Context Free Languages

**Remark:** Much of this discussion comes from:
Elements of the Theory of Computation by Lewis and Papadimitriou.

**Remark:** I may occasional use $e$ to denote the empty string $\varepsilon$.

**Motivation:**

Finite automata recognize languages. I.e. if you input a string the automaton will tell you if this string is acceptable. Now it is natural to want a more constructive approach to languages. Viz. can we construct a language which produces a string given a set of rules and some initial data? In fact regular expressions can be informally - for now - thought of as language generators:

**E.g.** Consider $a (a^* \cup b^*) b$. Clearly this is a regular expression and we can interpret this expression as follows:

Output $a$. Then do one of the following: (1) output some finite number of $a$'s or (2) output some finite number of $b$'s. Finally output $b$ and halt.

**Goal:** Construct a class of formal language generators. These generators will be Context Free Grammars.

**Informal Construction of a Context Free Grammar:**

Consider the aforementioned regular expression: $a (a^* \cup b^*) b$. An arbitrary string is of the form:

$$a \underbrace{- - - - - }_{\text{some middle piece}} b$$

So we might say: a string $S$ can be $aMb$ with $M$ some yet to be determined middle piece. We abbreviate as $S \to aMb$. Now we consider possible candidates for $M$: clearly $M$ must either be a string of $a$'s or a string of $b$'s. Denote these hypothetical strings $A$ and $B$ respectively. Using our above notation, we write: $M \to A$, $M \to B$. Now we think of how we might define $A$ and $B$ as to allow the construction of strings of $a$'s and $b$'s in some sort of iterative
fashion. One such way is as follows: $A \rightarrow aA$ and $B \rightarrow bB$. (Perhaps make a brief comment about these.) Since we want the process to halt, we also include $A \rightarrow e$ and $B \rightarrow e$. Applying all of these rules in sequence, we can build words. E.g.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$S \rightarrow aMb$</td>
</tr>
<tr>
<td>2</td>
<td>$M \rightarrow A$</td>
</tr>
<tr>
<td>3</td>
<td>$A \rightarrow aA$</td>
</tr>
<tr>
<td>4</td>
<td>$A \rightarrow aA$</td>
</tr>
<tr>
<td>5</td>
<td>$A \rightarrow e$</td>
</tr>
<tr>
<td></td>
<td>Output:</td>
</tr>
</tbody>
</table>

So after applying a sequence of the words, we have output a word of the language. This is an informal description of a Context Free Grammar.

**Remark:** Why Context Free? The are called context free because the rules do not account for context. E.g. in step (3) the derived word is $aaAb$. The "context" of $A$ can be thought of as $aa$ and $b$. However the rule $A \rightarrow aA$ is independent of the "context". Hence we are "Context Free". (Some more general grammars account for context.)

**Formal Construction:**

**Definition:** A context free grammar $G$ is a $4 - tuple$ $(V, \Sigma, R, S)$ where

- $V$ is an alphabet
- $\Sigma \subseteq V$ is a set of terminals. (The output words will be in this alphabet)
- $R \subseteq (V - \Sigma) \times V^*$ is a finite set of rules
- $S \in V - \Sigma$ is the start symbol.

**Notation:**

- We call $V - \Sigma$ the non-terminals.
- $(A, u) \in R$ is denoted $A \rightarrow u$. (I drop the subscript in the following discussion.)
Definitions:

1. For \( u, v \in V^* \) we write \( u \Rightarrow v \) iff there is \( x, y, v' \in V^* \) and \( A \in V - \Sigma \) such that \( u = xAy, v = xv'y, \) and \( A \rightarrow v' \in R. \) I.e. we can produce \( v \) from \( u \) by applying a rule from \( R. \) Diagram:

\[
\begin{array}{c|c}
\text{Rule} & \text{Word} \\
\hline
xAy = u & \hline
A \rightarrow v' & xv'y = v \\
\hline
\end{array}
\]

2. We define \( \Rightarrow^* \) to be the reflexive, transitive closure of \( \Rightarrow \).

3. \( L(G) := \) the language generated by \( G = \left\{ w \in \Sigma^* : S \Rightarrow^*_G w \right\}. \) We say that \( G \) generates \( L(G). \) (Remark: The machine halts when all non-terminals have been removed from the right-hand side.)

