

Pushdown Automata

Definition 1 A pushdown automaton (over Σ) is a quintuple $M = (Q, V, s, F, T)$, where:

- Q is a finite set (the states)
- V is an alphabet (the stack symbols)
- $s \in Q$ is the start state
- $F \subseteq Q$ is the set of final states
- $T \subseteq Q \times (\Sigma \cup \{\Lambda\}) \times V^* \times V^* \times Q$ is the transition relation. If $(q_1, a, v, w, q_2) \in T$, then we write $q_1 \xrightarrow{a, v/w} q_2$.

Intuitively, the presence of a transition (q_1, a, v, w, q_2) means that if the PDA is in state q_1 and the next input symbol is a and the top few symbols on the stack are v , then the PDA can move to state q_2 by popping the v from the stack and replacing it with w .

Definition 2 A configuration of a PDA $M = (Q, V, s, F, T)$ is an element of $Q \times \Sigma^* \times V^*$ — that is, a triple of the form (q, w, v) where q is a state, w is a string over Σ and v is a string of stack symbols.

Definition 3 Let $M = (Q, V, s, F, T)$ be a PDA. Then the one-step execution relation of M , written \mapsto_M , is defined by:

$$(q_1, aw, \alpha\gamma) \mapsto_M (q_2, w, \beta\gamma) \text{ if } q_1 \xrightarrow{a, \alpha/\beta} q_2 \text{ is in } T$$

The multi-step execution relation \mapsto_M^* is defined as follows. If C_0 and C_n are configurations of M , then $C_0 \mapsto_M^* C_n$ if there is a sequence of configurations C_1, \dots, C_{n-1} such that

$$C_0 \mapsto_M C_1 \mapsto_M \dots \mapsto_M C_{n-1} \mapsto_M C_n.$$

In other words, \mapsto_M^* is the reflexive, transitive closure of \mapsto_M .

We will often omit the subscript M , writing simply \mapsto and \mapsto^* , if the PDA in question is clear from context.

In proving theorems about the execution of pushdown automata, the following lemma will be very convenient. We state it without proof.

Lemma 1 Let $M = (Q, V, s, F, T)$ be a PDA. If $(q, w, \gamma) \mapsto_M^* (q', w', \gamma')$, then for any $v \in \Sigma^*$ and any $\delta \in V^*$, $(q, wv, \gamma\delta) \mapsto_M^* (q', w'v, \gamma'\delta)$.

Proof: Left as an exercise.

Acceptance of Context-Free Languages

The class of languages accepted by pushdown automata is exactly the context free language. To prove this, we will show how to convert a context-free grammar into an equivalent PDA, and how to convert a PDA into an equivalent CFG.

We'll do the grammar-to-automaton direction first. Here is the construction:

Constructing a PDA from a CFG

Let $G = (N, S, R)$ be a context-free grammar. Then the equivalent PDA is $M = (Q, V, s, F, T)$, where $Q = \{s, t\}$, $V = N \cup \Sigma$, $F = \{t\}$ and T contains the following transitions:

- $s \xrightarrow{\Lambda, \Lambda/S} t$
- $t \xrightarrow{\Lambda, A/w} t$ for each rule $A \rightarrow w \in R$.
- $t \xrightarrow{a, a/\Lambda} t$ for each terminal symbol $a \in \Sigma$.

The correctness of this construction will follow from the following Lemma.

Lemma 2 *Let G be a CFG and M be constructed from G as above. If $w \in \Sigma^*$ and $v \in \{\Lambda\} \cup N(N \cup \Sigma)^*$ (that is, w is the longest all-terminal prefix of wv), then $S \xRightarrow{L^*} wv$ iff $(t, w, S) \mapsto^* (t, \Lambda, v)$. (\xRightarrow{L} means a leftmost derivation.)*

Proof: We will prove each direction separately.

(\Rightarrow): Suppose $S \xRightarrow{L^*} wv$. We'll show that $(t, w, S) \mapsto^* (t, \Lambda, v)$ by induction on the length of the derivation in the grammar.

