**CFL's are in P**

# 1 Introduction

In this manuscript we prove that if $L$ is a Contex Free Language (CFL) then $L \in P$. In paticular, $L$ can be solved in time $O(n^3)$.

We need the following definitions before we can say what our steps are

**Notation 1.1**

1. Capital letters are nonterminals. Small letters are terminals (elements of $\Sigma$). $\sigma \in \Sigma$.
2. Let $\alpha, \beta \in (N \cup \Sigma)^*$. $\alpha \rightarrow_G^* \beta$ means that starting from $\alpha$ if you apply some finite number of productions you end up with $\beta$.

**Def 1.2** Let $G$ be a context free grammar.

1. $G$ is *e-free* if there are no productions of the form $A \rightarrow e$.
2. A *unit prodution* is a production of the form $A \rightarrow B$.
3. $G$ is in *Chomsky Normal Form* if every production is of the form either $A \rightarrow BC$ or $A \rightarrow \sigma$

Here is our rough plan. For this high level descrption we ignore the case where $e \in L(G)$.

1. We give a procedure that will take a CFG $G$ such that $e \notin L(G)$ and return an $e$-free CFG $G'$ such that $L(G) = L(G')$.
2. We give a procedure that will take a CFG $G$ with no $e$-productions and return a CFG $G'$ with no $e$-production AND no unit productions such that $L(G) = L(G')$.
3. We give a procedure that takes a CFG $G$ with no $e$-productions and no unit productions and return a grammar $G'$ in Chomsky Normal Form such that $L(G) = L(G')$.
4. We show that if $G$ is in Chomsky Normal Form then there exists an $O(n^3)$ time algorithm for $L(G)$.

**Note 1.3** Better algorithms are known. Let $\omega$ be the constant on matrix multiplication. Then there is an algorithm for CFL's that is in time $O(n^\omega)$.

## 2 Getting Rid of e-Productions

**Theorem 2.1** *There is an algorithm that does the following: Given a grammar $G$ (1) Determine if $e \in L(G)$, and (2) (in any case) return an e-free grammar $G'$ such that $L(G') = L(G) - \{e\}$.*

**Proof:** Do the following procedure until either there are no $e$-productions or there is one and it is $S \rightarrow e$.

1. If there exists a production of the form $A \rightarrow e$ $(A \neq S)$ then do the following:
   (a) Remove the production $A \rightarrow e$.
   (b) For every production of the form
   $B \rightarrow \alpha A \beta$
   add the production
   $B \rightarrow \alpha\beta$.
   (Note- you still KEEP the production $B \rightarrow \alpha A \beta$.)

If at the end there are NO $e$-productions then $e \notin L(G)$ and the resulting grammar is $G'$. If at the end there are is the $e$-productions $S \rightarrow e$ then $e \in L(G)$ and let $G'$ be the resulting grammar MINUES $S \rightarrow e$.
∎

## 3 Getting Rid of Unit Productions

**Def 3.1** Let $G = (N, \Sigma, P, S)$ be a CFG. A production of the form $A \rightarrow B$ is a *unit production.*

**Theorem 3.2** *There exists an algorithm that will, given a CFG $G = (N, \Sigma, P, S)$ with no e-productions, will output a grammar $G' = (N', \Sigma, P', S')$ with no e-production AND no unit productions such that $L(G) = L(G')$.*

**Proof:** We give the algorithm, show that it works in the correct time, but do not prove that it works.

We use $\Rightarrow$ to mean $\rightarrow_G^*$. We first find all $A, B \in N$ such that $A \Rightarrow B$. Since there are no e-productions this is easy and only involves unit-productions. Formally we make a directed graph out of all of the nonterminals, with an edge between $X$ and $Y$ if $X \rightarrow Y$. Then, all pairs $A, B$

such that there is a directed graph from $A$ to $B$ are all the pairs such that $A \Rightarrow B$.

Let the set of all $(A, B)$ such that $A \Rightarrow B$ be called SUPERUNITS.