4. A language \( L \) is \textit{context free} if \( L = L(G) \) for some context free grammar \( G. \)

5. We call a sequence of the following form

\[ w_0 \Rightarrow_G w_1 \Rightarrow_G ... \Rightarrow_G w_n \]

\( w_n \) a \textit{derivation} in \( G \) of \( w_n \) from \( w_0. \) (in \( n \)-steps)

Example 1:
Let \( G = (V, \Sigma, R, S) \) with \( V = \{ S, a, b \}, \Sigma = \{ a, b \} \) and \( R = \{ S \rightarrow aSb, S \rightarrow e \}. \) Here we have a possible derivation:

\[ S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb \]

So from \( S \) we derive \( aabb. \) The rules we are using can be illustrated as follows:

\[
\begin{array}{c|c}
\text{Rule} & \text{Word} \\
1 & S \rightarrow aSb & aSb \\
2 & S \rightarrow aSb & aaSbb \\
3 & S \rightarrow e & aaebb \\
Output & aabb
\end{array}
\]
As a corollary, we see that $L(G) = \{a^n b^n : n \geq 0\}$ which we know to be non-regular. So the class of context free languages include languages which are not regular. However, we will see that all regular languages are context free. (Draw picture.)

**Example 2:**
(This example gives a nice technique for proving facts about context free languages.)

Let $G = (V, \Sigma, R, S)$ be a grammar where:

- $V = \{a, b, S, A\}$
- $\Sigma = \{a, b\}$
- $R = \{S \rightarrow AA, A \rightarrow AAA, A \rightarrow a, A \rightarrow bA, A \rightarrow Ab\}$

**Claim:** $L(G)$ consists of all strings in $\Sigma^*$ with number of occurrences of $a$ even and greater then zero.

**proof:** First we will prove that all strings in $L(G)$ have an even, nonzero number of $a$’s. I.e. $L(G) \subseteq \{w \in \Sigma^* : |w|_a > 0 \text{ and even}\}$. We do this by proving the following stronger claim:

Whenever $w \in V^*$ and $S \xrightarrow{*} w$, then $|w|_A + |w|_a > 0$ and even.

We prove this stronger result by induction on the length of a derivation, $k$, for the following hypothesis: For all $w \in V^*$ such that $S \xrightarrow{*} w$ in $k$ or fewer steps, then $|w|_A + |w|_a > 0$ and even. (*) (NOTE: This approach by induction on the length of derivation is a typical approach.)

Base Case: $k = 1$. Inspecting the rules, we see that $AA$ is the only element of $V^*$ which can be derived from $S$. And so (*) trivially holds. Thus we have the base case.

Inductive Case: Suppose (*) holds up to $k$ and that $w \in V^*$ is such that $S \xrightarrow{*} w$ in $k + 1$ steps. Let’s write out the last steps of the derivation:

$$... \Rightarrow w' \Rightarrow w$$

So $S \xrightarrow{*} w'$ in $k$ or fewer steps. By the induction hypothesis, $|w'|_A + |w'|_a > 0$ and even. Inspecting the rules which allow $G$ to move from $w'$ to $w$, we make the following observations:
• $A \rightarrow AAA$ and $S \rightarrow AA$ add two additional $A$’s.

• $A \rightarrow a$ replaces $A$ with $a$.

• $A \rightarrow Ab$ and $A \rightarrow bA$ do not change the number of $a$’s or $A$’s.

So any rule applied to obtain $w'$ either does not change $|w'|_A + |w'|_a$ or increases it by 2. Hence $|w|_A + |w|_a > 0$ and even. So (*) holds in this case. This finishes the induction and we conclude that (*) holds for all $k$. Thus all $w$ such that $S \Rightarrow w$ are such that $|w|_A + |w|_a > 0$ and even. In particular all $w \in L(G)$ have this property. And since these do not contain any $A$’s, they have an even number of $a$’s. Thus $L(G) \subseteq \{w \in \Sigma^* : |w|_a > 0$ and even\}.

(Time permitting.) For the reverse inclusion, let $w \in \{w \in \Sigma^* : |w|_a > 0$ and even\}. So we can write $w = b^{m_1}ab^{m_2}a\ldots b^{m_{2n+1}}$ for some $n > 0$ and $m_1 \ldots m_{2n+1} \geq 0$. ($b^0 = \epsilon$.) Then $w$ can be produced by $G$ as follows:

