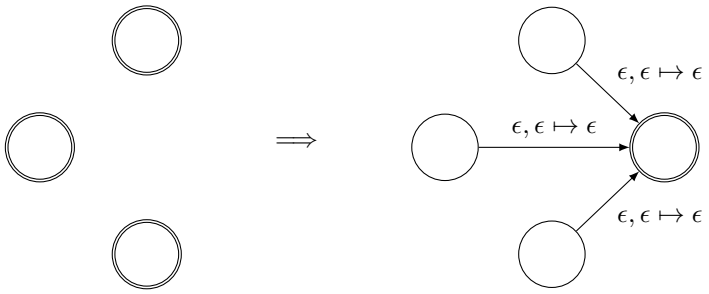
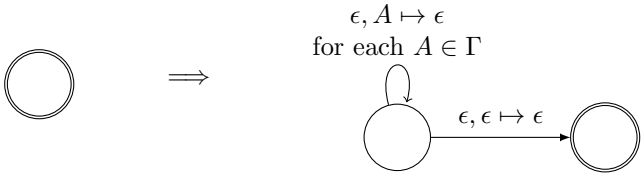
Now, we consider the other direction. In order to convert a pushdown automaton to a context-free grammar, we must first ensure the pushdown automaton has certain properties: namely, the pushdown automaton must have a single accepting state, it must empty its stack before accepting, and each transition 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 20. *Given a simplified pushdown automaton $\mathcal{M}$ recognizing a language $L(\mathcal{M})$, there exists a context-free grammar G such that $L(G) = L(\mathcal{M})$.*

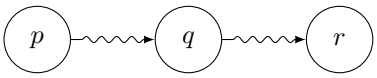
Proof. Suppose we are given a simplified pushdown automaton $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$. We will construct a context-free grammar $G = (V, \Sigma_G, R, S)$ that generates the language recognized by $\mathcal{M}$. For each pair of states p and q in $\mathcal{M}$,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

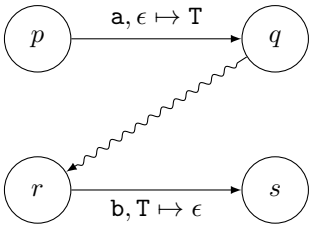
- The set of nonterminal symbols is $V = \{A_{pq} \mid p, q, \in Q\}$.
- The set of terminal symbols is $\Sigma_G = \Sigma$.

- 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, contains the following:
  - For each state $q \in Q$, add the rule $A_{qq} \to \epsilon$ to R.

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



  - 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. □

Since we know by Definition 11 that a language is context-free if there exists a context-free grammar generating the language, we can combine the previous two lemmas to get the main result of this section.

Theorem 21. *A language A is context-free if and only if there exists a pushdown automaton $\mathcal{M}$ such that $L(\mathcal{M}) = A$.*

Proof. ($\Rightarrow$): Follows from Lemma 19.

($\Leftarrow$): Follows from Lemma 20. □

Let's now consider some examples of converting from a context-free grammar to a pushdown automaton and vice versa.

Example 22. Consider the following context-free grammar G, where $V = \{S, A\}$ and $\Sigma_G = \{\mathtt{0}, \mathtt{1}, \mathtt{\#}\}$:

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

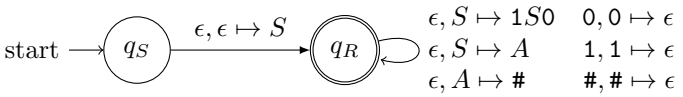
This grammar generates words of the form $\mathtt{0}^n\mathtt{\#}\mathtt{1}^n$, where $n \geq 0$.

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:

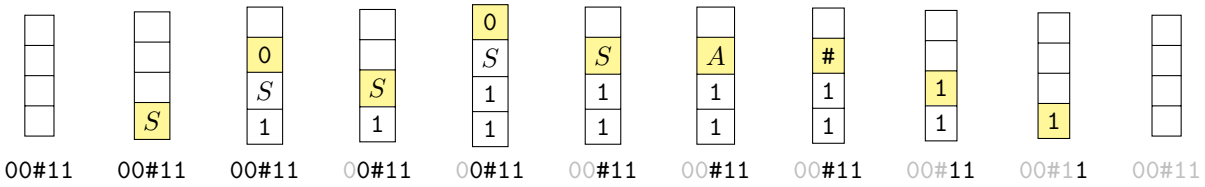
- $\delta(q_S, \epsilon, \epsilon) = \{(q_R, S)\}$. This initial transition pushes the start nonterminal S to the stack.

