CS Theory Fall 2022 Handout 6a: Context Free Languages

Alice Chen & Leonidas Pappajohn
yc3877@columbia.edu & lgp2116@columbia.edu

Credit to Fall 2020 TA: Bryce Monier
bjm2190@columbia.edu

October 20, 2022

1 CFG/CFL Overview

1.1 Key terms / facts

(a) A *context-free grammar (CFG)* is represented as a 4-tuple (V, Σ, R, S), where

- V is a set of *variables* or *nonterminals*
- Σ is a set of alphabet symbols or *terminals*
- R is a set of rules of the form $A \to \alpha$, where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$
- $S \in V$ is the designated start variable.

(b) Let $G = (V, \Sigma, R, S)$ be a grammar. Let $\alpha A \beta$ be a string of variables and alphabet symbols, or $\alpha, \beta \in (V \cup \Sigma)^*$ and $A \in V$. If there is a rule in R of the form $A \to \gamma$, where $\gamma \in (V \cup \Sigma)^*$, we write

$$\alpha A \beta \Longrightarrow \alpha \gamma \beta$$

and say $\alpha A \beta$ yields $\alpha \gamma \beta$. If there is a finite sequence of w_i so that

$$w \Longrightarrow w_1 \Longrightarrow \cdots \Longrightarrow w_n \Longrightarrow z$$

we write $w \stackrel{*}{\Longrightarrow} z$ and say w derives z.

(c) The language generated by a CFG $G = (V, \Sigma, R, S)$ is defined as

$$L(G) = \{w \in \Sigma^* \mid S \stackrel{*}{\Longrightarrow} w\}$$

If for a language L there exists a CFG G satisfying $L(G) = G$, we say L is a *context-free language (CFL)*.

(d) **Fact:** the set of regular languages is a proper subset of the set of context-free languages. In class, we saw one proof that regular languages are context-free, by transforming regular expressions to equivalent CFGs. A different proof follows from transforming DFAs to CFGs – we will show a proof below. A third proof follows from transforming NFAs to PDAs (which is an immediate transformation, as NFAs can be viewed as a special case of PDAs) – we will also mention this below. On the other hand, there are non-regular, context-free languages, such as $\{a^i b^i : \forall i \geq 0\}$. Thus, the regular languages are a proper subset of CFL's.

(e) A CFG (Q, Σ, V, R) is *right-linear* if every rule in R is of the form $A \to \epsilon$ or $A \to aB$ where $A, B \in V, a \in \Sigma$. We stated the theorem that a language is regular if and only if it is generated by a right-linear CFG. Below, we will prove one direction of this theorem (we will prove that every regular language has a corresponding right-linear CFG).

(f) We have a *leftmost derivation* if we replace the leftmost variable with one of its production bodies in every derivation step. For a given grammar G and a string $w \in L(G)$, each leftmost derivation of w corresponds to a unique parse tree. Similarly, each parse tree for w corresponds to a unique leftmost derivation for w.

(g) A CFG is *ambiguous* if we can find a string w in Σ^* having two different parse trees with S as root that both generate w. This also means that the string w has two distinct leftmost derivations. A CFG is *unambiguous* otherwise.

(h) A CFL is *inherently ambiguous* if all of its grammars are ambiguous. In other words, no matter how you formulate a grammar for the language, there will always be some string that has two different leftmost derivations.

(i) Remember DFA's and NFA's? Well, CFG's come with their own kinds of automata called *pushdown automata (PDA)*. We define these further down. Simply put, they are NFA's associated with a stack. **A language is context free if and only if there exists a PDA which recognizes it.**

(j) Remember the pumping lemma for natural languages? Well, CFL's have their own pumping lemma, often called the *tandem pumping lemma*. The lemma states that **if L is a CFL, L has some associated pumping length p such that** $\forall w \in L \ |w| \geq p, \ \exists u, v, x, y, z$

i) $w = uvxyz$

ii) $|vxy| \leq p$

iii) $|vy| > 0$

iv) $\forall i = 0, 1, 2, \ldots \ uv^i xy^i z \in L$

To prove a language is not a CFL, one might show that there exists a string w such that no such parsing exists.