Base case: The derivation takes zero steps, so $S = wv$. Since w consists only of terminals and S is a nonterminal, this means $w = \Lambda$ and $v = S$; hence $(t, w, S) = (t, \Lambda, v)$ and so the required PDA execution is accomplished with zero steps.

Inductive case: Assume the lemma holds for leftmost derivations of length n , and suppose the derivation $S \xRightarrow{L^*} wv$ takes $n + 1$ steps, *i.e.*:

$$S \xRightarrow{L} u_1 \xRightarrow{L} \dots \xRightarrow{L} u_n \xRightarrow{L} wv.$$

Since each step in this derivation expands the leftmost nonterminal, let A be the leftmost nonterminal in u_n . Then for some strings x and z (with $x \in \Sigma^*$) we must have $u_n = xAz$ and $wv = xyz$ where $A \rightarrow y$ is a rule of the grammar. Also, note that since w is the longest all-terminal prefix of xyz and x is all terminals, w must contain all of x and possibly some of y . So we must have $y = y_1y_2$, where $y_1 \in \Sigma^*$ and $w = xy_1$ and $v = y_2z$. Now, in the derivation above we have $S \xRightarrow{L^*} xAz$ in n steps, so the induction hypothesis gives $(t, x, S) \mapsto^* (t, \Lambda, Az)$. Since a PDA can only look one character ahead at a time, we also have

$$(t, xy_1, S) \mapsto^* (t, y_1, Az) \mapsto (t, y_1, yz) = (t, y_1, y_1y_2z) \mapsto^* (t, \Lambda, y_2z)$$

But $y_2z = v$, so this is what we needed to show.

(\Leftarrow): Suppose $(t, w, S) \mapsto^* (t, \Lambda, v)$. We'll show that $S \Rightarrow^* wv$ by induction on the number of steps in the execution.

Base case: The execution takes zero steps, so $w = \Lambda$ and $S = v$. Then clearly $S \Rightarrow^* wv$.

Inductive case: Suppose the lemma holds for all executions of length less than n , and suppose $(t, w, S) \mapsto^* (t, \Lambda, v)$ in n steps, *i.e.*,

$$(t, w, S) \mapsto C_1 \mapsto \dots \mapsto C_{n-1} \mapsto (t, \Lambda, v)$$

Observe that in this sequence must be at least one step using a transition that expands a nonterminal on the stack. (Since there are no terminals on the stack, that is the only kind of transition possible for the first step.) Consider the portion of the execution starting with the *last* expansion step:

$$(t, w, S) \mapsto^* (t, w', Av') \mapsto (t, w', xv') \mapsto^* (t, \Lambda, v)$$

The last group of steps are all of the kind that match an input character against a stack symbol; thus we conclude that $xv' = w'v$. Now, the input w' that remains before A is expanded must be a suffix of w , so it must be that $w = w_0w'$ and $(t, w_0, S) \mapsto^* (t, \Lambda, Av')$ in fewer than n steps. By the induction hypothesis, then,

$$S \Rightarrow^* w_0Av' \Rightarrow w_0xv' = w_0w'v = wv$$

which is what we needed to show.

Q.E.D.

It follows immediately that the pushdown automaton constructed for a grammar G accepts exactly $\mathcal{L}(G)$.

Lemma 3 *If L is context-free, then there is a PDA M such that $L = \mathcal{L}(M)$.*

Proof: Let G be a context-free grammar for L , and let M be constructed from G as above. We will show that $\mathcal{L}(M) = \mathcal{L}(G)$.

(\subseteq): Suppose $(s, w, \Lambda) \mapsto^* (t, \Lambda, \Lambda)$ (as it must if $w \in \mathcal{L}(M)$, since t is the only final state). The first step of this execution must use the transition that moves from s to t pushing S onto the stack; thus, $(s, w, \Lambda) \mapsto (t, w, S) \mapsto^* (t, \Lambda, \Lambda)$. By Lemma 2, $S \Rightarrow^* w$ and so $w \in \mathcal{L}(G)$.

