# Pushdown Automata

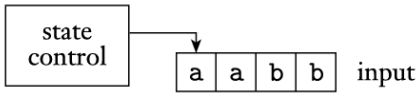
Alyssa Lytle

Fall 2025

For this lesson, we're going to go back to models of computation.

DFAs/NFAs are computational models that recognize regular languages. Similarly *pushdown automata* are computational models that recognize context-free languages.

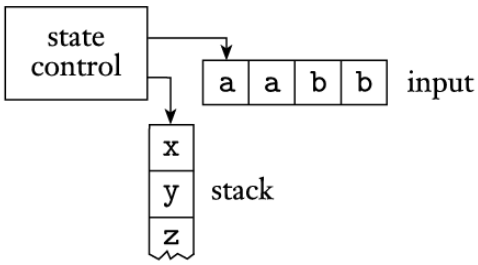
First, let's think about DFAs/NFAs a little differently using this representation:




**A Finite Automaton**[Sip96]

This figure is a schematic representation of a finite automaton. The control represents the states and transition function, the tape contains the input string, and the arrow represents the input head, pointing at the next input symbol to be read. [Sip96]

A pushdown automaton is similar in its design, but it includes a *stack* where the automaton can write symbols to read back later. In the nature of a stack, every time a new symbol is written to the stack, the other symbols are *pushed down*.




**A Pushdown Automaton**[Sip96]

Recall that adding an element to a stack is called *pushing* an element and removing one is called *popping*, as an element is "popped" of the top.

You can see how this design would be useful, because the stack allows us to track things. You can recall that this was a limitation for finite automata, and was the reason they couldn't be used to represents sets such as $\{a^n b^n | n \geq 0\}$.

**Example: Using a PDA to Recognize $a^n b^n$**

Abstractly, we can think of a strategy to use the stack to "track" that we have the same amount of $a$s and $b$s.

Every time an $a$ is encountered, push it onto the stack. Once a $b$ is encountered, pop an $a$ off the stack for every $b$ input. At the end of the input string, the stack should be empty.

Now we can formally define a PDA:

**Definition 1: Pushdown Automaton**

Let $\Sigma \cup \{\epsilon\}$ be denoted as $\Sigma_\epsilon$ and $\Gamma \cup \{\epsilon\}$ be denoted as $\Gamma_\epsilon$

A *pushdown automaton* is a 6-tuple $(Q, \Sigma, \Gamma, \delta, s, F)$ where

- $Q$ is the set of states
- $\Sigma$ is the the input alphabet
- $\Gamma$ is the stack alphabet
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$
- $s \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

Where $\mathcal{P}(Q \times \Gamma_\epsilon)$ is the powerset of $\mathcal{P}(Q \times \Gamma_\epsilon)$ (aka all possible subsets of $Q \times \Gamma_\epsilon$).

**An Aside: Epsilon Transitions** So far in this class we have avoided using these because they are not necessary, but in a nondeterministic automaton, "epsilon" transitions ($\epsilon$-transitions) can be useful in simplifying representation of a diagram. Essentially, they give us transitions over *no* input (aka the empty string $\epsilon$, which allows for more possibilities. In this example, you'll see that they are helpful in handling the beginning and end of an input, which is where I intend to use them in this course.

*Exercise: Why are epsilon transitions not necessary?*

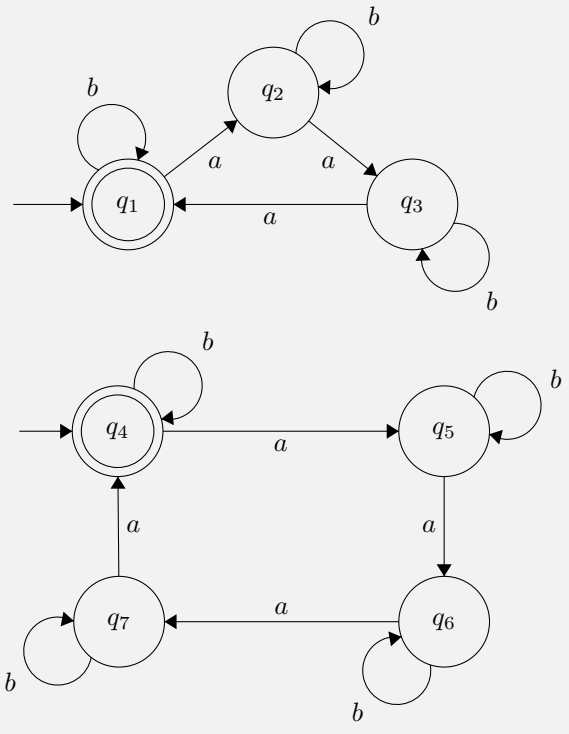
**Recall**

Recall this problem:

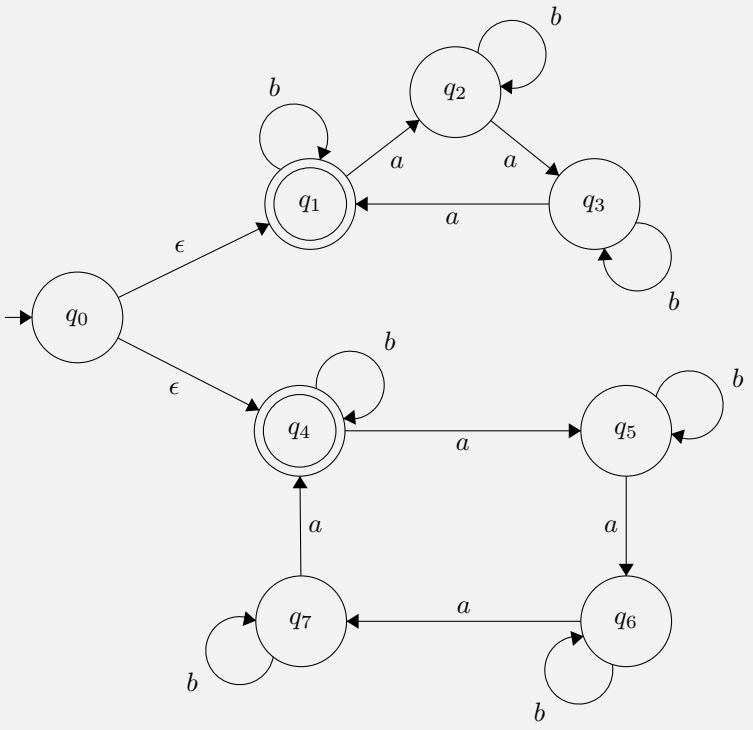
Design an NFA for the following languages over the alphabet $\Sigma = \{a, b\}$.

$L = \{w | w$ has $3m$ or $4m$ $a's$ for some $m \in \mathbb{N}\}$.

The solution can be written as follows:



However, with epsilon transitions, you can reduce the number of start states:




**Example: Using a PDA to Recognize $a^n b^n$ (continued)**

Now, we will demonstrate how to formally define the PDA that recognizes $\{a^n b^n | n \geq 0\}$

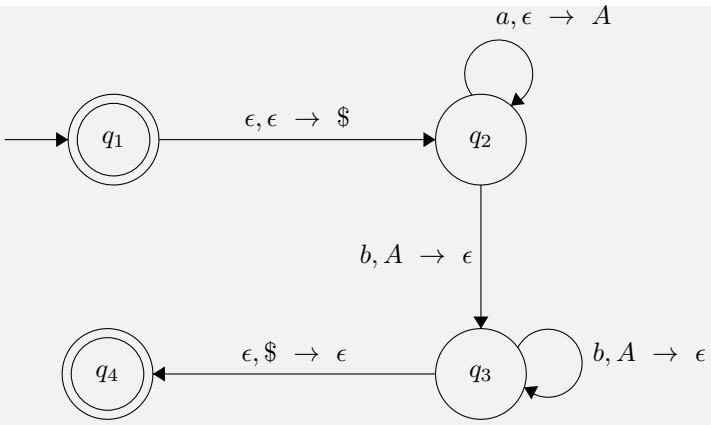
Define our PDA as $(Q, \Sigma, \Gamma, \delta, s, F)$ where:

- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{A, \$\}$
- $s = q_1$
- $F = \{q_1, q_4\}$

And our transition table is defined in the following way:

| Input: | $a$ | | | $b$ | | | $\epsilon$ | | |
|---|---|---|---|---|---|---|---|---|---|
| Stack: | A | \$ | $\epsilon$ | A | \$ | $\epsilon$ | A | \$ | $\epsilon$ |
| $q_1$ | | | | | | | | | $\{(q_2, \$)\}$ |
| $q_2$ | | | $\{(q_2, A)\}$ | $\{(q_3, \epsilon)\}$ | | | | | |
| $q_3$ | | | | $\{(q_3, \epsilon)\}$ | | | | $\{(q_4, \epsilon)\}$ | |
| $q_4$ | | | | | | | | | |

We also can draw a state diagram for it in the following way:




Basically, we push a \$ to the stack at the beginning so that when we reach that \$ again, we know we've reached the last element left in the stack.

**Why We use $\epsilon$ transitions + Testing for End Of Input** The way we've discussed acceptance of an input hasn't required knowing where the *end* of an input is. For this example, we need to know both when the stack is empty (because that tells us we've seen the same amount of $a$s and $b$s) and when the input is finished (aka we've transitioned over every input character).

To know when the stack is empty, we place that \$ in the stack at the beginning. Since a stack is "last in first out", we know that when we see the \$ again, we have removed every other item from the stack.

We use an $\epsilon$-transition from our start state $q_1$ to $q_2$ to allow us to initalize that \$ element in the stack. Another common convention is to *not* use $\epsilon$-transitions but instead to just add a "starting stack symbol" to the tuple definition, making it a 7-tuple. For this class, let's stick the the former convention of always starting with an empty stack.

In terms of knowing when we reach the end of the input, we essentially use an $\epsilon$-transition to an accept state to denote we've reached the end of the input tape. Another common convention you may see is adding \$ to $\Sigma$ and using that to denote the end of an input. (E.g. instead of testing it on input *aaabbb*, I would test it on input *aaabbb*\$). Either convention is fine for this class.

**Definition 2: Acceptance**

A PDA $M = (Q, \Sigma, \Gamma, \delta, s, F)$ *accepts* input $w = w_1 w_2 \dots w_m$, $w_i \in \Sigma_\epsilon$ if sequences 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:

- $r_0 = s$ and $s_0 = \epsilon$ ($M$ starts out in the start state with an empty stack.)
- For $i = 0, \dots, m-1$,
  $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, $s_i = at$, and $s_{i+1} = bt$ for $a, b \in \Gamma_\epsilon$ $t \in \Gamma^*$. (Given this input string, $M$ moves as expected over both the states and the stack.)
- $r_m \in F$ ($M$ ends in an accept state.)

# References

[Sip96] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.