Beware: As is the case with regular languages, it is possible for a language to 'pass' the tandem pumping lemma but **not** be context-free.

1.2 PDA Definitions

(a) A *pushdown automaton* (PDA) is similar to a non-deterministic finite automaton, except for an additional stack. It is represented as a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$

- Q is the set of states,
- Σ is the input alphabet (and $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$),
- Γ is the stack alphabet (and $\Gamma_\epsilon = \Gamma \cup \{\epsilon\}$),
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to P(Q \times \Gamma_\epsilon)$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

(b) A pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ accepts input w if there exists an accepting computation of M on w. In more detail, M accepts w if w can be written as $w = w_1 w_1 w_2 \dots w_m$, where each $w_i \in \Sigma_\epsilon$ and sequence of states $r_0, r_1, \dots, r_m \in Q$ and strings $s_0, s_1, \dots, s_m \in \Gamma^*$ exist that satisfy the following three conditions. The strings s_i represent the sequence of stack contents that M has on the accepting branch of the computation.

1. M starts out properly, in the start state and with and empty stack: $r_0 = q_0$ and $s_0 = \epsilon$.
2. M moves properly according to the state, stack, and next input symbol: For $i = 0, \dots, m-1$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = at$ and $s_{i+1} = bt$ for some $a, b \in \Gamma_\epsilon$ and $t \in \Gamma^*$
3. An accept state occurs at the end: $r_m \in F$

(c) A state diagram for a pushdown automaton can be drawn as shown below. The transition is taken as read "a" from input, pop "b" from stack, push "c" to stack

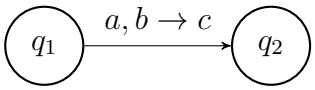


Figure 1: Example diagram for pushdown automaton

(d) **Fact:** Pushdown automata are equivalent in power to context-free grammars. A language L is context free if and only if there exists a pushdown automaton that recognizes L.

(e) Note that this immediately gives yet another proof that every regular language is context free. This is because for every regular language there exists a NFA that recognizes it, therefore there exists a PDA that recognizes it (the PDA can just ignore the stack / always pop ϵ and push ϵ to stack, and otherwise do the same as the NFA in terms of state transitions).

1.3 Closure properties

We saw in class some useful closure properties, corresponding to the regular operations:

(a) If L_1 and L_2 are context-free, so is $L_1 \cup L_2$

(b) If L_1 and L_2 are context-free, so is $L_1 \circ L_2$

(c) If L is context-free, so is L^*.

In contrast, some of the closure properties of regular languages do not work with context-free languages:

(a) If L_1 and L_2 are context-free, $L_1 \cap L_2$ might not be.

(b) If L is context-free, $\overline{L}$ (which we use to denote the complement of L) might not be.

Caution: with closure properties, we can only use what we prove. In particular, for context-free L_1 and L_2, $L_1 \cap L_2$ might be context free! But it also might not be.

As a simple example, consider a CFL L (over the alphabet Σ) as well as the regular language Σ^* (which is also a CFL since every regular language is a context free language). Consider, then, that $L \cap \Sigma^* = L$ is once again context-free.

Similarly, if any language L is regular, then $\overline{L}$ is also regular. But both languages are also context-free.

Exercise 1: As a more interesting example, show that the complement of $L = \{a^i b^i : i \geq 0\}$ is context-free.

Exercise 2: A further nice closure property is that if L_1 is context-free and L_2 is *regular*, then $L_1 \cap L_2$ is context-free. Can you prove it? (Hint: use PDAs).

1.4 CFG Example

Consider a CFG with the following rules. This grammar generates a subset of English sentences. Here are what the variables stand for: NP = Noun Phrase, VP = Verb Phrase, DT = Determiner, PP = Prepositional Phrase,

NN = Noun, VB = Verb, PREP = Preposition.

$$
\begin{aligned}
S &\longrightarrow NP\ VP \\
NP &\longrightarrow DT\ \ NN \mid NP\ \ PP \\
VP &\longrightarrow VB\ \ NP \mid VP\ \ PP \\
PP &\longrightarrow PREP\ \ NP \\
DT &\longrightarrow the \mid a \\
NN &\longrightarrow man \mid dog \mid telescope \\
VB &\longrightarrow saw \\
PREP &\longrightarrow with
\end{aligned}
$$

We will demonstrate the ambiguity derived from prepositional phrase attachment in English using the sentence *the man saw a dog with a telescope*.