1. Let $PROD$ be $P$ minus the UNIT productions.
2. Find the set of SUPERUNITES.
3. For all SUPERUNITS $A \Rightarrow B$ do
   (a) For all productions in $PROD$ of the form $X \rightarrow \alpha_1 A \alpha_2 A \cdots \alpha_{L-1} A \alpha_L$ add all productions that replace some of the $A$'s with $B$'s (there will be $2^L - 1$ new productions). Note that these new productions will NOT be unit productions.

∎

# 4 Chomsky Normal Form

**Def 4.1** A grammar Let $G = (N, \Sigma, P, S)$ is in *Chomsky Normal Form* if every production is either of the form $A \rightarrow BC$ or $A \rightarrow \sigma$ where $\sigma \in \Sigma$.

**Theorem 4.2** *There exists an algorithm that will, given a CFG $G = (N, \Sigma, P, S)$ with no e-productions, no unit productions, output a grammar $G' = (N', \Sigma, P', S')$ in Chomsky Normal Form such that such that $L(G) = L(G')$.*

**Proof:**
Look at each rule of the form $A \rightarrow \alpha_1 \alpha_2 \cdots \alpha_L$.

1. If $L = 2$ and $\alpha_1, \alpha_2$ are nonterminals then leave this production alone.
2. If $L = 2$ and at least one of $\alpha_1, \alpha_2$ is a terminal OR if $L \geq 3$ then we do the following:
   (a) Replace every terminal $\alpha_i$ with nonterminal $[\alpha_i]$ and add the rule $[\alpha_i] \rightarrow \alpha_i$.
   (b) Note that the rule is now of the form
   $A \rightarrow \beta_1 \cdots \beta_L$
   where each $\beta_i$ is a nonterminal.
   Replace this with the following:
   $A \rightarrow [\beta_1 \cdots \beta_{L-1}]\beta_L$
   $[\beta_1 \cdots \beta_{L-1}] \rightarrow [\beta_1 \cdots \beta_{L-2}]\beta_{L-1}$
   $[\beta_1 \cdots \beta_{L-2}] \rightarrow [\beta_1 \cdots \beta_{L-3}]\beta_{L-2}$
   etc until
   $[\beta_1\beta_2\beta_3]] \rightarrow [\beta_1\beta_2]\beta_3$
   $[\beta_1\beta_2] \rightarrow \beta_1\beta_2$.

∎

# 5 CFL's in P

**Theorem 5.1** *If $L$ is a CFL then $L$ is in $O(n^3)$.*

**Proof:** If $L = \emptyset$ then $L$ is in $O(n^3)$ time. Apply the procedure in Theorems 2.1 and 3.2 to determine if $e \in L(G)$ and also to obtain a $G'$ such that $L(G') = L(G) - \{e\}$.

We show that $L(G')$ is in $O(n^3)$. This time does not count for the algorithm. This time is preprocessing.

We use DYNAMIC PROGRAMMING! Intuitively: Given a string $w = w_1 w_2 \ldots w_n$ we want to look which nonterminals $A$ can produce $w_i \cdots w_j$. We do this, first for $i = j$ (that is $j - i = 0$) then for $j - i = 1$, $j - i = 2$, etc. The KEY is that $D$ generates $w_i w_{i+1} \ldots w_j$ iff $D \rightarrow BC$ and $B$ generates a prefix, say $w_i \cdots w_k$, and $C$ generates the remaining suffice, say $w_{k+1} \cdots w_n$.

The KEY definition is

$$A[i,j] = \{B \mid B \Rightarrow w_i \cdots w_j\}.$$

The formal algorithm is on then page.

There are $O(n^2)$ spaces in the array to fill out. Each one takes at most $O(n)$ to fill out.

CFL's in P

```
for i=1 to n
    A[i,i] = {B | B → w_i}
for d=1 to n−1
    for i=1 to n−d
        j=i+d
        A[i,j] = ⋃_{i≤k<j} {D | B ∈ A[i,k] ∧ C ∈ A[k+1,j] ∧ D → BC}
If S ∈ A[1,n] then output YES, else output NO.
```

∎