COMPUTATION: DAY 5

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Friday, February 20, 2026.

1. Context Free Grammars

1.1 Pushdown automata (PDAs) are more powerful than finite automata but not yet general computing devices. The languages they recognize are called *context-free languages (CFLs)*. Just as regular languages can be described by regular expressions, CFLs can be described by *context-free grammars (CFGs)*.

1.2 The class of all CFLs is a Kleene algebra under union, concatenation, and star, but unlike regular languages, it is *not closed under complement or intersection*. PDAs are inherently non-deterministic, so while membership of a string can be verified along some computational path, non-membership cannot generally be determined in the same way.

Definition 1.1. A *Context Free Grammar* (CFG) is a mathematical structure $\langle V, \Sigma, R, S\rangle$, where

- V is a finite set of *variables*, also called *non-terminals*.
- Σ is a finite set of *terminals*.
- R is a finite set of *rules*,

$$R = \{\, X \to w \mid X \in V,\, w \in (V \cup \Sigma)^* \,\}.$$

- $S \in V$ is the start symbol.

Definition 1.2 (Derivation)**.** Given a CFG $G = \langle V, \Sigma, R, S\rangle$, a *single-step derivation* has the form

$$uXv \to uwv,$$

where $u, v \in (V \cup \Sigma)^*$, $X \in V$, and $X \to w \in R$. A *derivation* is a finite sequence of single-step derivations.

Definition 1.3 (Language)**.** Given a CFG $G = \langle V, \Sigma, R, S \rangle$, the *language generated* by G, denoted $\mathcal{L}(G)$, is the set of all strings of terminals $v \in \Sigma^*$ such that there exists a derivation from S to v:

$$\mathcal{L}(G) = \{ \, v \in \Sigma^* \mid S \text{ derives } v \, \}.$$

1.3 The non-regular language $L = \{ \, a^i b^i \mid i \geq 0 \, \}$ is a CFL. It has the grammar,

$$S \longrightarrow a\,S\,b \;\mid\; \varepsilon,$$

where S is the only variable, a and b are the terminals, and we use a compact notation in which multiple rules with the same left-hand side are written on a single line with alternative right hand sides separated by the "or-bar" $|$.

1.4 An example derivation for $aabb$ is,

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow aabb.$$

1.5 A slight modification of the previous grammar,

$$S \longrightarrow a\,S\,b \;\mid\; a\,S \;\mid\; \varepsilon,$$

gives the language $L = \{ \, a^i b^j \mid i \geq j \, \}$. For instance, $aaabb$ is derived,

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow aaaSbb \rightarrow aaabb.$$

Theorem 1.1. Let $A \subseteq \Sigma^*$ be a regular language. Then A is a CFL.

1.6 [Proof] Given a regular language L let $N = \langle Q, \Sigma, \delta, q_o, F \rangle$ be the non-deterministic finite automata accepting in L. Convert N into a grammar as follows:

(1) The set of terminals is Σ.

(2) For each each state $q_i \in Q$, in the state set of M, define a variable $Q_i \in V$ in the CFG.

(3) For each transition $\delta(q_i, \sigma) = q_j$, define a grammar rule $Q_i \rightarrow \sigma Q_j$.

(4) For each final state $Q_i \in F$, define a grammar rule: $Q_i \rightarrow \varepsilon$.

(5) Add the rule $S \rightarrow Q_o$, with S the start variable.

Any derivation $S \rightarrow \sigma$ for $\sigma \in \Sigma^*$ will indicate a valid sequence of state transitions through N from the starting state to an accepting state. Conversely any such sequence of transitions when N is presented σ will provide a sequence of rule applications from S to terminal string σ. Note that all intermediate strings are of the form Xw with $X \in V$ and $w \in \Sigma^*$. $\square$

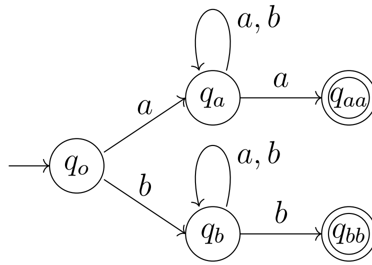

FIGURE 1. NFA accepting $a\{a,b\}^*a \cup b\{a,b\}^*b$

1.7 [Example] Consider the NFA accepting strings in $\{a,b\}^*$ starting and ending on the same letter,

$$(a(a \cup b)^* a) \cup (b(a \cup b)^* b)$$

and the five state NFA in figure 1. The grammar is,

$$\begin{aligned}
S &\longrightarrow Q_o \\
Q_o &\longrightarrow a\,Q_a \mid b\,Q_b \\
Q_a &\longrightarrow a\,Q_a \mid b\,Q_a \mid a\,Q_{aa} \\
Q_b &\longrightarrow a\,Q_b \mid b\,Q_b \mid b\,Q_{bb} \\
Q_{aa} &\longrightarrow \varepsilon \\
Q_{bb} &\longrightarrow \varepsilon
\end{aligned}$$

