Proof of the Bar-Hillel Pumping Lemma for Context Free Grammars

Prof. Michael Mascagni

Department of Computer Science
Department of Mathematics
Department of Scientific Computing
Graduate Program in Molecular Biophysics
Florida State University, Tallahassee, FL 32306 **USA**
AND
Applied and Computational Mathematics Division, Information Technology Laboratory
National Institute of Standards and Technology, Gaithersburg, MD 20899-8910 **USA**
E-mail: mascagni@fsu.edu or mascagni@nist.gov
URL: http://www.cs.fsu.edu/∼mascagni

Outline of the Proof

Outline of the Proof

- Background Material
  - The Pumping Lemma for Regular Languages
  - Background from Context-Free Languages
    - Context-Free Grammars
    - Derivations of Words in a CFG
    - Parse Trees
    - Splicing and Pruning Parse Trees
    - Chomsky Normal Form
- The Pumping Lemma for Context-Free Languages
  - Statement of the Bar-Hillel Pumping Lemma
  - An Important Lemma Needed to Prove the Pumping Lemma
  - Proof of the Main Result

The Pumping Lemma for Regular Languages

Statement of the Pumping Lemma for Regular Languages: Let L be a regular language and $x \in L$ be a word in L. Furthermore, assume that $|x| > n$, where n is the number of states in the Finite Automata (FA) that accepts L. The existence of such a FA is guaranteed by Kleene's Theorem. Then x can be written $x = uvw$, where:

1. $v \neq \Lambda$, and
2. $uv^i w \in L, \forall\ i = 0, 1, 2, ...$

The Pumping Lemma for Regular Languages

Statement of the Pumping Lemma for Regular Languages: Let L be a regular language and $x \in L$ be a word in L. Furthermore, assume that $|x| > n$, where n is the number of states in the Finite Automata (FA) that accepts L. The existence of such a FA is guaranteed by Kleene's Theorem. Then x can be written $x = uvw$, where:

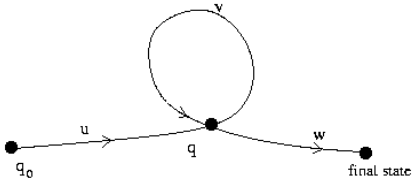
1. $v \neq \Lambda$, and
2. $uv^i w \in L, \forall\ i = 0, 1, 2, ...$

The Pumping Lemma for Regular Languages

Proof: Assume L and x are as defined above. In the processing of x, the number of transitions from state to state that the FA makes will be equal to or greater than n, so the sequence of states exhibited, counting the initial state, will have more than n terms. Since the number of terms in the sequence exceeds n, the number of states in the FA, we can invoke the pigeon hole principle to conclude that at least two distinct terms of the sequence will have the same value. That's the same as saying that some state will be revisited. Let q be the first such state that is revisited.

The Pumping Lemma for Regular Languages

Let u be the input string that causes the machine to first visit q.
Let v be the input string that causes the machine to visit q again.; clearly, $v \neq \Lambda$.
Let w be the input string that cause the FA to go from this second visit of q to the final state that signifies that the string is accepted.
The relative roles of u, v, and w are suggested by the figure below:



Now it should be clear that omitting v or repeating it multiple times, would give rise to a string that still bring us to the same final state. In other words, $uv^i w \in L \; \forall i = 0, 1, 2, \ldots$. $\square$

Context-Free Grammars

A Context-Free Grammar (CFG) is determined when we specify three items:

1. An alphabet, Σ, whose members we call terminals, from which we make the strings of the language to be generated. We frequently use lower case letters for terminals.
2. A finite set, disjoint from the terminals, called non-terminals, one of which is the start symbol. We frequently use S for the start symbol and other upper case letters for the remaining non-terminals.
3. A finite set of symbolic rules, called production rules, each of the form non-terminal $\rightarrow$ string

The string on the right of the arrow is permitted to be empty, a string of terminals, a string of non-terminals, or mixed.

Context-Free Grammars

A Context-Free Grammar (CFG) is determined when we specify three items:

1. An alphabet, Σ, whose members we call terminals, from which we make the strings of the language to be generated. We frequently use lower case letters for terminals.
2. A finite set, disjoint from the terminals, called non-terminals, one of which is the start symbol. We frequently use S for the start symbol and other upper case letters for the remaining non-terminals.
3. A finite set of symbolic rules, called production rules, each of the form non-terminal $\rightarrow$ string

The string on the right of the arrow is permitted to be empty, a string of terminals, a string of non-terminals, or mixed.

