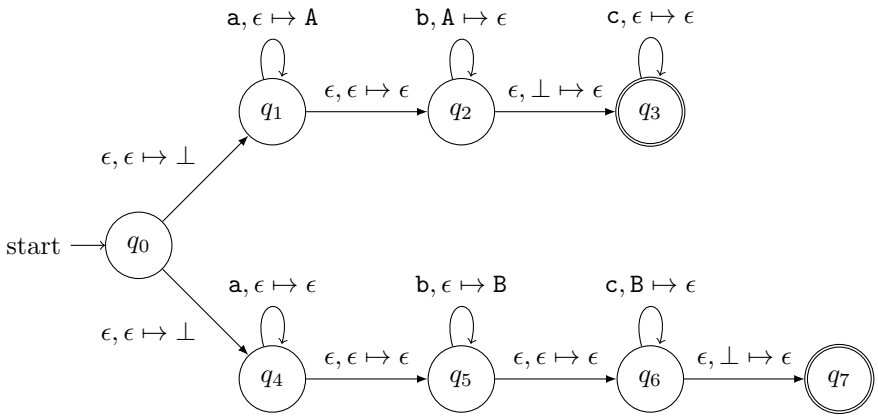
Note that, if the input word contains more right parentheses than left parentheses, then the pushdown automaton will not be able to pop a symbol P from the stack. Similarly, if the input word contains more left parentheses than right parentheses, then it will not be able to pop the "bottom of stack" symbol $\perp$ from the stack. In either case, it becomes stuck in state q_1 and unable to accept the input word.

Example 18. Consider the language $L_{\text{twoequal}} = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } j = k\}$ over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. A pushdown automaton recognizing this language must have two "branches": one branch to handle the case where $i = j$, and one branch to handle the case where $j = k$. Since we don't know in advance which branch we will need to take, we can use the nondeterminism inherent in the pushdown automaton model.

A pushdown automaton recognizing this language would therefore look like the following, where the upper branch handles the case $i = j$ and the lower branch handles the case $j = k$:



4 Equivalence of Models

You may recall from our discussion of regular languages that we proved a couple of exciting results: a language L is regular if and only if there exists a finite automaton recognizing L, and a language L is regular if and only if there exists a regular expression matching words in L. These two results allowed us to establish Kleene's theorem, which brought together all of our different representations of regular languages.

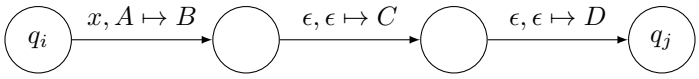
Now that we're focusing on context-free languages, and now that we have two ways of representing context-free languages—namely, context-free grammars and pushdown automata—it would be nice to establish a connection between the two representations. This brings us to yet another exciting result, which will be the focus of this section. Since the overall proof is quite lengthy, we will split the proof of the main result into two parts.

4.1 CFG ⇒ PDA

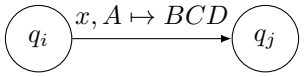
For the first half of our main result, we will show that we can convert any context-free grammar into a pushdown automaton recognizing the language generated by the grammar. Specifically, given a context-free grammar G, we will construct a pushdown automaton $\mathcal{M}$ that functions as a *top-down parser*[6] on its input word w; that is, beginning with the start nonterminal S, $\mathcal{M}$ will repeatedly apply rules from R to check whether w can be generated via a leftmost derivation. If so, then $\mathcal{M}$ will accept w.

[6] We could alternatively construct $\mathcal{M}$ to act as a *bottom-up parser*, where it applies rules backward starting from the input word w to see if the start nonterminal S can be reached. The outcome is the same, however, so we will not discuss this alternative construction here.

Note that, for the purposes of this proof, we will "condense" multiple transitions of our pushdown automaton into one transition; that is, if we have some sequence of transitions



then we will depict this sequence of transitions as one single transition of the form



and we replace the symbol A on the stack with the symbols BCD, in that order from bottom to top.