<table>
<thead>
<tr>
<th>Derivation</th>
<th>Rule Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \Rightarrow AA$</td>
<td>$S \rightarrow AA$ (once)</td>
</tr>
<tr>
<td>$\Rightarrow A^{2n}$</td>
<td>$A \rightarrow AAA$ ($n-1$ times)</td>
</tr>
<tr>
<td>$\Rightarrow b^{m_1}A^{2n}$</td>
<td>$A \rightarrow bA$ ($m_1$ times)</td>
</tr>
<tr>
<td>$\Rightarrow b^{m_1}aA^{2n-1}$</td>
<td>$A \rightarrow a$ (once)</td>
</tr>
<tr>
<td>$\Rightarrow b^{m_1}ab^{m_2}A^{2n-1}$</td>
<td>$A \rightarrow bA$ ($m_2$ times)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$\Rightarrow b^{m_1}ab^{m_2}\ldots b^{m_{2n}}A$</td>
<td>...</td>
</tr>
<tr>
<td>$\Rightarrow b^{m_1}ab^{m_2}\ldots b^{m_{2n}}A^{m_{2n+1}}$</td>
<td>$A \rightarrow Ab$ ($m_{2n+1}$ times)</td>
</tr>
<tr>
<td>$\Rightarrow b^{m_1}ab^{m_2}\ldots b^{m_{2n}}ab^{m_{2n+1}}$</td>
<td>$A \rightarrow a$ (once)</td>
</tr>
</tbody>
</table>

So we see that $S \Rightarrow w$ and therefore $w \in L(G)$. Thus we conclude that $L(G) = \{w \in \Sigma^* : |w|_a > 0$ and even\} and we have the claim.

\[\blacksquare\]

**Goal:** Show all regular languages are context free.

**Remark:** We accomplish this by defining a special sub collection of context free grammars; viz. regular grammars.
Definition: A context free grammar \( G = (V, \Sigma, R, S) \) is regular iff \( R \subseteq (V - \Sigma) \times \Sigma^*((V - \Sigma) \cup \{e\}) \)

(NOTE: A regular grammar is such that every rule contains at most one non-terminal in the right-hand side; E.g. \( A \rightarrow wB \).)

Example: (omit?) Let \( G = (V, \Sigma, R, S) \) be the context free grammar with

- \( V = \{S, A, B, a, b\} \)
- \( \Sigma = \{a, b\} \)
- \( R = \{S \rightarrow bA, S \rightarrow aB, A \rightarrow abaS, B \rightarrow babS, S \rightarrow e\} \)

A sample derivation performed by this machine:

\[
S \Rightarrow bA \Rightarrow babS \Rightarrow babaB \Rightarrow babaababS \Rightarrow babaababe = babaababe
\]

With a bit of work one can show that \( L(G) = (\{abab\} \cup \{baba\})^* \)

Theorem: A language is regular if and only if it is generated by a regular grammar.

Proof: (NOTE: Lewis defines finite automata a bit different then we do; hence this proof is an adaptation of the one he gives.)

Fix an alphabet \( \Sigma = \{a_1...a_k\} \)

(\(\Rightarrow\)) Suppose \( L \subseteq \Sigma^* \) is a regular language accepted by a deterministic automaton \( M = (\Omega, E, s_0, A, l) \). Let \( \Omega = \{q_1...q_m\} \) and define the regular grammar \( G = (V, \Sigma, R, S) \) where:

- \( V = \Sigma \cup \Omega \)
- \( S = s_0 \)
- \( R = \{q_i \rightarrow aq_j : a \in l(E_{i,j}) \text{ and } E_{i,j} \in E\} \cup \{q \rightarrow e : q \in A\} \)

With \( E_{i,j} \) is the edge from \( q_i \) to \( q_j \). Observe that \( G \) is clearly a regular grammar since the non-terminals are exactly \( \Omega \).

Claim: \( L(M) = L(G) \)

pf: (\(\supset\)). Take \( w = a_1...a_n \in L(G) \). Hence there exists some derivation of the following form:

\[
s_0 \Rightarrow a_1q_{i_1} \Rightarrow a_2q_{i_2}... \Rightarrow a_1...a_nq_{i_n} \Rightarrow a_1...a_n
\]
By construction of $G$, $s_0q_{i_1}...q_{i_n}$ is an edge path in $\Omega, E$. Moreover, $q_{i_n} \in A$ and $w$ is a label for this edge path. Hence $M$ recognizes $w$; i.e. $w \in l(M)$.

$(\subseteq)$ Take $w = a_1..a_n \in l(M)$. So there is some edge path $s_0q_{i_1}...q_{i_n}$ in $(\Omega, E)$ such that $q_{i_n} \in A$ and $a_k \in l(E_{i_k,i_{k+1}})$. (Define $q_{i_0} = s_0$). Hence the following rules are elements of $R$: \{ $q_{i_j} \to a_{j+1}q_{i_{j+1}} : 0 \leq j \leq n-1$ \} and $q_{i_n} \to e$. Hence, we have the following derivation:

$$s_0 \Rightarrow a_1q_{i_1} \Rightarrow ...a_1..a_nq_{i_n} \Rightarrow a_1...a_ne = a_1...a_n$$

So $w \in l(G)$. $\square$

Thus we have this direction of the theorem. (This is enough to accomplish our goal.)