- $\delta(q_R, \epsilon, S) = \{(q_R, \texttt{1S0}), (q_R, A)\}$. These nonterminal transitions account for the S rules.
- $\delta(q_R, \epsilon, A) = \{(q_R, \texttt{\#})\}$. This nonterminal transition accounts for the A rule.
- $\delta(q_R, \texttt{0}, \texttt{0}) = \{(q_R, \epsilon)\}$, $\delta(q_R, \texttt{1}, \texttt{1}) = \{(q_R, \epsilon)\}$, and $\delta(q_R, \texttt{\#}, \texttt{\#}) = \{(q_R, \epsilon)\}$. These terminal transitions match the terminal symbols on the stack to the input word symbols.

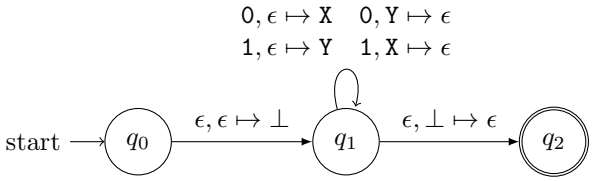
Diagrammatically, the pushdown automaton $\mathcal{M}$ looks like the following:



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



Example 23. Consider the following pushdown automaton $\mathcal{M}$, where $\Sigma = \{\texttt{0}, \texttt{1}\}$ and $\Gamma = \{\texttt{X}, \texttt{Y}\}$:



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

We convert the pushdown automaton $\mathcal{M}$ to a context-free grammar G. Let $V = \{A_{00}, A_{01}, A_{02}, A_{11}, A_{12}, A_{22}\}$ and take $\Sigma_G = \Sigma$. We also take $S = A_{02}$, since q_0 is the initial state and q_2 is the accepting state of $\mathcal{M}$. Finally, we add the following rules to the rule set R:

- $A_{00} \to \epsilon$, $A_{11} \to \epsilon$, and $A_{22} \to \epsilon$. These rules are of the first type.
- $A_{01} \to A_{01}A_{11}$, $A_{02} \to A_{01}A_{12}$, $A_{11} \to A_{11}A_{11}$, and $A_{12} \to A_{11}A_{12}$. These rules are of the second type.
- $A_{11} \to \texttt{0}A_{11}\texttt{1}$, $A_{11} \to \texttt{1}A_{11}\texttt{0}$, and $A_{02} \to A_{11}$. These rules are of the third type.

As an illustration, let's see how G derives an example input word 001110. Beginning from the start nonterminal A_{02}, the derivation proceeds in the following way:

$$A_{02} \Rightarrow A_{11} \Rightarrow A_{11}A_{11} \Rightarrow \texttt{0}A_{11}\texttt{1}\,A_{11} \Rightarrow \texttt{0}\,\texttt{0}A_{11}\texttt{1}\,\texttt{1}A_{11} \Rightarrow \texttt{00}\,\epsilon\,\texttt{11}A_{11} \Rightarrow \texttt{0011}\,\texttt{1}A_{11}\texttt{0} \Rightarrow \texttt{00111}\,\epsilon\,\texttt{0} = \texttt{001110}.$$

Note that, as a consequence of the equivalence between context-free grammars and pushdown automata, we get an important corollary relating the class of context-free languages to our familiar class of regular languages.

Corollary 24. *Every regular language is also a context-free language.*

Proof. Every regular language is recognized by some finite automaton. Since a finite automaton is a pushdown automaton that does not use the stack, every regular language is also accepted by some pushdown automaton. Therefore, every regular language is context-free. □

3 Proving a Language is Non-Context-Free

At the end of our discussion on regular languages, we saw that there exist certain languages that are nonregular, and we also saw that we can prove a language is nonregular by using the pumping lemma. One of the biggest obstacles we observed that results in a language being nonregular was, broadly speaking, having to count or otherwise keep track of symbols. Fortunately, by augmenting our machine model with a stack and creating a pushdown automaton, we were able to overcome this obstacle. Surely, this means that we can now recognize any language we want, right?