Figure 2: PP attached to NP

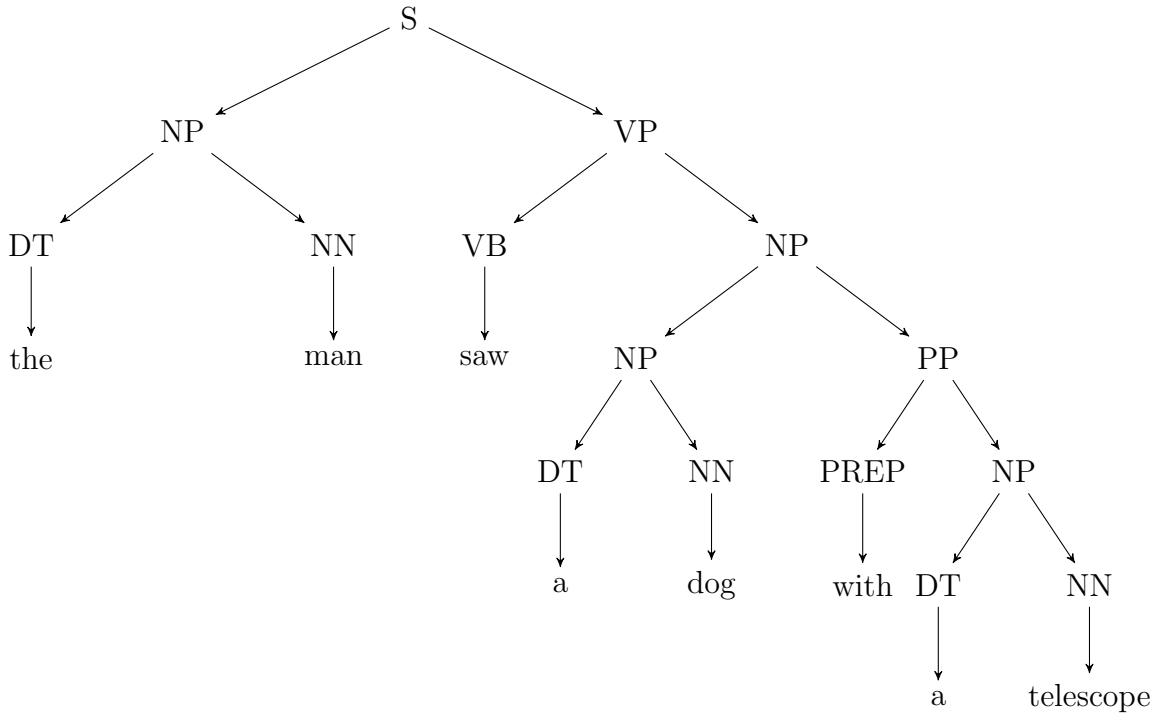
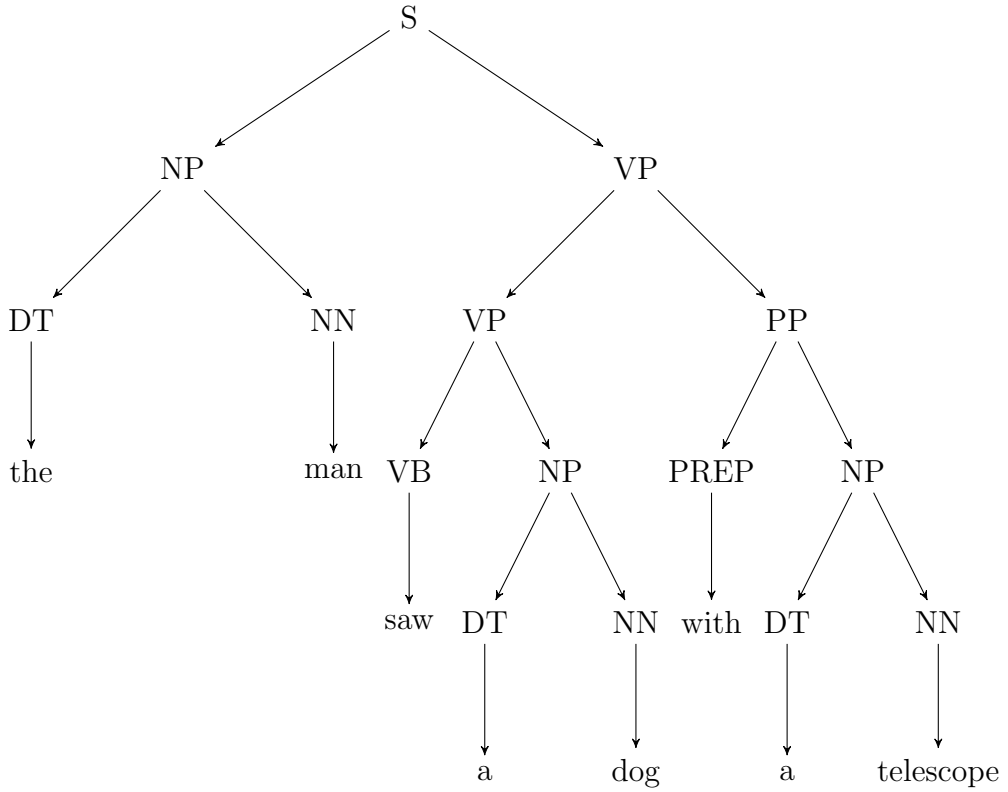