$(\Leftarrow)$ Let $L \subseteq \Sigma^*$ be recognized by $G = (V, \Sigma, R, S_0)$ a regular grammar. Say that $V - \Sigma = \{A_1..A_m\}$Here we define a $\varepsilon - NFA$ $M = (\Omega, E, S, A, l)$ with

- $\Omega = (V - \Sigma) \cup \{P\}$ with $P$ a new state. (i.e. not appearing in $V$.)
- $E = \{AB : A \to cB \in R\} \cup \{AP : A \to c \in R\}$
- $S = \{S_0\}$
- $A = \{P\}$

- $l(AB) = \begin{cases} \{c : A \to cB \in R\} & B \neq P \\ \{c : A \to c \in R\} & B = P \end{cases}$

It is clear that $M$ is an $\varepsilon - NFA$.

**claim:** $L(G) = L(M)$

**pf:** $(\subseteq)$ Take $w = a_1...a_n \in l(G)$. So there is some derivation; say:

$$S \Rightarrow w_1A_{i_1} \Rightarrow ... \Rightarrow w_1..w_kA_{i_k} \Rightarrow w_1...w_kw_{k+1} = w$$

Remark: $G$ might not build $w$ letter by letter; i.e. the $w_i$ could be words. By construction, we have the following edge path in $(\Omega, E)$:

$s_0A_{i_1}...A_{i_k}P$ \hfill (*)

Moreover, $w = w_1...w_{k+1}$ is a label for $(\text{*)}$ and $P$ is an accepting state; hence $w \in l(M)$. 

7
(≥) Let \( w = a_1...a_n \in L(M) \). So there is some edge path \( s_0A_{i_1}...A_{i_k}P \) with label \( w_1...w_k = w \). This implies that the following derivation exists in \( G \):
\[
S \Rightarrow w_1A_{i_1}... \Rightarrow w_1...w_{k-1}A_{i_{k-1}} \Rightarrow w_1...w_k
\]
So \( a_1...a_n \in L(G) \) and we have the claim. □
This finishes the proof and we conclude the theorem.

**Remark:** We have seen that not all context free languages are regular. So one might attempt to devise some sort of more powerful automaton to recognize these languages. To accomplish this, we might ask what we must add to a finite automaton so that the previous theorem will hold for all context free languages. NOTE: the machine we build, will appear to be substantially different from our notion of finite automata as Lewis defines finite automata differently. Let us consider a context free language and try to understand the problems which an automaton must overcome. Fix some alphabet \( \Sigma \) and let \( L = \{ww^R : w \in \Sigma^* \} \). Now a machine which recognizes \( L \) by reading from left to right - i.e. like an automaton - must "remember" the first part of the input string and compare it against the second piece. So our automata must incorporate a form of memory.

**Natural question:** What should be stored in the memory?

**Answer:** (possible) In the proof of the above theorem we could code up rules of the form \( A \rightarrow cB \) by labeled edges between vertices \( A \) and \( B \). The problem for a general context free grammar is there are rules of the form \( A \rightarrow cBd \). So we need a way to remember \( d \). Thus we create a form of memory which records the \( d \) in a particular order. This memory is the stack or pushdown store.

**Formal Construction:**

**Definition:** A pushdown automaton is a six tuple \( M = (K, \Sigma, \Gamma, \Delta, s, F) \) where

- \( K \) is a finite set of states
- \( \Sigma \) is an alphabet (called input symbols)
• $\Gamma$ is an alphabet (called stack symbols)
• $s \in K$ is the initial states
• $F \subseteq K$ is the set of final states
• $\Delta$ is the transition relation and is a finite subset of: $(K \times \Sigma^* \times \Gamma^*) \times (K \times \Gamma^*)$

**Intuition:** If $((p, u, \beta), (q, \gamma)) \in \Delta$ then whenever $M$ is in state $p$ with $\beta$ at the top of the stack we have the following possible operation: $M$ may read $u$ from the input, replace $\beta$ by $\gamma$ on the top of the stack, and enter state $q$.

**Definitions:**
• $((p, u, \beta), (q, \gamma)) \in \Delta$ is called a transition.
• To push a symbol is to add it to the top of the stack.
• To pop a symbol is to remove it from the top of the stack.

