PROOF We now give the formal details of the construction of the pushdown automaton $P = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, F)$. To make the construction clearer, we use shorthand notation for the transition function. This notation provides a way to write an entire string on the stack in one step of the machine. We can simulate this action by introducing additional states to write the string one symbol at a time, as implemented in the following formal construction.

Let q and r be states of the PDA and let a be in Σ_ε and s be in Γ_ε. Say that we want the PDA to go from q to r when it reads a and pops s. Furthermore, we want it to push the entire string $u = u_1 \cdots u_l$ on the stack at the same time. We can implement this action by introducing new states $q_1, \ldots, q_{l-1}$ and setting the transition function as follows:

$$\begin{aligned}
&\delta(q, a, s) \text{ to contain } (q_1, u_l),\\
&\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, u_{l-1})\},\\
&\delta(q_2, \varepsilon, \varepsilon) = \{(q_3, u_{l-2})\},\\
&\qquad\vdots\\
&\delta(q_{l-1}, \varepsilon, \varepsilon) = \{(r, u_1)\}.
\end{aligned}$$

We use the notation $(r, u) \in \delta(q, a, s)$ to mean that when q is the state of the automaton, a is the next input symbol, and s is the symbol on the top of the stack, the PDA may read the a and pop the s, then push the string u onto the stack and go on to the state r. The following figure shows this implementation.

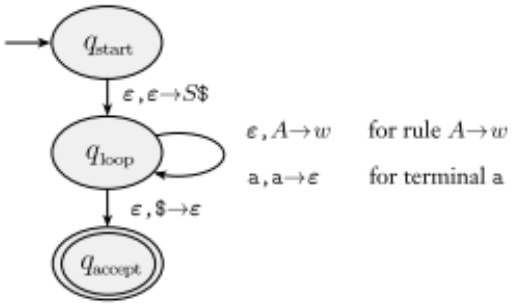
The states of P are $Q = \{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup E$, where E is the set of states we need for implementing the shorthand just described. The start state is q_{start}. The only accept state is q_{accept}.

The transition function is defined as follows. We begin by initializing the stack to contain the symbols \$ and S, implementing step 1 in the informal description: $\delta(q_{\text{start}}, \varepsilon, \varepsilon) = \{(q_{\text{loop}}, S\$)\}$. Then we put in transitions for the main loop of step 2.

First, we handle case (a) wherein the top of the stack contains a variable. Let $\delta(q_{\text{loop}}, \varepsilon, A) = \{(q_{\text{loop}}, w) | \text{ where } A \to w \text{ is a rule in } R\}$.

Second, we handle case (b) wherein the top of the stack contains a terminal. Let $\delta(q_{\text{loop}}, a, a) = \{(q_{\text{loop}}, \varepsilon)\}$.

Finally, we handle case (c) wherein the empty stack marker \$ is on the top of the stack. Let $\delta(q_{\text{loop}}, \varepsilon, \$) = \{(q_{\text{accept}}, \varepsilon)\}$.



PROOF Let G be a CFG for CFL A. Let b be the maximum number of symbols in the right-hand side of a rule (assume at least 2). In any parse tree using this grammar, we know that a node can have no more than b children. In other words, at most b leaves are 1 step from the start variable; at most b^2 leaves are within 2 steps of the start variable; and at most b^h leaves are within h steps of the start variable. So, if the height of the parse tree is at most h, the length of the string generated is at most b^h. Conversely, if a generated string is at least $b^h + 1$ long, each of its parse trees must be at least $h + 1$ high.

Say $|V|$ is the number of variables in G. We set p, the pumping length, to be $b^{|V|+1}$. Now if s is a string in A and its length is p or more, its parse tree must be at least $|V| + 1$ high, because $b^{|V|+1} \geq b^{|V|} + 1$.

To see how to pump any such string s, let τ be one of its parse trees. If s has several parse trees, choose τ to be a parse tree that has the smallest number of nodes. We know that τ must be at least $|V| + 1$ high, so its longest path from the root to a leaf has length at least $|V| + 1$. That path has at least $|V| + 2$ nodes; one at a terminal, the others at variables. Hence that path has at least $|V| + 1$ variables. With G having only $|V|$ variables, some variable R appears more than once on that path. For convenience later, we select R to be a variable that repeats among the lowest $|V| + 1$ variables on this path.

We divide s into $uvxyz$ according to Figure 2.35. Each occurrence of R has a subtree under it, generating a part of the string s. The upper occurrence of R has a larger subtree and generates vxy, whereas the lower occurrence generates just x with a smaller subtree. Both of these subtrees are generated by the same variable, so we may substitute one for the other and still obtain a valid parse tree. Replacing the smaller by the larger repeatedly gives parse trees for the strings uv^ixy^iz at each $i > 1$. Replacing the larger by the smaller generates the string uxz. That establishes condition 1 of the lemma. We now turn to conditions 2 and 3.

To get condition 2, we must be sure that v and y are not both ε. If they were, the parse tree obtained by substituting the smaller subtree for the larger would have fewer nodes than τ does and would still generate s. This result isn't possible because we had already chosen τ to be a parse tree for s with the smallest number of nodes. That is the reason for selecting τ in this way.

In order to get condition 3, we need to be sure that vxy has length at most p. In the parse tree for s the upper occurrence of R generates vxy. We chose R so that both occurrences fall within the bottom $|V| + 1$ variables on the path, and we chose the longest path in the parse tree, so the subtree where R generates vxy is at most $|V| + 1$ high. A tree of this height can generate a string of length at most $b^{|V|+1} = p$.