Well, not exactly. While the stack goes a long way in helping us to recognize more than just the class of regular languages, it isn't the magic solution we need in order to recognize *any* language. Consider, for example, the language

$$L_{\mathtt{a=b=c}} = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}.$$

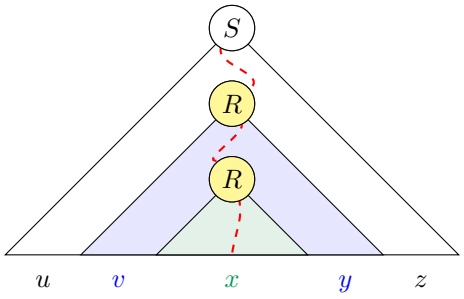
We know that a pushdown automaton can accept words of the form $\mathtt{a}^n\mathtt{b}^n$ by pushing one symbol to the stack for each a that is read, and then popping one symbol from the stack for each b that is read. When it comes to recognizing words of the form $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$, however, we run into a problem: after we read all of the bs in the word, our stack will be empty and we will therefore have forgotten the value of n by the time we have to count the cs! We also can't cheat our way around this problem by, for example, pushing two symbols to the stack for each a we read; if we try that, then reading either b or c would require us to pop the same symbol, and we can draw a conclusion that such an approach would result in the pushdown automaton accidentally accepting words where bs and cs are either out of order or having mismatched counts.

Thus, there do indeed exist languages that are not context-free, and so we require a technique to prove the non-context-freeness of a language. Fortunately, we're mostly familiar with such a technique already: the pumping lemma for regular languages is a special case of the more general pumping lemma for context-free languages.

3.1 The Pumping Lemma for Context-Free Languages

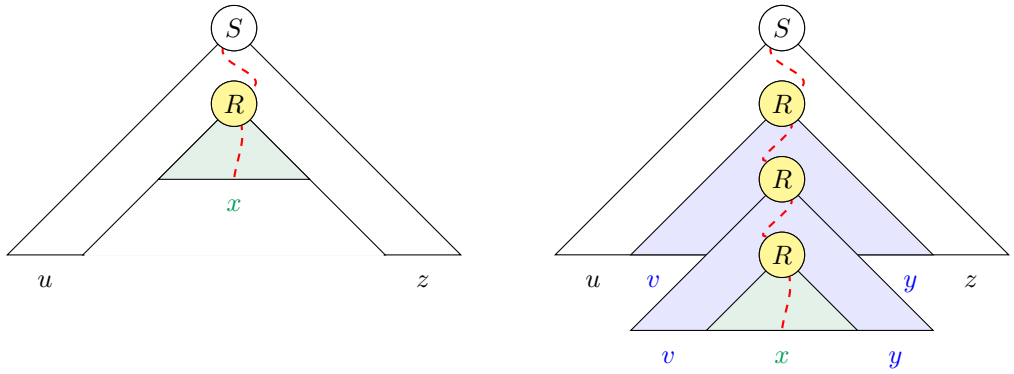
You might be wondering at this point what we mean by the pumping lemma for regular languages being a "special case". Since regular languages are, in a sense, simpler than context-free languages, our formulation of the pumping lemma for regular languages was accordingly simpler: pumping the middle portion of any sufficiently-long word in a regular language results in us obtaining another word that also belongs to the language. Pumping this middle portion of the word essentially corresponds to us traversing a loop somewhere in the finite automaton recognizing the language.

With context-free languages, however, we can't just pump one portion of the word. To understand why not, recall that we can represent the derivation of a word in a context-free language using a parse tree. If our word is sufficiently long,[8] then the parse tree will be rather deep. Then, since we have only a finite number of both rules and nonterminal symbols, the pigeonhole principle tells us that there must exist some path from the root of the parse tree to a leaf of the parse tree where some nonterminal symbol R appears more than once along that path.



[8] "Sufficiently long" in this context is measured in terms of both the maximum number of symbols on the right-hand side of any rule of the context-free grammar and the size of the set of nonterminal symbols.

Suppose, as we did in the illustration, that we decompose our word w into five parts, denoted $uvxyz$. If we then take the subtree rooted at the first occurrence of R, we can repeat this subtree zero or more times by appending the subtree to the other occurrences of R. In essence, we are pumping the segment of the subtree between both occurrences of R when we perform this repetition, and as a result, we are pumping the v and y portions of the word simultaneously.



With this idea in mind, the formal statement of the pumping lemma for context-free languages is quite similar to that of the pumping lemma for regular languages, modulo the appropriate changes.