Figure 3: PP attached to VP



Example 2. Construct a PDA M_1 that recognizes $L = \{0^n 1^n | n \geq 0\}$

2 Regular Languages $\subsetneq$ Context Free Languages

Recall that in class, we showed a way to convert any regular expression into an equivalent CFG. This proved that regular languages are a subset of context free languages. As mentioned above (and mentioned in class), a different way to prove it is to note that NFAs can be viewed as a special case of PDAs, so if a language is regular, it has an NFA, and therefore it also has a PDA and is context free. Next, we will present an alternative proof, this time based on DFAs.

Note: We prove this by proving one direction of a theorem mentioned in class: that every regular language is generated by a right linear grammar. We give this proof in order to provide students with extra practice. The proof is not part of the required material, although the statement is.

Theorem. *If a language L over the alphabet Σ is regular, then L is context-free.*

Proof. Since L is regular, we can construct a DFA $M = (Q, \Sigma, \delta, Q_0, F)$ such that $L(M) = L$. Note we will use *upper-case* letters Q_i to represent states in Q (corresponding to nonterminals in the grammar). So $Q_0 \in Q$ for example is the designated start state. We will construct a CFG $G = (V, \Sigma, R, S) = (Q, \Sigma, R, Q_0)$ satisfying $L(G) = L$. Conceptually, we simulate the computation on the DFA using a CFG.

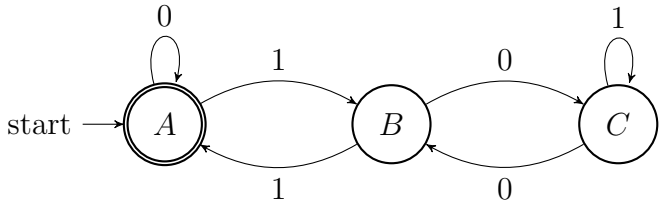
The set of nonterminal for G will be identical to the set of state for the DFA, $V = Q$. The starting nonterminal will be the start state of M, $S = Q_0$. Then we define R by the rule below:

$$\forall (Q_i, a) \in Q \times \Sigma, \text{ if } \delta(Q_i, a) = Q_j, \text{ we add a rule } \mathbf{Q_i} \longrightarrow \mathbf{aQ_j} \text{ to } R$$

and further, we add a rule to R

$$\forall Q_i \in F, \text{ add a rule } \mathbf{Q_i} \longrightarrow \epsilon \text{ to } R$$

Example: We will give a simple example of a DFA and the corresponding grammar. Take the DFA below, which recognizes binary numbers that are divisible by 3:



The corresponding grammar from the construction described above would

bet $G = (\{A, B, C\}, \{0, 1\}, R, A)$ with production rules R given by:

$$\begin{aligned} A &\longrightarrow 0A \mid 1B \mid \epsilon \\ B &\longrightarrow 0C \mid 1A \\ C &\longrightarrow 0B \mid 1C \end{aligned}$$

Justification: We claim that $L(G) = L(M)$. First, suppose $w \in L(G)$, where $w = w_1 w_2 \dots w_n$ with each $w_i \in \Sigma$. If $w \in L(G)$, there is a computation such that $Q_0 \stackrel{*}{\Longrightarrow} w$. Inspecting all of the rules we added to R, we see there will always be at most a single non-terminal, and it will be the rightmost character of the current string. Further, we see the last production rule to generate w must be of the form $Q_i \to \epsilon$ for some $Q_i \in F$. We can only apply such a production once, at the last step of the derivation of w, because this rule will result in a string with no nonterminals. Building a string from the production rules will build from the beginning of w to the end, adding a single character to the prefix at each step. In view of all of these considerations, a derivation of w must look like:

$$Q_0 \Longrightarrow w_1 Q_{i_1} \Longrightarrow w_1 w_2 Q_{i_2} \stackrel{*}{\Longrightarrow} w_1 \dots w_n Q_{i_n} \Longrightarrow w_1 \dots w_n = w$$

where each derivation is of the form $Q_{i_k} \to w_{k+1} Q_{i_{k+1}}$. for $k = 0, \dots, n-1$, and $Q_{i_n} \to \epsilon$ for $k = n$. But because of how we defined R, this means precisely that there is a computation on the DFA M given by

$$Q_0 \xrightarrow{w_1} Q_{i_1} \xrightarrow{w_2} Q_{i_2} \xrightarrow{w_3} \cdots \xrightarrow{w_n} Q_{i_n}$$

with $Q_{i_n} \in F$. This is an accepting computation of w on M, so we conclude $w \in L(M)$.

On the other hand, suppose $w \in L(M)$. Then there exists a computation

$$Q_0 \xrightarrow{w_1} Q_{i_1} \xrightarrow{w_2} Q_{i_2} \xrightarrow{w_3} \cdots \xrightarrow{w_n} Q_{i_n}$$

with Q_{i_n}. But this means precisely that we can apply the rule corresponding to each computation $\delta(Q_{i_k}, w_{k+1}) = Q_{i_{k+1}}$ to get a derivation in G:

$$Q_0 \Longrightarrow w_1 Q_{i_1} \Longrightarrow w_1 w_2 Q_{i_2} \stackrel{*}{\Longrightarrow} w_1 \dots w_n Q_{i_n} \Longrightarrow w_1 \dots w_n = w$$

So that $Q_0 \stackrel{*}{\Longrightarrow} w$, and $w \in L(G)$. We conclude $L(M) = L(G)$. Again, this argument is not a completely formal proof, but should convince you that our construction works.

We have shown that for every language L recognized by a DFA, there is a grammar that produces L. In fact, the CFG we constructed is right-linear. Thus, we've shown that any DFA can be transformed to an equivalent right-linear CFG. We conclude regular languages $\subset$ context-free languages. $\square$

Extra: The above construction shows that any DFA corresponds to a right-linear grammar. On the other hand, given a right-linear grammar, we can actually reverse the construction above to get an equivalent NFA.

3 Additional Problems

1. Consider the CFG $G_1 = (V, \Sigma, R, S)$ with $V = \{S\}$, $\Sigma = \{'(', ')'\}$, start variable S, and production rules R given by

   $$S \to SS \,|\, (S) \,|\, \epsilon$$

   What language does this grammar generate?

2. Consider the CFG $G_2 = (V, \Sigma, R, S)$ with $V = \{S\}$, $\Sigma = \{a, b, c\}$, start variable S, and with production rules R given by

   $$S \to aS \,|\, aSbS \,|\, c$$

   This grammar models **if-then** and **if-then-else** statements in programming languages where a stands for **if-condition-then**, b for **else**, and c for some other statement. Is the language generated by G_2 regular? Is G_2 ambiguous?