Lemma 19. *Given a context-free grammar G generating a language $L(G)$, there exists a pushdown automaton $\mathcal{M}$ such that $L(\mathcal{M}) = L(G)$.*

Proof. Suppose we are given a context-free grammar $G = (V, \Sigma_G, R, S)$. We construct a pushdown automaton $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$ that recognizes the language generated by G in the following way:

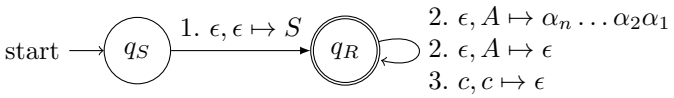
- The set of states is $Q = \{q_S, q_R\}$. The first state, q_S, corresponds to the point during the computation at which the context-free grammar G begins to generate the word. The second state, q_R, corresponds to the remainder of the computation where G applies rules from its rule set.
- The input alphabet is $\Sigma = \Sigma_G$. If $\mathcal{M}$ accepts its input word, then the word could be generated by G, and therefore it must consist of terminal symbols.
- The stack alphabet is $\Gamma = V \cup \Sigma_G$. We will use the stack of $\mathcal{M}$ to keep track of where we are in the leftmost derivation of the word.
- The initial state is $q_0 = q_S$.
- The final state is $F = \{q_R\}$.
- The transition function δ consists of three types of transitions:
  1. **Initial transition**: $\delta(q_S, \epsilon, \epsilon) = \{(q_R, S)\}$. This transition initializes the stack by pushing to it the start nonterminal S, and then moves to the state q_R for the remainder of the computation.
  2. **Nonterminal transition**: $\delta(q_R, \epsilon, A) = \{(q_R, \alpha_n \dots \alpha_2 \alpha_1)\}$ for each rule of the form $A \to \alpha_1 \alpha_2 \dots \alpha_n$, where $A \in V$ and $\alpha_i \in V \cup \Sigma_G$ for all i. Transitions of this form simulate the application of a given rule by popping the left-hand side (A) from the stack and pushing the right-hand side ($\alpha_1 \alpha_2 \dots \alpha_n$) to the stack in its place in reverse order. Pushing the symbols in reverse ensures that the next symbol we need to read (α_1) is at the top of the stack.

     Note that if $n = 0$, then the transition will be of the form $\delta(q_R, \epsilon, A) = \{(q_R, \epsilon)\}$.
  3. **Terminal transition**: $\delta(q_R, c, c) = \{(q_R, \epsilon)\}$ for each terminal symbol $c \in \Sigma_G$. Transitions of this form compare a terminal symbol on the stack to the current input word symbol. If the two symbols match, then the computation continues.

During the computation, after the initial transition is followed, $\mathcal{M}$ follows either nonterminal transitions or terminal transitions until its stack is empty or it runs out of input word symbols. If a nonterminal symbol A is at the top of the stack, $\mathcal{M}$ nondeterministically chooses one of the rules for A and follows the corresponding transition. If a terminal symbol c is at the top of the stack, $\mathcal{M}$ performs the comparison between input and stack symbol as described earlier.

By this construction, we can see that $\mathcal{M}$ finishes its computation with an empty stack and no input word symbols of w left to read whenever $S \Rightarrow^* w$, and so $\mathcal{M}$ accepts the input word w if w can be generated by the context-free grammar G. Therefore, $L(\mathcal{M}) = L(G)$ as desired. □

Visually, we can think of the pushdown automaton constructed in the proof of Lemma 19 in the following way, where the number of each transition corresponds to its type:



Note that we don't require a "bottom of stack" symbol $\perp$ for this pushdown automaton, since we're only using the stack to keep track of where we are in the grammar's derivation.

Example 20. Consider the following context-free grammar G, where $V = \{S, A\}$ and $\Sigma_G = \{\mathtt{0}, \mathtt{1}, \mathtt{\#}\}$:

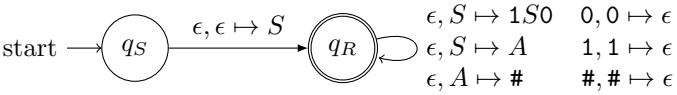
$$S \to \mathtt{0}S\mathtt{1} \mid A$$
$$A \to \mathtt{\#}$$

This grammar generates words of the form $\mathtt{0}^n\mathtt{\#}\mathtt{1}^n$, where $n \geq 0$.

