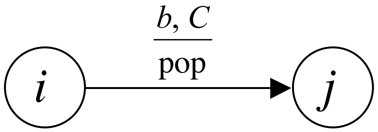
Section 12.2 Pushdown Automata

A pushdown automaton (PDA) is a finite automaton with a stack that has stack operations pop, push, and nop. PDAs always start with one designated symbol on the stack. A state transition depends on the input symbol and the top of the stack. The machine then performs a stack operation and enters the next state.

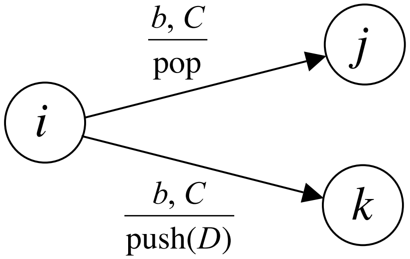
Representation can be graphical or with sets of 5-tuples. For example:

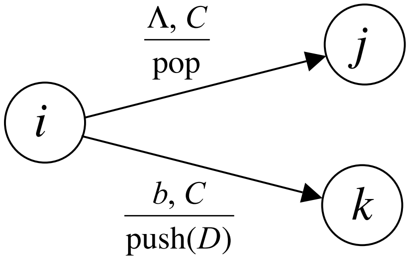


or (i, b, C, pop, j)

Execution: If the machine is in state *i* and the input letter is *b* and *C* is on top of the stack, then pop the stack and enter state *j*.

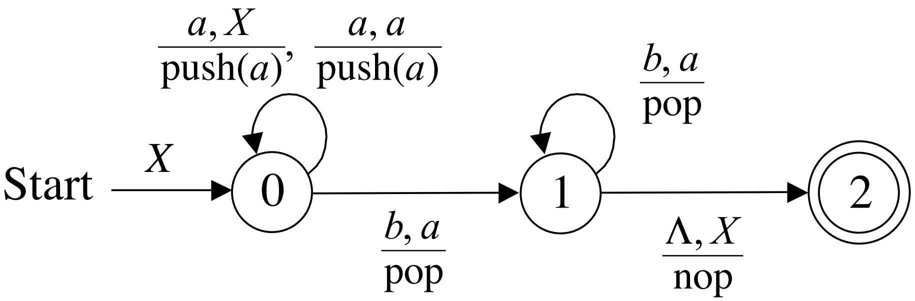
Nondeterminism can occur in two ways, as in the following examples.





Acceptance: A string w is *accepted* by a PDA if there is a path from the start state to a final state such that the input symbols on the path edges concatenate to w. Otherwise, w is *rejected*.

Example. A PDA to accept the language $\{a^n b^n \mid n > 0\}$ as a graph and as a set of 5-tuples.



$\frac{a, X}{\text{push}(a)}$, $\frac{a, a}{\text{push}(a)}$
$\frac{b, a}{\text{pop}}$
Start
X
0
1
2
$\frac{b, a}{\text{pop}}$
$\frac{\Lambda, X}{\text{nop}}$