Context-Free Grammars

A Context-Free Grammar (CFG) is determined when we specify three items:

1. An alphabet, Σ, whose members we call terminals, from which we make the strings of the language to be generated. We frequently use lower case letters for terminals.
2. A finite set, disjoint from the terminals, called non-terminals, one of which is the start symbol. We frequently use S for the start symbol and other upper case letters for the remaining non-terminals.
3. A finite set of symbolic rules, called production rules, each of the form non-terminal $\rightarrow$ string

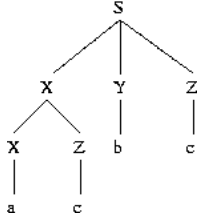
The string on the right of the arrow is permitted to be empty, a string of terminals, a string of non-terminals, or mixed.

Derivations of Words in a CFG

Consider the following CFG:

1. $S \to XYZ$
2. $X \to XZ$
3. $X \to a$
4. $Y \to b$
5. $Z \to c$

We can derive the word *acbc* with the following Derivation:
$S \Rightarrow XYZ \Rightarrow XZYZ \Rightarrow aZYZ \Rightarrow acYZ \Rightarrow acbZ \Rightarrow acbc$
A graphic representation of this derivation is given by a Parse Tree

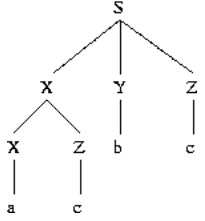


Derivations of Words in a CFG

Consider the following CFG:

1. $S \to XYZ$
2. $X \to XZ$
3. $X \to a$
4. $Y \to b$
5. $Z \to c$

We can derive the word *acbc* with the following Derivation:
$S \Rightarrow XYZ \Rightarrow XZYZ \Rightarrow aZYZ \Rightarrow acYZ \Rightarrow acbZ \Rightarrow acbc$
A graphic representation of this derivation is given by a Parse Tree

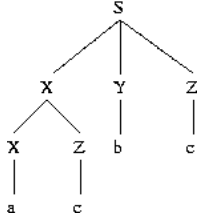


Derivations of Words in a CFG

Consider the following CFG:

1. $S \to XYZ$
2. $X \to XZ$
3. $X \to a$
4. $Y \to b$
5. $Z \to c$

We can derive the word *acbc* with the following Derivation:
$S \Rightarrow XYZ \Rightarrow XZYZ \Rightarrow aZYZ \Rightarrow acYZ \Rightarrow acbZ \Rightarrow acbc$
A graphic representation of this derivation is given by a Parse Tree

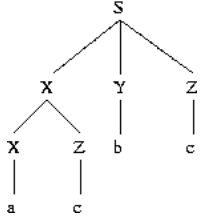


Derivations of Words in a CFG

Consider the following CFG:

1. $S \to XYZ$
2. $X \to XZ$
3. $X \to a$
4. $Y \to b$
5. $Z \to c$

We can derive the word *acbc* with the following Derivation:
$S \Rightarrow XYZ \Rightarrow XZYZ \Rightarrow aZYZ \Rightarrow acYZ \Rightarrow acbZ \Rightarrow acbc$
A graphic representation of this derivation is given by a Parse Tree

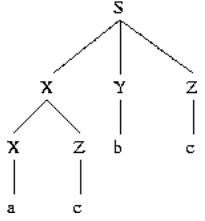


Derivations of Words in a CFG

Consider the following CFG:

1. $S \to XYZ$
2. $X \to XZ$
3. $X \to a$
4. $Y \to b$
5. $Z \to c$

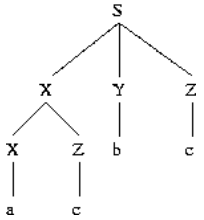
We can derive the word *acbc* with the following Derivation:
$S \Rightarrow XYZ \Rightarrow XZYZ \Rightarrow aZYZ \Rightarrow acYZ \Rightarrow acbZ \Rightarrow acbc$
A graphic representation of this derivation is given by a Parse Tree



Parse Trees

Parse trees have certain properties:

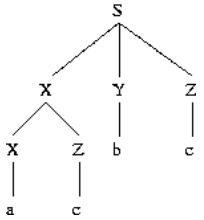
1. S is the root
2. All the leaves are terminals
3. Each non-terminal has at least one branch



Parse Trees

Parse trees have certain properties:

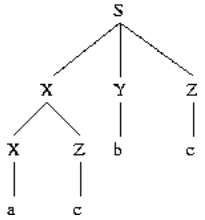
1. S is the root
2. All the leaves are terminals
3. Each non-terminal has at least one branch



