomsky Normal Form Grammar. If the length of the longest path from root to a node is n, then size of the string w generated is at most 2^{n-1}.

Pumping Lemma

Pumping Lemma for CFL: Let L be a CFL. Then there exists a constant n such that, if z is any string in L such that $|z| \geq n$, then we can write $z = uvwxy$ such that:

1. $|vwx| \leq n$
2. $vx \neq \epsilon$
3. For all $i \geq 0$, $uv^iwx^iy \in L$.

Example: $\{0^n1^n2^n \mid n \geq 1\}$ is not a CFL.

Proof of Pumping Lemma for CFL.

Choose a Chomsky Normal Form grammar $G = (V, T, P, S)$ for $L - \{\epsilon\}$.

(without loss of generality, we assume $L \neq \emptyset$ and $L \neq \{\epsilon\}$).

Let $m = |V|$. Let $n = 2^m$. Suppose z of length at least 2^m is given. Consider the parse tree for z. This parse tree must have a path (from root) of length at least $m+1$ (by Theorem proved earlier). Consider the path to the leaf with largest depth. In this path, among the last $m+1$ non-terminals, there must be two nonterminals which are same (by pigeonhole principle). Suppose this looks like:

Then, $z = uvwxy$, where $S \Rightarrow_G^* uAy \Rightarrow_G^* uvAxy \Rightarrow_G^* uvwxy$.

Thus, we have $A \Rightarrow_G^* vAx$, $A \Rightarrow_G^* w$.

Thus, $S \Rightarrow_G^* uv^iwx^iy$, for all i.

Note that length of vwx is at most 2^m.

Closure Properties:

Substitution: Suppose we map each terminal a to a CFL L_a (i.e., $s(a) = L_a$).

Theorem: Suppose L is CFL over Σ and s is a substitution on Σ such that $s(a) = L_a$ is CFL, for each $a \in \Sigma$. Then, $\cup_{w \in L} s(w)$ is a CFL.

Reversal

$L^R = \{w^R \mid w \in L\}$

If L is CFL, then L^R is CFL.

If L is CFL and R is regular, then $L \cap R$ is CFL.

Testing whether CFL is $\emptyset$ or not.

Testing membership in a CFL.
CYK algorithm. Using Chomsky Normal Form.
For $w = a_1 \ldots a_n$, we determine (using a dynamic programming algorithm) the set $X_{i,j}$ of nonterminals which generate the string $a_i a_{i+1} \ldots a_j$.
Note that $X_{i,i}$ is just the set of non-terminals which generate a_i.
$X_{i,j}$ then contains all A such that $A \rightarrow BC$ and $B \in X_{i,k}$, $C \in X_{k+1,j}$, for $i \leq k < j$.
Running Time of the algorithm is n^3.