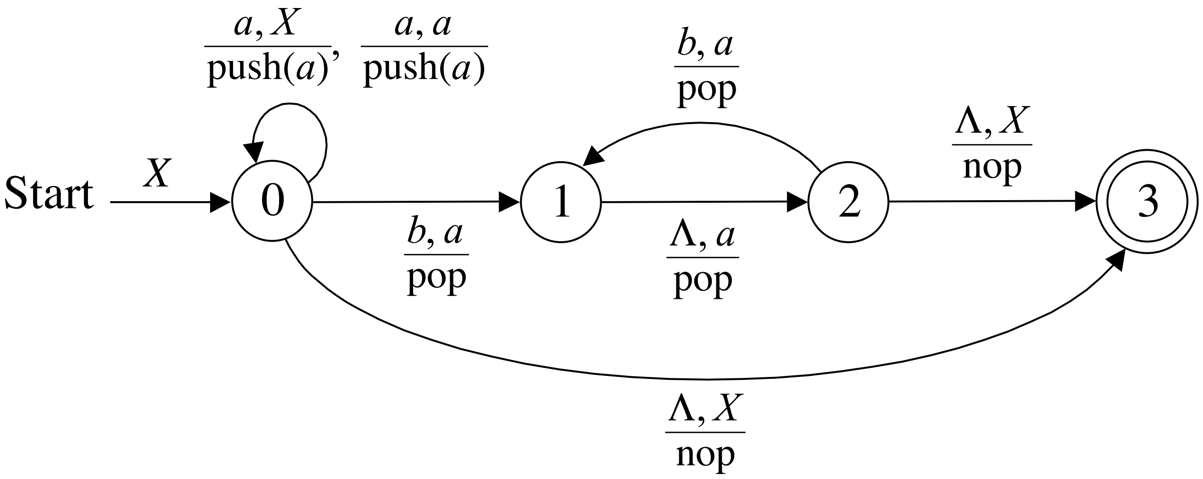
$(0, a, X, \text{push}(a), 0)$
$(0, a, a, \text{push}(a), 0)$
$(0, b, a, \text{pop}, 1)$
$(1, b, a, \text{pop}, 1)$
$(1, \Lambda, X, \text{nop}, 2)$.

Quiz. How would you modify the machine to accept $\{a^n b^n \mid n \in \mathbf{N}\}$?

Answer: Add the instruction $(0, \Lambda, X, \text{nop}, 2)$.

Example/*Quiz*. Find a PDA to accept the language $\{a^{2n} b^n \mid n \in \mathbf{N}\}$.

A solution:



$\frac{a, X}{\text{push}(a)}$, $\frac{a, a}{\text{push}(a)}$
$\frac{b, a}{\text{pop}}$
$\frac{\Lambda, X}{\text{nop}}$
Start
X
0
1
2
3
$\frac{b, a}{\text{pop}}$
$\frac{\Lambda, a}{\text{pop}}$
$\frac{\Lambda, X}{\text{nop}}$