Parse Trees

Parse trees have certain properties:

1. S is the root
2. All the leaves are terminals
3. Each non-terminal has at least one branch



Splicing and Pruning Parse Trees

A particularly interesting situation arises when two different vertices, α and β, lie on the same path in the parse tree are labeled by the same non-terminal, X
In such a case one of the vertices is a descendant of the other, say, β is a descendant of α
We wish to consider two important operations on this parse tree which can be performed:

1. The first operation, which we call pruning, is to remove the subtree rooted at α and to graft the subtree rooted at β in its place
2. The second operation, which we call splicing, is to remove the subtree rooted at β and to graft an exact copy of the subtree rooted at α in its place

Because α and β are labeled by the same variable, the trees obtained by pruning and splicing are themselves parse trees of different words in the language, see the figures

Splicing and Pruning Parse Trees

A particularly interesting situation arises when two different vertices, α and β, lie on the same path in the parse tree are labeled by the same non-terminal, X
In such a case one of the vertices is a descendant of the other, say, β is a descendant of α
We wish to consider two important operations on this parse tree which can be performed:

1. The first operation, which we call pruning, is to remove the subtree rooted at α and to graft the subtree rooted at β in its place
2. The second operation, which we call splicing, is to remove the subtree rooted at β and to graft an exact copy of the subtree rooted at α in its place

Because α and β are labeled by the same variable, the trees obtained by pruning and splicing are themselves parse trees of different words in the language, see the figures

Splicing and Pruning Parse Trees

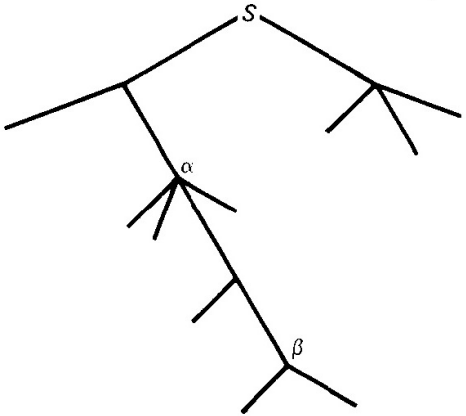


Figure: Parse tree with $\alpha = \beta = X$ a non-terminal

Splicing and Pruning Parse Trees

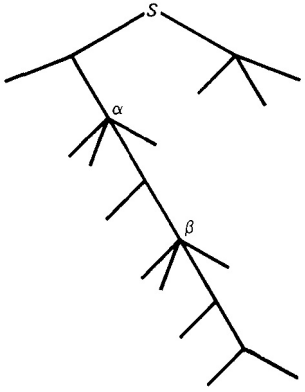


Figure: β-rooted $\leftarrow$ α-rooted (Splicing)

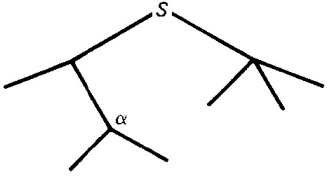


Figure: α-rooted $\leftarrow$ β-rooted (Pruning)

Chomsky Normal Form

A Chomsky Normal Form Grammar is a CFG with all production rules of the form:

1. non-terminal $\rightarrow$ terminal
2. non-terminal $\rightarrow$ (non-terminal)(non-terminal)

Chomsky Normal Form Theorem: L is a context-free language $\iff$ $L - \{\Lambda\}$ can be generated by a CNF grammar

Important points about CNF grammars germane to the pumping lemma

1. CNF parse trees are binary
2. All CNF parse trees have non-terminals labeling the branches
3. Only the leaves of CNF trees are labeled with terminals

Chomsky Normal Form

A Chomsky Normal Form Grammar is a CFG with all production rules of the form:

1. non-terminal $\rightarrow$ terminal
2. non-terminal $\rightarrow$ (non-terminal)(non-terminal)

Chomsky Normal Form Theorem: L is a context-free language $\iff$ $L - \{\Lambda\}$ can be generated by a CNF grammar

Important points about CNF grammars germane to the pumping lemma

1. CNF parse trees are binary
2. All CNF parse trees have non-terminals labeling the branches
3. Only the leaves of CNF trees are labeled with terminals

Chomsky Normal Form

A Chomsky Normal Form Grammar is a CFG with all production rules of the form:

1. non-terminal $\rightarrow$ terminal
2. non-terminal $\rightarrow$ (non-terminal)(non-terminal)