(\supseteq): Suppose $S \Rightarrow^* w$; then $S \xRightarrow{L} w$. By Lemma 2, $(t, w, S) \mapsto^* (t, \Lambda, \Lambda)$; hence $(s, w, \Lambda) \mapsto (t, w, S) \mapsto^* (t, \Lambda, \Lambda)$ and so $w \in \mathcal{L}(M)$.

Q.E.D.

Now, we show how to construct a context-free grammar equivalent to a given pushdown automaton. To simplify the construction, we will limit ourselves for the moment to pushdown automata that only use certain limited types of transitions; we call such automata “simple”, and define them as follows.

Definition 4 *A pushdown automaton $M = (Q, V, s, F, T)$ is simple if every transition $p \xrightarrow{a, \alpha, \beta} q \in T$ satisfies the following:*

1. $q \neq s$;
2. if $p = s$, then $a = \alpha = \Lambda$ and $|\beta| = 1$; and
3. If $p \neq s$, then $|\alpha| = 1$ and $|\beta| \leq 2$.

In other words, a simple pushdown automaton must begin execution by pushing a single symbol onto the stack, and never reenter its start state; furthermore, at each subsequent step it removes one symbol from the stack and replaces it with at most two symbols. These limitations make it easier to construct a grammar to recognize the language accepted by the PDA:

Constructing a CFG from a simple PDA

Let $M = (Q, V, s, F, T)$ be a simple PDA. Then the equivalent grammar is $G = (N, S, R)$ where

- $N = \{S\} \cup \{B_{pq} \mid B \in V \cup \{\Lambda\}, p, q \in Q\}$.
- R contains the following rules:
 - $S \rightarrow A_{qf}$ for each transition of the form $s \xrightarrow{\Lambda, \Lambda/A} q \in T$ and each $f \in F$.
 - $A_{pr} \rightarrow aB_{qr}$ for every transition $p \xrightarrow{a, A/B} q \in T$ and every $q \in Q$.
 - $A_{pt} \rightarrow aB_{qr}B'_{rt}$ for every transition $p \xrightarrow{a, A/BB'} q$ and every pair of states q, q' .
 - $\Lambda_{qq} \rightarrow \Lambda$ for each state q .

We now prove the main lemma for the correctness of this construction.

Lemma 4 *Let $M = (Q, V, s, F, T)$ be a simple PDA and $G = (N, S, R)$ be constructed from M as above. Then for any $A \in N \cup \{\Lambda\}$ and $p \in Q \setminus \{s\}$, $(p, w, A) \mapsto_M^* (q, \Lambda, \Lambda)$ iff $A_{pq} \Rightarrow^* w$.*

Proof: We will prove each direction separately.

(\Rightarrow): Suppose $(p, w, A) \mapsto^* (q, \Lambda, \Lambda)$. We will show $A_{pq} \Rightarrow^* w$ by induction on the number of steps in the computation.

Base case: The computation takes zero steps, *i.e.*, $p = q$, $w = \Lambda$ and $A = \Lambda$. But by construction, the grammar G contains the rule $\Lambda_{pp} \rightarrow \Lambda$; hence $A_{pq} = \Lambda_{pp} \Rightarrow^* \Lambda = w$.

Induction step: Suppose this direction of the lemma holds for computations of length less than n , and that $(p, w, A) \mapsto^* (q, \Lambda, \Lambda)$ in n steps. There are two cases, depending on the type of transition used in the first step of the computation (we note that by assumption, $p \neq s$).

Case 1: The first step is of the form $(p, aw', A) \mapsto (p', w', A')$, where $w = aw'$, $a \in \Sigma \cup \{\Lambda\}$, $A' \in V \cup \{\Lambda\}$ and the transition $p \xrightarrow{a, A/A'} p'$ is in T . The remainder of the computation therefore has $n - 1$ steps and the form $(p', w', A') \mapsto^* (q, \Lambda, \Lambda)$; by the induction hypothesis, $A'_{p'q} \Rightarrow^* w'$. By construction, the grammar G contains the rule $A_{pq} \rightarrow aA'_{p'q}$, so we have a derivation $A_{pq} \Rightarrow aA'_{p'q} \Rightarrow^* aw' = w$.