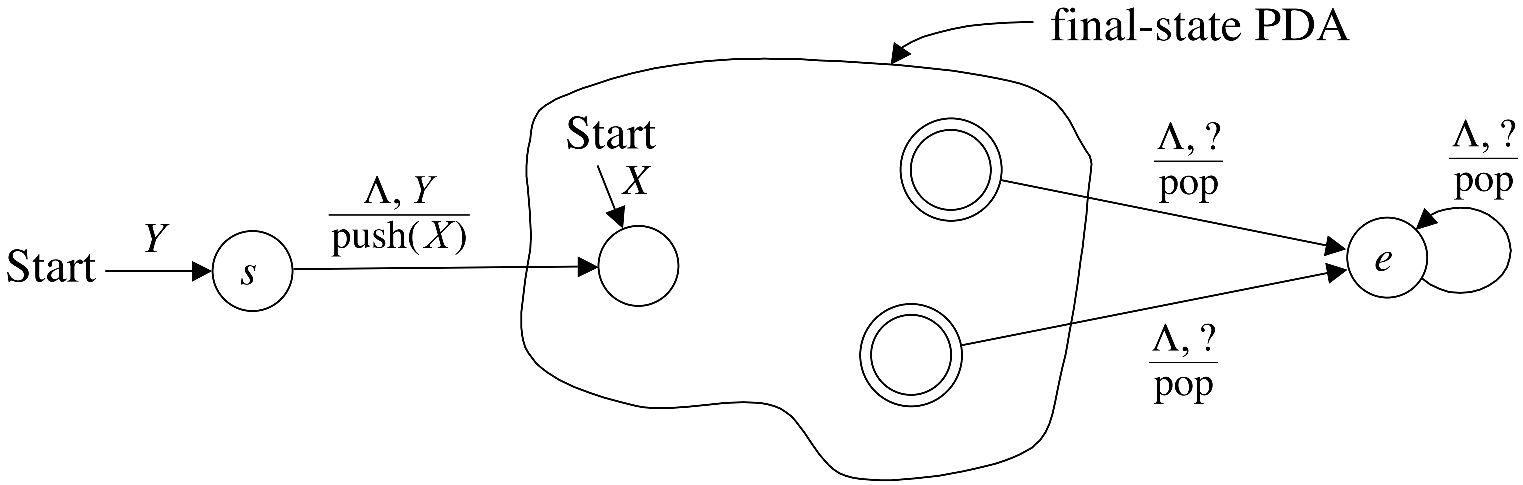
<u>ID for input *aaaabb*:</u>

$(0, aaaabb, X)$
$(0, aaabb, aX)$
$(0, aabb, aaX)$
$(0, abb, aaaX)$
$(0, bb, aaaaX)$
$(1, b, aaaX)$
$(2, b, aaX)$
$(1, \Lambda, aX)$
$(2, \Lambda, X)$
$(3, \Lambda, X)$.

Empty-Stack Acceptance. A PDA can also be defined to accept a string by the condition that the stack is empty. The two types of acceptance are equivalent in that they both define PDAs that accept the same class of languages.

Transform final-state acceptance to empty-stack acceptance:

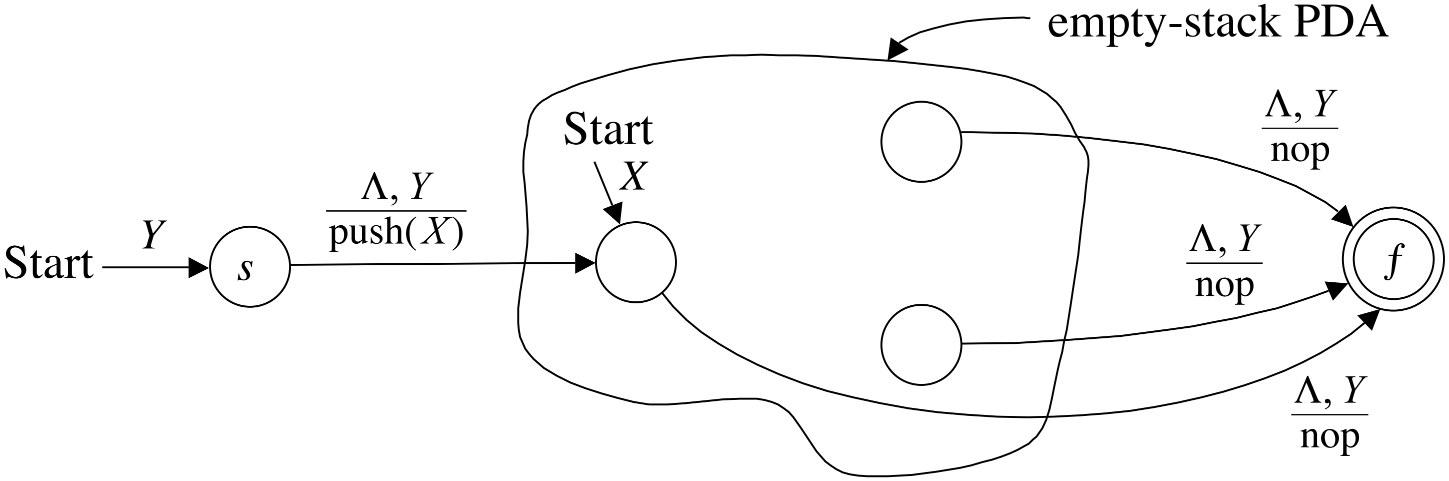
Introduce a new start state *s*, a new start symbol *Y*, a new empty state *e*, and construct edges and actions as shown in the diagram below. Note that ? stands for any stack symbol.


final-state PDA
Start
Y
s
$\frac{\Lambda, Y}{\text{push}(X)}$
Start
X
$\frac{\Lambda, ?}{\text{pop}}$
$\frac{\Lambda, ?}{\text{pop}}$
e
$\frac{\Lambda, ?}{\text{pop}}$

Quiz: If the final-state PDA is deterministic, the transformed empty-stack PDA could be nondeterministic. Why?