It is often possible to read a CFG right from the regular expression. Here is how it would be for this regular expression,

$$\begin{aligned}
S &\longrightarrow aTa \mid bTb \\
T &\longrightarrow \varepsilon \mid aT \mid bT
\end{aligned}$$

The first rule expresses the union of two languages; the second rule is a standard way of doing the Kleene Star (on a union).

Theorem 1.2. The class of CFL's is a Kleene Algebra.

1.8 [Proof] We leave this to the reader. The idea is to give grammars for the empty language and the language of just the empty string. And then, given two grammars build a third grammar that is the union, and likewise for the concatenation. And finally, given a grammar create a grammar that is the Kleene star of the original grammar.

2. The usefulness of CFLs

2.1

Why of all classes of languages, should we be interested in CFLs? The class of CFL's is a Kleene algebra, and it does contain the class of regular languages. Both regular expressions and CFL's "compute" in that they can express composition, choice, and some recursion.

2.2 The regular languages express recursion in a way limited to *iteration*, as evidenced by the regular expression star being expressed as a CFG rule,

$$a^* \iff A \to aA \mid \varepsilon.$$

A CFL can express more general patterns of recursion such as nested structures and hierarchical structures. Balanced parentheses, arithmetic expressions, block-structured programming languages, and many fragments of natural language all exhibit recursive, nested form.

2.3 Balanced parenthesis: In order for a computer to understand a math formula, a prerequisite is that it understands the balancing of parentheses. A formula like

$$a(b + c)$$

has balanced parenthesis and has meaning; the formula

$$a(b + (c$$

does not have balanced parentheses and has no meaning.

2.4 We use figure 2 to show how this demonstrates how a CFG can use recursion on substructures in defining a language. For clarity of the typography, we replace (by a and) by b. For example,

$$aababbab \iff (()())().$$

Figure 2 is a graph of the surplus of a's over b's in each prefix of the string. It goes up by one on each a and down by one on each b. To be in the language means,

1. the graph begins and ends at zero and,
2. the graph never goes below zero.

2.5 The recursive structure is:

1. Basis: the empty string is a string of balanced parentheses,
2. Induction: a non-zero string of balanced parenthesis is of the form aSb where S is a string of balanced parentheses or is a sequence of strings of balanced parentheses.

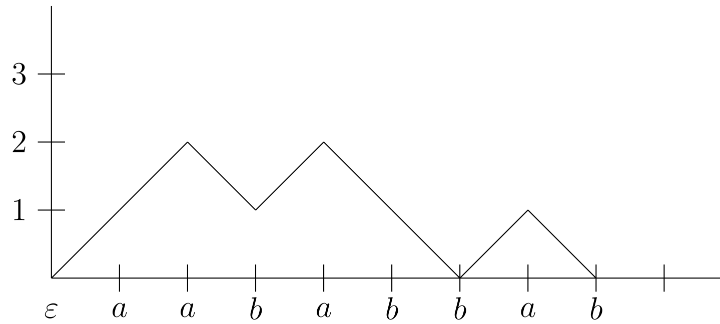

FIGURE 2. Graph of excess opening parenthesis by string position

An induction on the length of the string follows.

2.6 In our example *aababbab* it is not true that *ababba* is balanced — if in the graph we draw a horizontal line at 1 connecting or first and last letter of this substring, the graph of substring will go below that line. But it can be broken into *aababb* followed by *ab*, which are balanced. In turn, the string *aababb* is of the form aSb for the balanced string $S = abab$. And so forth.

2.7 This gives a recursive description of the language,

$$S \longrightarrow SS \mid a\,S\,b \mid \varepsilon$$

2.8 A parse tree for this grammar.

2.9 To prove this grammar is what we want, two directions of suitability must be considered. First, does the grammar generate only balanced strings. Second, does any string of balanced a's and b's get generated by the grammar. The first requirement is easy enough, by inspection of the grammar. The second requirement might need some structure to ascertain.

Lemma 2.1. Any balanced string in $\{\, a, b \,\}$ can be generated by the above grammar.

Proof: Consider a graph of with the horizontal axis the character index in the string, and the vertical axis the excess of a's over b's seen up the the processing of that index, see Figure 2 for an example with the string *aababbab*.