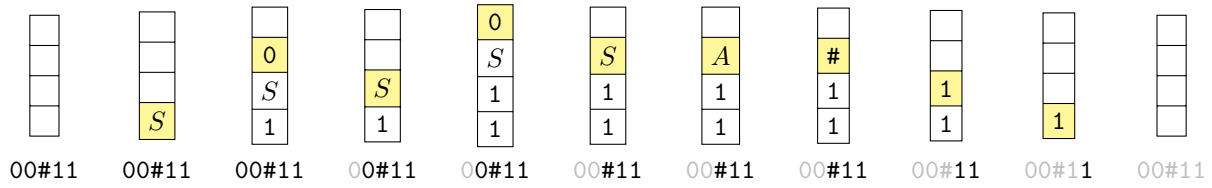
We convert the context-free grammar G to a pushdown automaton $\mathcal{M}$. Take $Q = \{q_S, q_R\}$, $\Sigma = \Sigma_G$, $\Gamma = V \cup \Sigma_G$, $q_0 = q_S$, and $F = \{q_R\}$. Finally, add the following transitions to δ:

- $\delta(q_S, \epsilon, \epsilon) = \{(q_R, S)\}$. This initial transition pushes the start nonterminal S to the stack.
- $\delta(q_R, \epsilon, S) = \{(q_R, \mathtt{1}S\mathtt{0}), (q_R, A)\}$. These nonterminal transitions account for the S rules.
- $\delta(q_R, \epsilon, A) = \{(q_R, \mathtt{\#})\}$. This nonterminal transition accounts for the A rule.
- $\delta(q_R, \mathtt{0}, \mathtt{0}) = \{(q_R, \epsilon)\}$, $\delta(q_R, \mathtt{1}, \mathtt{1}) = \{(q_R, \epsilon)\}$, and $\delta(q_R, \mathtt{\#}, \mathtt{\#}) = \{(q_R, \epsilon)\}$. These terminal transitions match the terminal symbols on the stack to the input word symbols.

This pushdown automaton $\mathcal{M}$ looks like the following:



As an illustration of the computation of $\mathcal{M}$, let's look at the stack as $\mathcal{M}$ reads an example input word $\mathtt{00\#11}$. We can see that G generates this word by the derivation $S \Rightarrow \mathtt{0}S\mathtt{1} \Rightarrow \mathtt{00}S\mathtt{11} \Rightarrow \mathtt{00}A\mathtt{11} \Rightarrow \mathtt{00\#11}$.

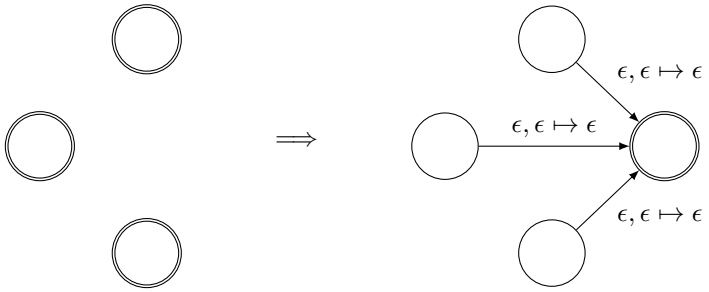


4.2 PDA ⇒ CFG

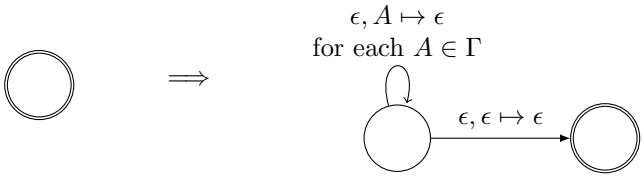
Now, we consider the other half of our main result. In order to convert a pushdown automaton to a context-free grammar, we must first ensure the pushdown automaton has certain properties: namely, the pushdown automaton must have a single accepting state, it must empty its stack before accepting, and each transition of the pushdown automaton must either push to or pop from the stack, but not both simultaneously. Let us refer to a pushdown automaton with these properties as a *simplified pushdown automaton*.

