1 What is a CFG?

A CFG is a compact description of a context-free language CFL.

Let’s get some terminology straight: Grammar versus Language.

We have already seen it: Regular expressions (the “grammar”) versus regular languages (potentially infinite set of strings).

All regular languages are context-free (but not vice versa).

A CFL is not regular, but still has a simple enough structure that it can be recognized using a single-stack automaton.

They arise in programming languages and all kinds of other situations.

The syntax of regular expressions is a CFL!

The key pattern in a CFG is “(((()())))” or “((()))(())(((()))())”

In fact, all CFL strings are “grown inside out”

OK let’s illustrate these facts now.

• ($\{0, 1\}, \{S\}, S, \emptyset$) is a CFG

• ($\{0, 1\}, \{S, T\}, S, P$) is a CFG where $P$ is this set of rules:
  - $S \rightarrow 0T \mid 0$
  - $T \rightarrow 1T \mid 1$

  Notice that this CFG is the same as the one below:
  - $S \rightarrow 0T$
  - $S \rightarrow 0$
  - $T \rightarrow 1T$
  - $T \rightarrow 1$

• ($\{0, 1\}, \{S, T, U\}, S, P$) is a CFG where $P$ is this set of rules:
  - $S \rightarrow TU$
  - $T \rightarrow 0T \mid 0$
  - $U \rightarrow 1U \mid \epsilon$ -- epsilon

• Strictly this is a CFG although no one would want to use it: ($\{0, 1\}, \{S\}, S, P$) is a CFG where $P$ is this set of rules:
  - $S \rightarrow S$

• Strictly this is a CFG although no one would want to use it: ($\{0, 1\}, \{S\}, S, P$) is a CFG where $P$ is this set of rules:
  - $S \rightarrow \epsilon$
Strictly this is a CFG although no one would want to use it: $(\{0, 1\}, \{S, T\}, S, P)$ is a CFG where $P$ is this set of rules:

$S \to \epsilon$

$T \to 1$

$G = (\{0, 1\}, \{S, T, U\}, S, P)$ is a CFG where $P$ is this set of rules:

$S \to TU$

$T \to 0T \mid 0$

$U \to 1U \mid 1$

although you should use a regular-expression whenever possible, i.e., $0^+1^+$.

In $G$ above, see how you lose control of the “balance” between 0s and 1s. This is what regular languages do: “forget counts”

CFGs also don’t strictly count, but can match up counts!

$G_{bal} = (\{0, 1\}, \{S\}, S, P)$ is a CFG where $P$ is this set of rules:

$S \to 0S1 \mid \epsilon$

grows inside out. It matches 0 with a 1, but then once the match is seen it “forgets” the exact numbers of 0s and 1s.

2 Tricks to Evolve a CFG “Inside-Out”

Let’s understand the “inside out” trick well, because this is how you will be designing most CFGs. Here on, I’ll merely show you the rules:

- What does this CFG generate?
  
  $S \to 0S1S \mid 1S0S \mid \epsilon$

- Do things change if I add one more rule?
  
  $S \to 0S1S \mid 1S0S \mid SS \mid \epsilon$

- When asked to do “obtain a CFG for all strings where the number of zeros are twice as many as the number of 1s”, let us consider these attempts:
  
  - How about:
    
    $S \to 001S \mid 010S \mid 100S$

  - How about:
    
    $S \to 0S0S1 \mid 0S1S0 \mid 1S0S0$

  - Do we need this:
    
    $S \to S0S0S1S \mid S0S1S0S \mid S1S0S0S$

- When asked to design a grammar for $\{0^n1^m \mid n, m \geq 0\}$ go tell them “use a regular expression!”
• When asked to design a grammar for \( G_{001} = \{0^{2n}1^n \mid n \geq 0\} \), can you tell them “use a regular expression?” Build sufficient intuitions. If sure it is not a reg language, then use the Pumping Lemma.

• A CFG for \( G_{001} \): wrong attempt (why)?
  
  \[
  \begin{align*}
  S &\rightarrow T \ U \\
  T &\rightarrow 00 \ T \mid e \\
  U &\rightarrow 1 \ U \mid e
  \end{align*}
  \]

• The way to think of a CFG for \( G_{001} \): you need to grow inside out! You can grow “00.1” inside out:
  
  \[
  S \rightarrow 00S1 \mid e
  \]

• Cool fact: Any CFG over a singleton alphabet is regular. What does this CFG generate?
  
  \[
  S \rightarrow ( \ S \ ) \mid S S \mid e
  \]

What does this CFG generate?

\[
S \rightarrow ( \ S \ ( \mid S S \mid e
\]

What does this CFG generate?

\[
S \rightarrow 0 S 0 \mid S S \mid e
\]

• Consider \( L = \{a^i b^j c^k \mid i, j, k \geq 0 \land if(i = 0) \text{then } j = k\} \).

• Regular? (Naah!)

• How to pump? Not beginning with \( a \)! Once you increase or decrease \( a \) you don’t fall out of the language!

• Reverse and pump? Yes! IF original regular, reversal preserves regularity. But then can mangle reversal by pumping it out of shape. Hence original can’t be regular.

• USING and ABUSING closure arguments:
  
  - USE: Show \( \{w \mid w \text{ has equal number of 0s and 1s}\} \) is non-regular.
    
    Hint: intersect with \( 0^* \ 1^* \). The language then becomes what?
    
    Can you show that language non-regular?
    
    Then original language is regular!
  
  - ABUSE: Show \( L_{eq} = \{w \mid w \text{ has equal number of 0s and 1s}\} \) is non-regular.
    
    BAD Hint: intersect with \( 2 \ 2^* \) (some junk). Resulting language is EMPTY.
    
    Empty is REGULAR.
    
    Hence original language is regular! (Naah!)

3 Consistency and Completeness

Consistency: all the generated strings are correct according to the language. Example: Is this palindrome? (Nahh!)
S \rightarrow T U
T \rightarrow 0 T 1 | 1 T 0 | e
U \rightarrow 0 U 1 | 1 U 0 | e

Consistency: Is this palindromic? (Yes, but not all are captured!)

S \rightarrow 0 S 0 | 1 S 1 | e

Completeness: Fill in the missing palindromes (for instance). What are they?
Do we need to add the $S \ S$ part to make this language complete with respect to $L_{eq}$?

S \rightarrow 0 S 1 S | 1 S 0 S | S S | e

4 When is Something not a CFL?

$L_{ww} = \{ww \mid w \in \{0,1\}^*\}$ is not a CFL. A proof has to wait.

5 Closure under Kleene Ops. (Groan, not under Compl.!!)

CFLs are closed under all Kleene operators (union, concatenation, star).
CFLs are not closed under complementation. $L_{ww}$ is not a CFL (believe me). But its complement is (will write a CFG).

S \rightarrow T U | U T | Oddlen
T \rightarrow P 0 P
U \rightarrow Q 1 Q
P \rightarrow 0 | 1
Q \rightarrow 0 | 1
Oddlen \rightarrow P | P P Oddlen

6 Ambiguity and Inherent Ambiguity

Have multiple parses.

S \rightarrow E + E | E * E | num

Inherent is when every CFG is ambiguous (for some string). $\{0^i 1^j 2^k \mid i, j, k \geq 0 \land (i = j \lor j = k)\}$