Let s be a string of balanced parenthesis. If $|s| = 0$ the grammar generates the string.

Suppose S is a string of length $2i$, and for all shorter strings the grammar generates such strings. There are two cases,

(1) The graph drawn for this string shows a return to zero in the middle of the graph,

(2) or it does not.

In the first case, the rule $S \rightarrow SS$ applies at the point of return to zero, separating s into $s's''$ each shorter than s and balanced.

In the second case, the rule $S \rightarrow aSb$ applies, separating s in $a\,s'b$, with s' shorter than s and balanced.

In both cases, the induction hypothesis applies. □

2.1. **Same number a's and b's.** The grammar for equal number of a's and b's is given by this gramma,

$$S \quad \longrightarrow \quad a\,S\,b\,|\,b\,S\,a\,|\,S\,S\,|\,\varepsilon$$

The proof uses the graph above. The graph begins and ends at zero but can take negative values. If the graph has some portion negative and some portion positive, it most be zero somewhere between a negative and positive value, The rule $S \rightarrow SS$ applies to drive an induction. Else it remains positive or remains negative, and in the first case the rule $S \rightarrow aSb$ applies, in the second the rules $S \rightarrow bSa$ applies, and that drives the induction.

2.2. **Unequal a's and b's.** In general, the complement of a context free language is not a context free language. However, in this case it is. We work in steps. First we need a grammar for exactly one more a's than b's,

$$\begin{aligned} S_a &\quad \longrightarrow \quad T\,a\,T \\ T &\quad \longrightarrow \quad a\,T\,b\,|\,b\,T\,a\,|\,T\,T\,|\,\varepsilon \end{aligned}$$

To prove this, the graph begins at zero and ends at one. Draw a line at one, and find the rightmost point the graph goes below this line. Such a point must exist, and is happens when an a is countered, and divides the string s into $s'a\,s''$ where s' and s'' are both balanced.

Likewise a grammar with the roles a and b reversed,

$$\begin{aligned} S_b &\quad \longrightarrow \quad U\,b\,U \\ U &\quad \longrightarrow \quad a\,U\,b\,|\,b\,U\,a\,|\,U\,U\,|\,\varepsilon \end{aligned}$$

The the (almost) union of the two languages (rules for U and T above are included in the grammar), and finally the union of the two languages,

$$\begin{aligned} S &\longrightarrow S_a \,|\, S_b \\ T &\longrightarrow a\,T \\ U &\longrightarrow b\,U \end{aligned}$$

3. Limitations of the CFG proof system

3.1 It would seem that the language that compares two strings for equality would not be context free. The set of such strings is,

$$\{ww \mid w \in \Sigma^*\}$$

Note that the set of strings which are palindromes is context free,

$$\{w \in \Sigma^* \mid w = w^R\}$$

where w^R is the string w written backwards. It is generated by the very simple grammar,

$$S \;\to\; aSa \mid bSb \mid \varepsilon$$

It seems like the difference between the copy language of palindromes is that a natural recursion supports matching the letters s at the same distance from the ends of the string,

$$(a \mid b)^i\, x\, (a \mid b)^j\, x\, (a \mid b)^i$$

but not the task of matching the letters x the same distance from the beginning of the string and its midpoint,

$$(a \mid b)^i\, x\, (a \mid b)^j\;\; (a \mid b)^i\, x\, (a \mid b)^j$$

but this is in fact not the problem.

3.2 Suppose we want to test whether the i-th letters in w and v are equal, when $|w| = |v|$ and the two strings are presented as wv. If we are only focused on matching the two letters marked with an x in the strings.

$$\begin{aligned} w &= (a \mid b)^i\, x\, (a \mid b)^j \\ v &= (a \mid b)^i\, x\, (a \mid b)^j \end{aligned}$$

then this observation,

$$(a \mid b)^i\, x\, (a \mid b)^j\;\; (a \mid b)^i\, x\, (a \mid b)^j = (a \mid b)^i\, x\, (a \mid b)^i\;\; (a \mid b)^j\, x\, (a \mid b)^j$$

gives us a natural recursion for the test.

3.3 As an example $abc\,abc$ has all its parsings correct as following,

$$abc\,abc = (\underline{a})(bc\,\underline{a}\,bc) = (a\,\underline{b}\,c)(a\,\underline{b}\,c) = (ab\,\underline{c}\,ab)(\underline{c})$$

However, we need to enforce that *all parsings* hold simultaneously. This we can not do.