3. Consider the CFG $G_3 = (V, \Sigma, R, S)$ with $V = \{S, B\}$, $\Sigma = \{a, b\}$, with start variable S and production rules R given by

   $$S \to aBa$$

   $$B \to BB \,|\, b \,|\, \epsilon$$

   What language does this grammar generate?

4. For the alphabet $\Sigma = \{a, b, c, d\}$, define the language

$$L = \{cw | w \in \{a, b\}^*, w = w^R\} \cup \{dw | w \in \{a, b\}^*\}$$

Prove that L is context free.

4 Applications / Motivation

In this section, we will discuss the motivations behind understanding context-free languages, and the applications of these understandings. This is not required material, but you might find it interesting.

4.1 What does "Context-Free" mean?

To start off, you might be wondering why this set of languages is called "context-free" ... what is the 'context' and what exactly makes these languages 'free' of it? To put it simply, 'context' refers to the symbols to the left and right of a non-terminal symbol as one is deriving a string. For example, if we have the CFG:

$$S \to aBa$$

$$B \to BB \,|\, b \,|\, \epsilon$$

So, let's say we are creating a string in the language of this CFG, and we currently have the string $abBa$. When looking at the non-terminal B, we don't care about the symbols to the left and right of B – they have no bearing on how we will replace B. This is what makes the language "context-free"... when we see a non-terminal, the symbols around it do not dictate the possible ways in which we can replace it.

What would it look like if we *did* care about the context of non-terminals? Well, that is what we call a *context-sensitive language (CSL)*. In similar fashion to context-free **grammars**, we can make *context-sensitive grammars (CSG)*. In a CSG, all rules are of the form:

$$\alpha A \beta \to \alpha \gamma \beta$$

where

$$A \in V$$

$$\alpha, \beta \in (V \cup \Sigma)^*$$
$$\gamma \in (V \cup \Sigma) \circ (V \cup \Sigma)^*$$

In words, what this rule would mean is that we can replace A with γ when we see the patterns (of non-terminals and terminals potentially) α to the **left** of A and β to the **right** of A. (Additionally, we may have a rule $S \to \varepsilon$ to allow production of ε, if S doesn't appear on the righthandside of other rules).

We will not cover CSG's in this course, but it turns out:

$$\{L \mid L \text{ is a } CFL\} \subsetneq \{L \mid L \text{ is a } CSL\}$$

4.2 What are some applications of CFL's / CFG's

CFL's are most prevalent in the fields of Linguistics and Natural Language Processing (NLP). CFG's are good models for many languages, such as English (consider the CFG from section 1.4). In this context, we can make our non-terminals generally correspond to grammatical groups of words (nouns, verbs) and our terminals correspond to specific words themselves. In NLP, *probabilistic context-free grammars (PCFG)* are often used to express the probability that a non-terminal follows a rule (for example, the probability that the **NOUN** non-terminal turns into *chicken*). In this case, one must make sure that the probabilities of all the rules originating from a non-terminal add up to 1.

Interestingly, however, there are some languages which have structures that cannot be captured by CFL's.

For example, in Dutch, it is possible to have this kind of structure, which loosely mirrors the non-context free language: $\{a^n b^m c^n d^m\}$:

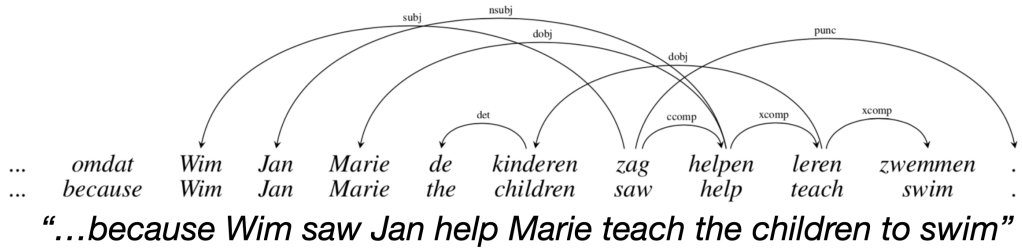


Figure 4: Credit – Professor Daniel Bauer, NLP Fall 2022 Slides