Case 2: The first step is of the form $(p, aw', A) \mapsto (p', w', A_1A_2)$ where $w = aw'$, $a \in \Sigma \cup \{\Lambda\}$, $A_1, A_2 \in V$ and the transition $p \xrightarrow{a, A/A_1A_2} p'$ is in T . Because the PDA M is simple, we can separate the remainder of the computation into two parts by finding p'', w_1, w_2 such that $w' = w_1w_2$ and:

$$(p', w_1, A_1) \mapsto^* (p'', \Lambda, \Lambda) \quad \text{and} \quad (p'', w_2, A_2) \mapsto^* (q, \Lambda, \Lambda) \quad (*)$$

where the total number of steps in both of these computations is $n - 1$. (This is a nontrivial consequence of simplicity whose proof we are eliding right now.) By construction, the grammar must have the rule $A_{pq} \rightarrow aA_{1p''}A_{2p''q}$; using the induction hypothesis, we can form a derivation:

$$A_{pq} \Rightarrow a\underline{A_{1p''}}\underline{A_{2p''q}} \Rightarrow^* aw_1\underline{A_{2p''q}} \Rightarrow^* aw_1w_2 = aw' = w$$

In either case, we have found a derivation of $A_{pq} \Rightarrow^* w$, which is what we needed.

(\Leftarrow): Suppose $A_{pq} \Rightarrow^* w$. We will show $(p, w, A) \mapsto^* (q, \Lambda, \Lambda)$ by induction on the number of steps in the derivation. Note that the derivation must have at least one step, since A_{pq} is a nonterminal but $w \in \Sigma^*$.

Base case: The derivation has one step. The only rules with Λ on the right-hand side are those of the form $\Lambda_{qq} \rightarrow \Lambda$. In this case we have $A = \Lambda$, $w = \Lambda$ and $p = q$; hence trivially $(p, w, A) = (q, \Lambda, \Lambda) \mapsto^* (q, \Lambda, \Lambda)$.

Induction step: The derivation has n steps where $n > 1$. Suppose this direction of the lemma holds for derivations with fewer than n steps. There are two cases to consider:

Case 1: The rule used in the first step is of the form $A_{pq} \rightarrow aB_{p'q}$; this means that $w = aw'$ and there is a derivation of $B_{p'q} \Rightarrow^* w'$ in $n - 1$ steps. By the induction hypothesis, $(p', w', B) \mapsto^* (q, \Lambda, \Lambda)$. Moreover, by the construction of the grammar the PDA M must have the transition $p \xrightarrow{a, A/B} p'$; using this transition we get

$$(p, w, A) = (p, aw', A) \mapsto (p', w', B) \mapsto^* (q, \Lambda, \Lambda)$$

Case 2: The rule used in the first step is of the form $A_{pq} \rightarrow aB_{rt}C_{tq}$, $w = aw'$ and there is a derivation of $B_{rt}C_{tq} \Rightarrow^* w'$ with $n - 1$ steps. It follows that there must be some way to express w' as w_1w_2 so that there are derivations of $B_{rt} \Rightarrow^* w_1$ and $C_{tq} \Rightarrow^* w_2$ with fewer than $n - 1$ steps each. By the induction hypothesis, $(r, w_1, B) \mapsto^* (t, \Lambda, \Lambda)$ and $(t, w_2, C) \mapsto^* (q, \Lambda, \Lambda)$; by Lemma 1 this first result implies $(r, w_1w_2, BC) \mapsto^* (t, w_2, C)$. Moreover, by the construction of the grammar, M must have the transition $p \xrightarrow{a, A/BC} r$; using this, we get:

$$(p, w, A) = (p, aw', A) \mapsto (r, w', BC) = (r, w_1w_2, BC) \mapsto^* (t, w_2, C) \mapsto^* (q, \Lambda, \Lambda)$$

In either case, we have constructed an execution $(p, w, A) \mapsto^* (q, \Lambda, \Lambda)$ in M , which is what we needed.