Answer: If the final-state PDA has an edge emitted from a final state, then the empty-stack PDA constructs a nondeterministic choice from that final state. For example, a final-state PDA for $\{ab^nc^n \mid n \in \mathbf{N}\}$ could have a final state *k* to accept the string *a* that also emits an edge to take care of any string $ab^{n+1}c^{n+1}$. The empty-stack PDA would construct a nondeterministic choice from *k* by adding instructions of the form (*k*, Λ, ?, pop, *e*).

Transform empty-stack acceptance to final-state acceptance:
Introduce a new start state s, a new start symbol Y, a new final state f, and construct edges and actions as shown in the diagram below.


empty-stack PDA
Start
Y
s
$\frac{\Lambda, Y}{\text{push}(X)}$
Start
X
$\frac{\Lambda, Y}{\text{nop}}$
$\frac{\Lambda, Y}{\text{nop}}$
$\frac{\Lambda, Y}{\text{nop}}$
f

Quiz: If the empty-stack PDA is deterministic, then the transformed final-state PDA is also deterministic. Why?

Answer: Because the instructions of the empty-stack PDA do not use stack symbol Y. So the only time a new instruction can be executed is when Y is on top of the stack, meaning that the empty-stack PDA has emptied it's original stack and can't move.

Theorem: The context-free languages are exactly the languages accepted by PDAs.

Proof: We'll see two algorithms, one to transform a C-F grammar into an empty-stack PDA, and one to transform an empty-stack PDA into a C-F grammar. QED.

Transform a C-F grammar to empty-stack PDA

The idea is to construct a PDA whose execution corresponds to a leftmost derivation. The PDA has one state, but we allow multiple stack operations. The stack symbols are the terminals and nonterminals of the grammar. The designated starting stack symbol is the grammar start symbol. We'll illustrate the algorithm with an example.

Grammar	Terminals	Corresponding PDA Instructions
$S \to aSb \mid BC$	a	$(0, a, a, \text{pop}, 0)$
$B \to bB \mid b$	b	$(0, b, b, \text{pop}, 0)$
$C \to cC \mid \Lambda.$	c	$(0, c, c, \text{pop}, 0)$
	Productions	
	$S \to aSb$	$(0, \Lambda, S, \langle \text{pop}, \text{push}(b), \text{push}(S), \text{push}(a) \rangle, 0)$
	$S \to BC$	$(0, \Lambda, S, \langle \text{pop}, \text{push}(C), \text{push}(B) \rangle, 0)$
	$B \to bB$	$(0, \Lambda, B, \langle \text{pop}, \text{push}(B), \text{push}(b) \rangle, 0)$
	$B \to b$	$(0, \Lambda, B, \langle \text{pop}, \text{push}(b) \rangle, 0)$
	$C \to cC$	$(0, \Lambda, C, \langle \text{pop}, \text{push}(C), \text{push}(c) \rangle, 0)$
	$C \to \Lambda$	$(0, \Lambda, C, \text{pop}, 0)$

Example. A leftmost derivation of *abbcb*:

$$\begin{aligned} S &\Rightarrow aSb \\ &\Rightarrow aBCb \\ &\Rightarrow abBCb \\ &\Rightarrow abbCb \\ &\Rightarrow abbcCb \\ &\Rightarrow abbcb. \end{aligned}$$

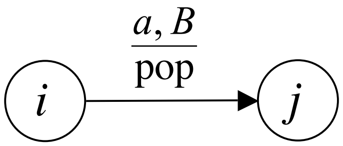
Example. ID for input string *abbcb*:

$(0, abbcb, S)$
$(0, abbcb, aSb)$
$(0, bbcb, Sb)$
$(0, bbcb, BCb)$
$(0, bbcb, bBCb)$
$(0, bcb, BCb)$
$(0, bcb, bCb)$
$(0, cb, Cb)$
$(0, cb, cCb)$
$(0, b, Cb)$
$(0, b, b)$
$(0, \Lambda, \Lambda)$.