Chomsky Normal Form Theorem: L is a context-free language $\iff$ $L - \{\Lambda\}$ can be generated by a CNF grammar

Important points about CNF grammars germane to the pumping lemma

1. CNF parse trees are binary
2. All CNF parse trees have non-terminals labeling the branches
3. Only the leaves of CNF trees are labeled with terminals

Chomsky Normal Form

A Chomsky Normal Form Grammar is a CFG with all production rules of the form:

1. non-terminal $\rightarrow$ terminal
2. non-terminal $\rightarrow$ (non-terminal)(non-terminal)

Chomsky Normal Form Theorem: L is a context-free language $\iff$ $L - \{\Lambda\}$ can be generated by a CNF grammar

Important points about CNF grammars germane to the pumping lemma

1. CNF parse trees are binary
2. All CNF parse trees have non-terminals labeling the branches
3. Only the leaves of CNF trees are labeled with terminals

Chomsky Normal Form

A Chomsky Normal Form Grammar is a CFG with all production rules of the form:

1. non-terminal $\rightarrow$ terminal
2. non-terminal $\rightarrow$ (non-terminal)(non-terminal)

Chomsky Normal Form Theorem: L is a context-free language $\iff$ $L - \{\Lambda\}$ can be generated by a CNF grammar

Important points about CNF grammars germane to the pumping lemma

1. CNF parse trees are binary
2. All CNF parse trees have non-terminals labeling the branches
3. Only the leaves of CNF trees are labeled with terminals

The Pumping Lemma for Context-Free Languages

The Pumping Lemma for Context-Free Languages (Bar-Hillel):
Let Γ be a CNF grammar with exactly n non-terminals, and let L be the languages generated by Γ. Then $\forall x \in L$ with $|x| > 2^n$, we have $x = r_1 q_1 r q_2 r_2$, where

1. $|q_1 r q_2| \leq 2^n$
2. $q_1 q_2 \neq 0$
3. $r_1 {q_1}^i r {q_2}^i r_2 \in L, \forall i \geq 0$

The Pumping Lemma for Context-Free Languages

The Pumping Lemma for Context-Free Languages (Bar-Hillel):
Let Γ be a CNF grammar with exactly n non-terminals, and let L be the languages generated by Γ. Then $\forall x \in L$ with $|x| > 2^n$, we have $x = r_1 q_1 r q_2 r_2$, where

1. $|q_1 r q_2| \leq 2^n$
2. $q_1 q_2 \neq 0$
3. $r_1 {q_1}^i r {q_2}^i r_2 \in L, \forall i \geq 0$

The Pumping Lemma for Context-Free Languages

The Pumping Lemma for Context-Free Languages (Bar-Hillel):
Let Γ be a CNF grammar with exactly n non-terminals, and let L be the languages generated by Γ. Then $\forall x \in L$ with $|x| > 2^n$, we have $x = r_1 q_1 r q_2 r_2$, where

1. $|q_1 r q_2| \leq 2^n$
2. $q_1 q_2 \neq 0$
3. $r_1 {q_1}^i r {q_2}^i r_2 \in L, \forall i \geq 0$

The Pumping Lemma for Context-Free Languages

First we must prove the following lemma:

Lemma: Let $u \in L(\Gamma)$, where Γ is a CNF grammar. If the parse tree for $u \in L(\Gamma)$ has no path with more than k nodes, then $|u| \leq 2^{k-1}$

Proof: We will prove this theorem by induction.

Base Case: Assume that u is just a single terminal, let us call it a, i.e. $u = a$. Then the parse tree of u consists of $k = 2$ nodes corresponding to the production rule $S \to a$, and there is no path in the parse tree with more two nodes, and: $|u| = |a| = 1 \leq 2^{2-2} = 2^0 = 1$.

The Pumping Lemma for Context-Free Languages

Induction Hypothesis: If $w \in L(\Gamma)$ has no path in the parse tree with more than $k-1$ nodes, then $|w| \leq 2^{k-3}$. Given this we wish to prove that if $u \in L(\Gamma)$ has no path in the parse tree with more than k nodes, then $|u| \leq 2^{k-2}$.