Since we have shown both directions of the equivalence, our proof is complete.

Q.E.D.

As before, the correctness result follows easily from this main lemma.

Lemma 5 *Let M be a simple pushdown automaton. Then $\mathcal{L}(M)$ is context-free.*

Proof: Construct the grammar $G = (N, S, R)$ from M as described above. We claim that $\mathcal{L}(G) = \mathcal{L}(M)$, which means $\mathcal{L}(M)$ is context-free. We'll show inclusion in both directions.

(\subseteq): Suppose $w \in \mathcal{L}(G)$, i.e. there is a derivation $S \Rightarrow^* w$ in G . The first rule used in the derivation must be one of the form $S \rightarrow A_{qf}$ for some state q and final state f ; the remaining steps form a derivation of $A_{qf} \Rightarrow^* w$. By Lemma 4, $(q, w, A) \mapsto^* (f, \Lambda, \Lambda)$. By construction, the PDA M has the transition $s \xrightarrow{\Lambda, \Lambda/A} q$; hence

$$(s, w, \Lambda) \mapsto (q, w, A) \mapsto^* (f, \Lambda, \Lambda)$$

where f is final. By definition, then, $w \in \mathcal{L}(M)$.

(\supseteq): Suppose $w \in \mathcal{L}(M)$; this means for some final state f , $(s, w, \Lambda) \mapsto^* (f, \Lambda, \Lambda)$. Because M is simple, the first step in this execution must be $(s, w, \Lambda) \mapsto (q, w, A)$ for some state q and stack symbol

A and the rest of the execution gives $(q, w, A) \mapsto^* (f, \Lambda, \Lambda)$. By Lemma 4, $A_{qf} \Rightarrow^* w$; moreover, by construction the grammar contains the rule $S \rightarrow A_{qf}$; thus $S \Rightarrow A_{qf} \Rightarrow^* w$.

Q.E.D.

In order to show that the language accepted by **any** pushdown automaton is context-free, we must show that for any PDA there is a **simple** PDA that accepts the same language. Constructing such a PDA is not difficult, but we will not go into the formal details. The basic steps of the construction are:

1. Create a new start state s' and a new final state f' (final states of the original PDA become nonfinal). Choose a new stack symbol the PDA does not already use (say, $\#$) and add the transitions $s' \xrightarrow{\Lambda, \Lambda / \#} s$ and $f \xrightarrow{\Lambda, \# / \Lambda} f'$ for each final state f of the original PDA.

2. Remove each transition of the form $p \xrightarrow{a, \alpha / \beta} q$ where $|\beta| > 1$, and create new intermediate states connected by transitions that push the characters of β one at a time. For example, if $\beta = b_1 \cdots b_n$:

$$p \xrightarrow{\alpha, \Lambda / b_1} r_1 \xrightarrow{\Lambda, \Lambda / b_2} \cdots \xrightarrow{\Lambda, \Lambda / b_{n-1}} r_{n-1} \xrightarrow{\Lambda, \Lambda / b_n} q$$

3. Remove each transition $p \xrightarrow{a, \alpha / \beta} q$ where $p \neq s'$ and $|\alpha| > 1$, and create new intermediate states connected by transitions that pop the characters of α one at a time.
4. Replace each transition $p \xrightarrow{a, \Lambda / \beta} q$ with transitions $p \xrightarrow{a, b / \beta b} q$ for every stack symbol b , including $\#$.

The simple pda created by this construction operates by first pushing the “bottom-of-stack marker” $\#$ onto its stack and entering the start state of the original PDA. It then simulates the original PDA, except that it must push and pop stack symbols one at a time. If the original PDA might have reached a final state with an empty stack, the simple PDA can reach that same state with only $\#$ on the stack; it can then pop the $\#$, leaving the stack empty, and move to the new final state.