Lemma 25 (Pumping lemma for context-free languages)**.** *For all context-free languages L, there exists $p \geq 1$ where, for all $w \in L$ with $|w| \geq p$, there exists a decomposition of w into five parts $w = uvxyz$ such that*

1. *$|vy| > 0$;*
2. *$|vxy| \leq p$; and*
3. *for all $i \geq 0$, $uv^ixy^iz \in L$.*

Additionally, just like before, we can write a proof that some language is non-context-free by simply following a common set of steps. As a consequence, non-context-freeness proofs will share a similar structure. To see such an example of a proof, let's revisit the language we introduced at the beginning of this section.

Example 26. Let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, and consider the language $L_{\mathrm{a=b=c}} = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$. We will use the pumping lemma to show that this language is non-context-free.

Assume by way of contradiction that the language is context-free, and let p denote the pumping constant given by the pumping lemma. We choose the word $w = \mathtt{a}^p\mathtt{b}^p\mathtt{c}^p$. Clearly, $w \in L_{\mathrm{a=b=c}}$ and $|w| \geq p$. Thus, there exists a decomposition $w = uvxyz$ satisfying the three conditions of the pumping lemma.

Observe that the first condition of the pumping lemma requires that *either* part v or part y is nonempty; potentially both could be nonempty. We consider two cases, depending on the contents of the parts v and y of the word w:

1. Both part v and part y contain some number of a single alphabet symbol; that is, v contains only `a`s, only `b`s, or only `c`s, and likewise for y. (Note that v and y do not need to contain the *same* alphabet symbol; for example, v could contain only `a`s and y could contain only `b`s.)

   In this case, pumping v and y once to obtain the word uv^2xy^2z results in the word containing unequal numbers of `a`s, `b`s, and `c`s. This violates the third condition of the pumping lemma.

2. Either part v or part y contains some number of multiple alphabet symbols; that is, either v or y contains both `a`s and `b`s, or both `b`s and `c`s.

   In this case, pumping v and y once to obtain the word uv^2xy^2z results in the word containing symbols out of order. This violates the third condition of the pumping lemma.

In all cases, one of the conditions of the pumping lemma is violated. As a consequence, the language cannot be context-free.

By a similar line of reasoning, we can show that the language $L = \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^n\mathtt{d}^m \mid m, n \geq 1\}$ is non-context-free, since there's no way for us to separate the counts of as/cs and bs/ds using only one stack.

Recall that, earlier, we used the pumping lemma for regular languages to show that the language $L_{\text{pal}} = \{ww^{\text{R}} \mid w \in \Sigma^*\}$ was nonregular. We can easily show that this language is context-free; on the other hand, here we will show that a very similar language is, in fact, non-context-free.

Example 27. Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, and consider the language $L_{\text{double}} = \{ww \mid w \in \Sigma^*\}$. We will use the pumping lemma to show that this language is non-context-free.

Assume by way of contradiction that the language is context-free, and let p denote the pumping constant given by the pumping lemma. We choose the word $s = \mathtt{a}^p\mathtt{b}^p\mathtt{a}^p\mathtt{b}^p$. Clearly, $s \in L_{\text{double}}$ and $|s| \geq p$. Thus, there exists a decomposition $s = uvxyz$ satisfying the three conditions of the pumping lemma.

Observe that the second condition of the pumping lemma requires that $|vxy| \leq p$. Here, we will consider two cases, depending on the contents of the middle portion vxy of the word s:

1. If vxy occurs entirely within the first half of s (that is, within the first occurrence of $\mathtt{a}^p\mathtt{b}^p$), then as a consequence of the fact that $|vxy| \leq p$, we must have one of the following subcases: vxy contains all as, vxy contains all bs, or vxy contains both as and bs, where all as occur before bs.

   In any of these three subcases, pumping v and y once to obtain the word uv^2xy^2z results in the first half of the word differing from the second half of the word. We can make an analogous argument if vxy occurs entirely in the second half of s. This violates the third condition of the pumping lemma.

2. If vxy straddles both halves of s, then v and y must contain different symbols as a consequence of the fact that $|vxy| \leq p$. Pumping v and y down to obtain the word $uv^0xy^0z = uxz$ results in the first half containing fewer bs than the second half, and the second half containing fewer as than the first half. This violates the third condition of the pumping lemma.

In all cases, one of the conditions of the pumping lemma is violated. As a consequence, the language cannot be context-free.

Finally, let's revisit our diagram by adding to it the class of context-free languages. Of course, knowing now that there exist languages that aren't context-free, our diagram must not be complete just yet. We still have some language classes to add, and we will be taking care of those classes in the lectures to come.