Fortunately, it's easy to convert from a pushdown automaton to a simplified pushdown automaton.

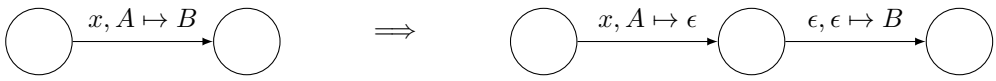
- To ensure the pushdown automaton has a single accepting state, we make each original accepting state non-accepting and add epsilon transitions from those states to a new single accepting state.



- To ensure the pushdown automaton empties its stack before accepting, we add a state immediately before the accepting state that removes all symbols from the stack.



- To ensure that each transition of the pushdown automaton either pushes to or pops from the stack, but not both, we split each transition that both pushes and pops into two separate transitions.



Additionally, if we have an epsilon transition that neither pushes nor pops, then we replace it with two "dummy" transitions that push and then immediately pop the same stack symbol.

With a simplified pushdown automaton, we can now perform the conversion to a context-free grammar.

Lemma 21. *Given a simplified pushdown automaton $\mathcal{M}$ recognizing a language $L(\mathcal{M})$, there exists a context-free grammar G such that $L(G) = L(\mathcal{M})$.*

Proof. Suppose we are given a simplified pushdown automaton $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$. We will construct a context-free grammar $G = (V, \Sigma_G, R, S)$ that generates the language recognized by $\mathcal{M}$.

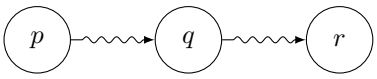
For each pair of states p and q in $\mathcal{M}$, our grammar will include a rule A_{pq} that simulates the computation of $\mathcal{M}$ starting in state p with some stack contents and ending in state q with the same stack contents. (Note that the stack may be manipulated during this computation; we just ensure that the contents of the stack are the same at the beginning and the end.)

We construct G in the following way:

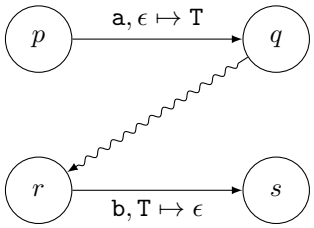
- The set of nonterminal symbols is $V = \{A_{pq} \mid p, q, \in Q\}$.
- The set of terminal symbols is $\Sigma_G = \Sigma$.
- The start nonterminal is $S = A_{q_0 q_{\text{accept}}}$ (i.e., the rule corresponding to the computation starting in state q_0 and ending in state q_{accept}).

- The set of rules R consists of three types of rules:
  1. For each state $q \in Q$, add the rule $A_{qq} \to \epsilon$ to R.

  2. For each triplet of states $p, q, r \in Q$, add the rule $A_{pr} \to A_{pq}A_{qr}$ to R.



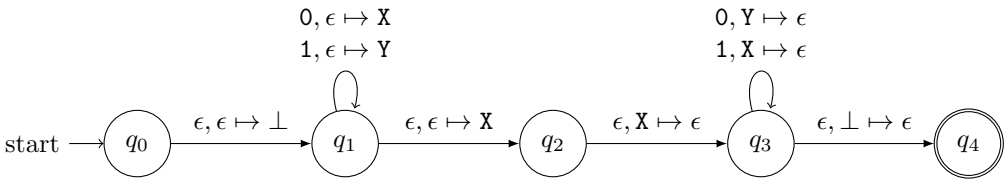
  3. For each quadruplet of states $p, q, r, s \in Q$, input symbols $\mathtt{a}, \mathtt{b} \in \Sigma \cup \{\epsilon\}$, and stack symbol $\mathtt{T} \in \Gamma$, if $(q, \mathtt{T}) \in \delta(p, \mathtt{a}, \epsilon)$ and $(s, \epsilon) \in \delta(r, \mathtt{b}, \mathtt{T})$, then add the rule $A_{ps} \to \mathtt{a}A_{qr}\mathtt{b}$ to R.