3.4 But what we *can do* is show that at least one parsing *does not* hold. Here is that grammar,

$$\begin{array}{lcl} S & \rightarrow & A\,B \mid B\,A \\ A & \rightarrow & aAa \mid aAb \mid bAa \mid bAb \mid a \\ B & \rightarrow & aBa \mid aBb \mid bBa \mid bBb \mid b \end{array}$$

Lemma 3.1. The above grammar generates the language,

$$\{w\,v \in \{\,a, b\,\}^* \mid |w| = |v| \text{ and } w \neq v,\ \}$$

3.5 [Proof] To reduce notational clutter, and string denoted by x, x' or x'' has length i, and any string denoted by y, y' or y'' has length j.

[⇒] Any string generated by the grammar can be written as,

$$s = x\,a\,x'\,y\,b\,y' \text{ or } x\,b\,x'\,y\,a\,y'$$

The string $x'y$ can be re-parsed as $y''x''$ because both strings are of length $i + j$, so,

$$s = x\,a\,y''x''\,b\,y' \text{ or } x\,b\,y''x''\,a\,y'.$$

These strings are of the form $w\,v$ with $w \neq v$. For instance, in the first case, $w = x\,a\,y''$ and $v = x''\,b\,y'$.

This is a picture of what is happening. The black dot (•) is any a or b. Here is two 4 length strings re-parsed as a 5 length string followed by an 3 length string,

$$(\bullet \quad \bullet \quad a \quad \bullet \,|\, \bullet) \quad (\bullet \quad b \quad \bullet)$$

Here is two 5 length strings re-parsed as a 3 length string followed by a 7 length string,

$$(\bullet \quad a \quad \bullet) \quad (\bullet \quad \bullet \,|\, \bullet \quad b \quad \bullet \quad \bullet \quad \bullet)$$

[⇐] Consider $s = v\,w$ where v and w are equal length and unequal. v and w must be unequal in at least one place, so either

$$v = x\,a\,y \text{ and } w = x'\,b\,y'$$

or

$$v = x\,b\,y \text{ and } w = x'\,a\,y'.$$

Write $y\,x' = x''\,y''$ and so,

$$s = x\,a\,x''\,y''\,b\,y'$$

or

$$s = x\,b\,x''\,y''\,a\,y'$$

which are of the form of a string generated by the language. □

Theorem 3.1. The class of context free languages is not closed by complements. That is, there as a context free language whose complement is not a context free language.

3.6 [**Proof**] Our example is almost there. First we need to know that the language $\{ww\}$ is not context free. For that we need the CFL version of the pumping lemma. Assuming this, the complement of that language is the language,

$$\{uv \mid |u| = |v|,\ u \neq v\} \cup \{v \mid \text{the length of } v \text{ is odd}\}$$

and this language is context free. It is the union of the context free language of the previous theorem and the obviously CFL strings of odd length. □

3.7 What proved to be the difficulty of considering the complement of a CFL, at least in this case, is that one derivation was sufficient to prove that two strings are unequal. However to prove two strings are equal requires multiple statements. Which is something that cannot be done by a context free grammar.

4. The Pumping Lemma for CFL's

4.1 To complete the demonstration of a language which is context free, whose complement is not context free, we need to show that the copy language is not context free. We do this with a form of the pumping lemma for context free languages.

Theorem 4.1. Given a CFL $\mathcal{L}$, there exists a p such that for all $s \in \mathcal{L}$, $|s| \geq p$ there exists string u, v, x, y and z such that,

(1) $s = uvxyz$,

(2) $|vxy| < p$,

(3) $vy \neq \varepsilon$

(4) $\forall i, uv^ixy^iz \in \mathcal{L}$.

4.2 [**Proof sketch**] The proof is by considering the parse tree. Any path of depth exceeding the number of variable in a grammar for the language will have a repeated

variable on the path. The section between occurrences of this variable can be removed or repeated, giving the pumped versions of the string.

4.3 The language $w\,w$ is not context free. Select $s = a^p b^p a^p b^p$. If the language were a CFL the pumping lemma would find a vxy that can be pumped. Investigating cases, every such vxy cannot be pumped, so the language is not a CFL.

4.4 The class of CFL's is not closed by Intersection. Given CFL's A and B, it is not always true that $A \cap B$ is a CFL. Consider the languages,

$$\begin{aligned} A &= \{\, a^i b^i c^j \,|\, i, j \geq 0 \,\} \\ B &= \{\, a^i b^j c^j \,|\, i, j \geq 0 \,\} \end{aligned}$$

The language

$$A \cap B = \{\, a^i b^i c^i \,|\, j \geq 0 \,\}$$

is not context free. This is can proved with the pumping lemma.