Let $u \in L(\Gamma)$ has no path in the parse tree with more than k nodes. Since Γ is a CNF grammar, and u is longer than a single character, the first production rule used to derive u is of the form non-terminal $\rightarrow$ (non-terminal)(non-terminal). In particular, S is the non-terminal on the left, and we can call the two non-terminals on the right X and Y, so that the root of the parse tree for u starts with the production rule $S \rightarrow XY$. The subtrees rooted at X and Y are shorter that the whole tree, and so have have no path longer than $k-1$. By the induction hypothesis, the derivations rooted at X and Y are for words no longer than 2^{k-3}, so u must be shorter than the longest words derivable in each subtree:

$$|u| \leq 2^{k-3} + 2^{k-3} = 2^{k-2} \square$$

The Pumping Lemma for Context-Free Languages

The following figure illustrates the points used in the proof of the lemma

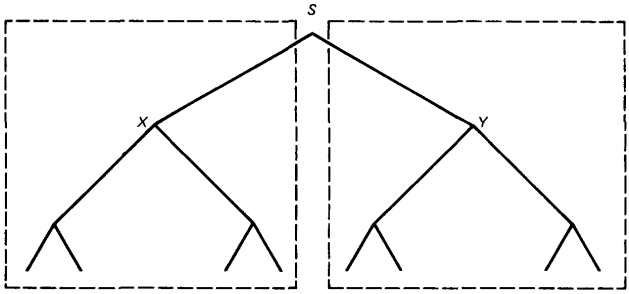


Figure: Parse tree for $u \in \Gamma$ starting from $S \rightarrow XY$

The Pumping Lemma for Context-Free Languages

Proof of the Pumping Lemma: Let $x \in L$, where $|x| > 2^n$, and consider the parse tree for $x \in \Gamma$. Let us find the longest path in the derivation of x in the parse tree, it's node are labeled by $\alpha_1, \alpha_2, \ldots, \alpha_m$. Then $m \geq n+2$. This is due to the lemma, since if $m < n+1$, by $|x| \leq 2^{n-1}$.

We note that α_m must leaf, and is labeled by a terminal, and so $\alpha_1, \alpha_2, \ldots, \alpha_{m-1}$ are all labeled by non-terminals. These $m-1 \geq n+1$ non-terminals have at least one repeated non-terminal, which we will call X. This is due to the pigeon-hole principle and observing that $m-1 \geq n+2$, and there are only n non-terminals in Γ. We also note that α_m is a leaf, and hence a terminal.

The Pumping Lemma for Context-Free Languages

Because the non-terminal, X appears at least twice in the parse tree of x, we know that the parse tree must look like the figure below:

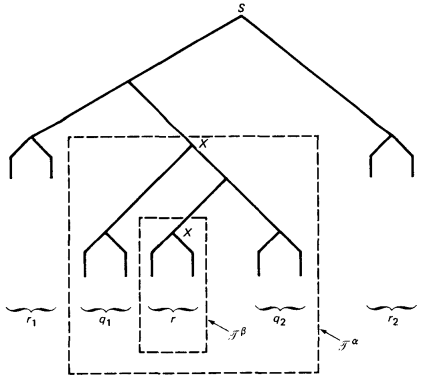


Figure: Parse tree of $x \in \Gamma$ with non-terminal X revisited

The Pumping Lemma for Context-Free Languages

Let us focus on the subtree rooted at the first occurrence of X, which we will call $\mathcal{T}^{\alpha}$ and notice that X reoccurs below. Let us denote the subtree rooted at this next occurrence of X as $\mathcal{T}^{\beta}$. Let us denote the word that is derived by $\mathcal{T}^{\beta}$ as $r = \langle \mathcal{T}^{\beta} \rangle$. Similarly, we denote $\langle \mathcal{T}^{\alpha} \rangle = q_1 r q_2$, and similarly that $x = r_1 q_1 r q_2 r_2$, as per the figure.

Since $\mathcal{T}^{\alpha}$ and $\mathcal{T}^{\beta}$ both rooted at X we may use pruning and splicing. If we remove $\mathcal{T}^{\beta}$, we know that the word $r_1 q_1 q_2 r_2 \in L$. Similarly we can replace $\mathcal{T}^{\beta}$ with as as many copies of $\mathcal{T}^{\alpha}$ as we wish (including zero copies, i.e. the Kleene star), which shows that:

$$r_1 {q_1}^i r {q_2}^i r_2 \in L, \forall i \geq 0.$$

Finally, we note that any path in $\mathcal{T}^{\alpha}$ consists of no more than $n + 2$ nodes. and so by the lemma

$$|\langle \mathcal{T}^{\alpha} \rangle| = |q_1 \langle \mathcal{T}^{\beta} \rangle q_2| = |q_1 r q_2| \leq 2^n.$$

□