Transform an empty-stack PDA to a C-F grammar

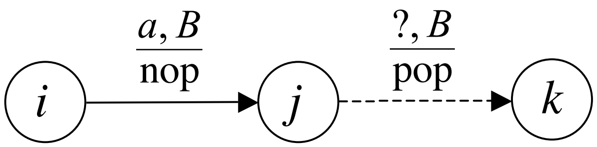
The idea is to construct a grammar whose leftmost derivations correspond to computation sequences. For each stack symbol B and each pair of states i and j construct a nonterminal B_{ij} from which a leftmost derivation will correspond to a computation sequence that starts in state i with B on the stack and ends in state j with B popped from the stack.

PDA Instruction — Corresponding Grammar Productions

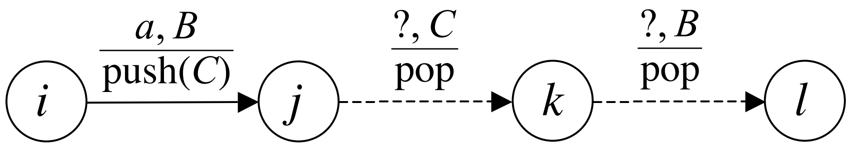


$B_{ij} \to a$

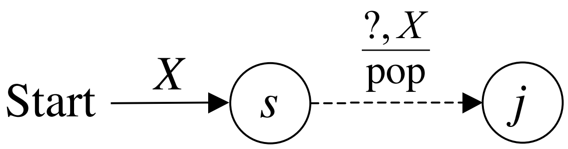
(For the following instructions, the dotted lines represent *possible* paths to states that pop the desired stack symbol.)



$B_{ik} \to aB_{jk}$ for each state k.

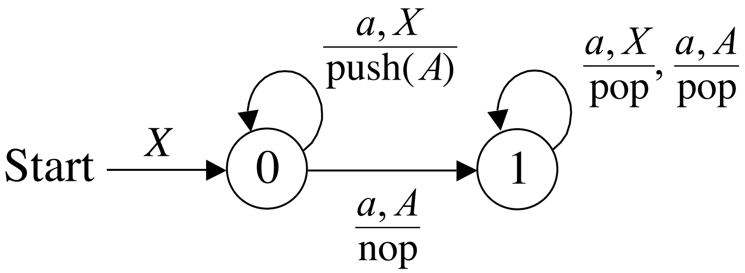


$B_{il} \to aC_{jk}B_{kl}$ for each state k and l.



$S \to X_{sj}$ for each state j. (S is start symbol)

Example/Quiz. Use the algorithm to transform the following empty-stack PDA into a C-F grammar.


$\frac{a, X}{\text{push}(A)}$
$\frac{a, X}{\text{pop}}, \frac{a, A}{\text{pop}}$
Start
X
0
1
$\frac{a, A}{\text{nop}}$

Solution:

The start state 0 with X on the stack gives: $S \rightarrow X_{00} \mid X_{01}$

The pop operation (1, a, X, pop, 1) gives: $X_{11} \rightarrow a$

The pop operation (1, a, A, pop, 1) gives: $A_{11} \rightarrow a$

The nop operation (0, a, A, nop, 1) gives:

$$A_{00} \rightarrow aA_{10}$$
$$A_{01} \rightarrow aA_{11}$$

The push operation (0, a, X, push(A), 0) gives:

$$X_{00} \rightarrow aA_{00}X_{00} \mid aA_{01}X_{10}$$
$$X_{01} \rightarrow aA_{00}X_{01} \mid aA_{01}X_{11}$$