The first type of rule is a "dummy" rule that essentially corresponds to staying in the state q and adding nothing to the derivation. The second type of rule breaks down the overall computation into smaller components, taking into account intermediate states. Finally, the third type of rule adds terminal symbols to the derivation depending on the components of the overall computation.

With these rules, we can establish that the rule $A_{q_0 q_{\text{accept}}}$ generates a word w if and only if, starting in the state q_0 with an empty stack, the computation of $\mathcal{M}$ on w ends in the state q_{accept} also with an empty stack. Therefore, w is generated by the context-free grammar G if $\mathcal{M}$ accepts w, and $L(G) = L(\mathcal{M})$ as desired. □

Example 22. Consider the following simplified pushdown automaton $\mathcal{M}$, where $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ and $\Gamma = \{\mathtt{X}, \mathtt{Y}\}$:



This pushdown automaton recognizes words of the form $w \cdot \overline{w}^{\mathrm{R}}$, where $\overline{w}$ is w with 0s and 1s swapped.

We convert the pushdown automaton $\mathcal{M}$ to a context-free grammar G. Let $V = \{A_{00}, A_{01}, A_{02}, A_{03}, A_{04}, A_{11}, A_{12}, A_{13}, A_{14}, A_{22}, A_{23}, A_{24}, A_{33}, A_{34}, A_{44}\}$ and take $\Sigma_G = \Sigma$. We also take $S = A_{04}$, since q_0 is the initial state and q_4 is the accepting state of $\mathcal{M}$. Finally, we add the following rules to the rule set R:

- Type 1 rules: $A_{00} \to \epsilon$, $A_{11} \to \epsilon$, $A_{22} \to \epsilon$, $A_{33} \to \epsilon$, and $A_{44} \to \epsilon$.
- Type 2 rules: $A_{01} \to A_{00}A_{01} \mid A_{01}A_{11}$, $A_{02} \to A_{00}A_{02} \mid A_{01}A_{12} \mid A_{02}A_{22}$, $A_{03} \to A_{00}A_{03} \mid A_{01}A_{13} \mid A_{02}A_{23} \mid A_{03}A_{33}$, $A_{04} \to A_{00}A_{04} \mid A_{01}A_{14} \mid A_{02}A_{24} \mid A_{03}A_{34} \mid A_{04}A_{44}$, $A_{12} \to A_{11}A_{12} \mid A_{12}A_{22}$, $A_{13} \to A_{11}A_{13} \mid A_{12}A_{23} \mid A_{13}A_{33}$, $A_{14} \to A_{11}A_{14} \mid A_{12}A_{24} \mid A_{13}A_{34} \mid A_{14}A_{44}$, $A_{23} \to A_{22}A_{23} \mid A_{23}A_{33}$, $A_{24} \to A_{22}A_{24} \mid A_{23}A_{34} \mid A_{24}A_{44}$, and $A_{34} \to A_{33}A_{34} \mid A_{34}A_{44}$.
- Type 3 rules: $A_{13} \to \mathtt{0}A_{13}\mathtt{1} \mid \mathtt{1}A_{13}\mathtt{0} \mid \epsilon A_{22}\epsilon$ (or just A_{22}) and $A_{04} \to \epsilon A_{13}\epsilon$ (or just A_{13}).

As an illustration, let's see how G derives an example input word 001011. Beginning from the start nonterminal A_{04}, the derivation proceeds in the following way:

$$A_{04} \Rightarrow A_{13} \Rightarrow \mathtt{0}A_{13}\mathtt{1} \Rightarrow \mathtt{0}\,\mathtt{0}A_{13}\mathtt{1}\,\mathtt{1} \Rightarrow \mathtt{00}\,\mathtt{1}A_{13}\mathtt{0}\,\mathtt{11} \Rightarrow \mathtt{001}\,A_{22}\,\mathtt{011} \Rightarrow \mathtt{001}\,\epsilon\,\mathtt{011} = \mathtt{001011}.$$