**Example:** The transition $((p, u, e), (q, a))$ pushes $a$ while $((p, u, a), (q, e))$ pops $a$.

**Definition:** A configuration $(k, e, g)$ of a pushdown automaton $M$ is defined to by a member of $K \times \Sigma^* \times \Gamma^*$ where
• $k$ is the state of the machine
• $e$ is the portion of the input yet to be read.
• $g$ is the pushdown store read top down.

E.g. If $(q, w, abc)$ is a configuration then $a$ is the top of the stack and $c$ is the bottom.

**Definition:** For every $((p, u, \beta), (q, \gamma)) \in \Delta$ and $x \in \Sigma^*$, $\alpha \in \Gamma^*$ we define:

$$(p, ux, \beta\alpha) \vdash_M (q, x, \gamma\alpha)$$
We say that $\vdash_M$ holds between configurations when they can be represented in the above form for some transition, $x$, and $\alpha$.

**More Definitions:**

- The reflexive, transitive closure of $\vdash_M$ is denoted by $\vdash^*_M$.
- We say that $M$ accepts a string $w \in \Sigma^*$ iff $(s, w, e) \vdash^*_M (p, e, e)$ with $p \in F$.
- Alternatively: $M$ accepts a string $w \in \Sigma^*$ iff there is a sequence of configurations $C_0, .., C_n$ with $n \geq 0$, $C_0 = (s, w, e)$, $C_n = (p, e, e)$ $p \in F$, and
  
  $C_0 \vdash_M C_1 ... \vdash_M C_n$

- A sequence $C_0, .. C_n$ such that $C_i \vdash_M C_{i+1}$ $0 \leq i \leq n - 1$ is called a computation of length $n$. (alt: a computation with $n$-steps)

- The language accepted by $M$ - denoted $L(M)$ - is the set of all strings accepted by $M$.

**Theorem:** The class of languages accepted by pushdown automata is exactly the class of context free languages.

**proof:**
See text reference page 112.

**Example:**

**Goal:** Create a pushdown automaton $M$ which accepts $L = \{wcw^R : w \in \{a, b\}^*\}$.

E.g. $ababcababa \in L$.

Let $M = (K, \Sigma, \Gamma, \Delta, s, F)$ where

- $K = \{s, f\}$
- $\Sigma = \{a, b, c\}$
- $\Gamma = \{a, b\}$
\begin{itemize}
  \item $F = \{ f \}$
  \item $\Delta$ contains the following transitions
    \begin{enumerate}
      \item $((s, a, e), (s, a))$
      \item $((s, b, e), (s, b))$
      \item $((s, c, e), (f, e))$
      \item $((f, a, a), (f, e))$
      \item $((f, b, b), (f, e))$
    \end{enumerate}
  \end{itemize}

The automaton operates in the following manner: As it reads the first half of its input, it remains in its initial state $s$ and uses transitions 1 and 2 to transfer symbols from the input string onto the pushdown stack. When the machine sees $c$ in the input string, it switch from state $s$ to state $f$ without changing the stack. Hereafter only transitions 4 and 5 are used. these permit removal of the top symbol on the stack provided that it matches the next input symbol. If there is no match then the operation ceases. If the automaton reaches the configuration $(f, e, e)$ then the input is in the form $wcw^R$ and the automaton accepts the input. On the other hand, if the automaton detects a mismatch between input and the stack symbols, or if the input is exhausted before the stack is emptied, then it does not accept.

For a concrete illustration, consider the following table:

<table>
<thead>
<tr>
<th>State</th>
<th>Unread Input</th>
<th>Stack</th>
<th>Transition Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>$abcbba$</td>
<td>$e$</td>
<td>$-$</td>
</tr>
<tr>
<td>$s$</td>
<td>$bbcbba$</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>$s$</td>
<td>$cbba$</td>
<td>$ba$</td>
<td>2</td>
</tr>
<tr>
<td>$s$</td>
<td>$cbba$</td>
<td>$bba$</td>
<td>2</td>
</tr>
<tr>
<td>$f$</td>
<td>$bbba$</td>
<td>$bba$</td>
<td>3</td>
</tr>
<tr>
<td>$f$</td>
<td>$bbba$</td>
<td>$ba$</td>
<td>5</td>
</tr>
<tr>
<td>$f$</td>
<td>$ba$</td>
<td>$ba$</td>
<td>5</td>
</tr>
<tr>
<td>$f$</td>
<td>$a$</td>
<td>$a$</td>
<td>5</td>
</tr>
<tr>
<td>$f$</td>
<td>$e$</td>
<td>$e$</td>
<td>4</td>
</tr>
</tbody>
</table>

$\dagger$ So we conclude that $abcbba$ is accepted by $M$. 