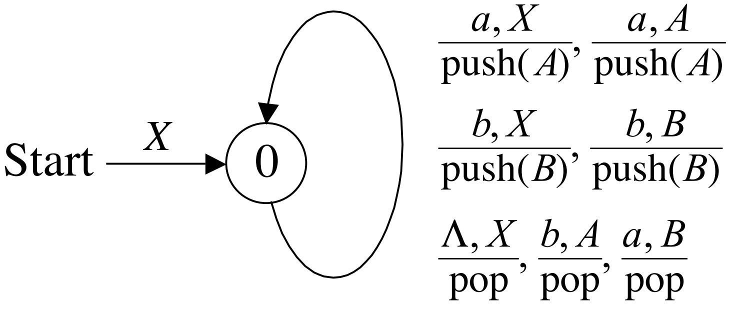
Quiz: Simplify the preceding grammar by deleting productions that do not derive terminal strings and productions that are not reachable from the start symbol.

Solution: $S \rightarrow X_{01}$

$$X_{01} \rightarrow aA_{01}X_{11}$$
$$A_{01} \rightarrow aA_{11}$$
$$X_{11} \rightarrow a$$
$$A_{11} \rightarrow a$$

or, $S \rightarrow aaaa$.

Example. We'll try to find a grammar for the language $L = \{w \in \{a, b\}^* \mid n_a(w) = n_b(w)\}$ by constructing an empty-stack PDA to accept L and then transforming it into a C-F grammar. Consider the following single-state empty-stack PDA.


Start
X
0
$\frac{a, X}{\text{push}(A)}, \frac{a, A}{\text{push}(A)}$
$\frac{b, X}{\text{push}(B)}, \frac{b, B}{\text{push}(B)}$
$\frac{\Lambda, X}{\text{pop}}, \frac{b, A}{\text{pop}}, \frac{a, B}{\text{pop}}$

Idea of Proof: Assume, WLOG, that the input string begins with a. Then A is pushed and this action continues for each consecutive a. When the first b occurs, A is popped, which means that an a-b pair has been counted. The only way to get back to X on top is if the string has an equal number of a's and b's so far. If the input is exhausted, then pop X and accept the string. Otherwise, the process continues as described.

PDA Transformed into C-F grammar:

$S \to X_{00}$
$X_{00} \to \Lambda \mid aA_{00}X_{00} \mid bB_{00}X_{00}$
$A_{00} \to b \mid aA_{00}A_{00}$
$B_{00} \to a \mid bB_{00}B_{00}$.

A Simplified C-F grammar:

$S \to \Lambda \mid aAS \mid bBS$
$A \to b \mid aAA$
$B \to a \mid bBB$.

Example Derivation of aababb:

$S \Rightarrow aAS \Rightarrow aaAAS \Rightarrow aabAS$
$\Rightarrow aabaAAS \Rightarrow aababAS$
$\Rightarrow aababbS \Rightarrow aababb$.

Nondeterministic PDAs are ***more*** *powerful than deterministic PDAs*.
An example is to consider the language of even palindromes over {*a, b*}. A context-free grammar for the language is given by

$S \rightarrow \Lambda \mid aSa \mid bSb$

Any PDA to accept the language must make a nondeterministic decision to start comparing the second half of a string with the reverse of the first half.

Quiz: Transform the grammar into a PDA

A Solution (using the algorithm)

(0, *a, a*, pop, 0)

(0, *b, b*, pop, 0)

(0, Λ, *S*, pop, 0)

(0, Λ, *S*, ⟨pop, push(*a*), push(*S*), push(*a*)⟩, 0)

(0, Λ, *S*, ⟨pop, push(*b*), push(*S*), push(*b*)⟩, 0).