Procedure for Converting a PDA to a CFG

First modify the PDA as follows:

1. Only one character must be popped from the stack at a time. (All of the PDAs we've seen do this). It may be necessary to add extra states to make this happen.

   For convenience, we will use the functional form describing transitions on the PDA:

   $\delta(q,u,A) = (r,B)$

   where

   q is the "from" state
   u is the consumed character
   A is the popped character
   r is the "to" state
   B represents the pushed character(s)

2. For every transition that does not inspect the stack (i.e., the pop character is λ), add one transition that pops a single character and pushes it back again, *for each letter in the stack alphabet*. For example, if the stack alphabet is $\{X, Y\}$, and you have the transition

   $\delta(q,u,\lambda) = (r,X)$

   then you *add* the transitions

   $\delta(q,u,X) = (r,XX)$
   $\delta(q,u,Y) = (r,XY)$

   thus leaving the new X on the top of the stack.

3. The conversion process proceeds as follows:

   a) Add a rule $S \rightarrow \langle s\lambda f \rangle$ for the start state, s, and *each final state*, f.
   For example:
   $S \rightarrow \langle s\lambda f_1 \rangle$
   $S \rightarrow \langle s\lambda f_2 \rangle$

   b) Add a rule $\langle q\lambda q \rangle \rightarrow \lambda$ for each state q.
   For example:
   $\langle q_1\lambda q_1 \rangle \rightarrow \lambda$
   $\langle q_2\lambda q_2 \rangle \rightarrow \lambda$

c) For each transition, in the PDA, that pushes a single character (including λ), such as

$\delta(q,u,A) = (r,B)$

add rules of the form

$<qAp> \rightarrow u<rBp>$

for each state p *in the machine*. The letter *u* can be lambda (in which case it disappears).

d) For each state in the PDA that pushes two (or more) characters, such as

$\delta(q,u,A) = (r,BC)$

add rules of the form

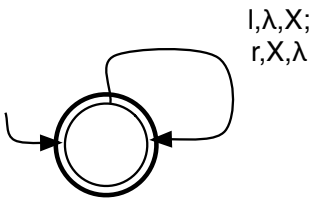
$<qAp> \rightarrow u<rBt><tCp>$

for all possible combinations of states p and t in the machine. (This pattern extends to arbitrary length strings pushed onto the stack: $<qAp> \rightarrow u<rBt><tCn><nDp>$, etc.)

Don't bother with non-terminals $<qAr>$ where it is not possible in the PDA to go from state *q* to state *r* (i.e., *omit* any rules where a path of any length does not exist to get from state *q* to state *r*). Simplify the grammar to taste (rename non-terminals for each "$<q_nA_nr_n>$", combine transitions with the same start term separating destination terms with "|", eliminate unit productions, dead-ends and jails) and you're (much easier said than) done!

Example

Here is a PDA for the language of balanced parentheses:



Here we are using $\Sigma = \{l, r\}$—for "left" and "right"—and $\Gamma = \{X\}$. In functional form, this PDA is:

$\delta(s,l,e) = (s,X)$ (rule 1)
$\delta(s,r,X) = (s,e)$ (rule 2)

First, we need to add the following rule to accompany rule 1. Note that we still *keep* rule 1, we just add the following new rule to the machine:

$\delta(s,l,X) = (s,XX)$ (rule 3)

Now we can construct the grammar. First the boilerplate rules (types **a** and **b** listed above):

S → <sλs>
<sλs> → λ

Now from rule 1 above we get the type **c** rule

<sλs> → l<sXs>

and for rule 2 we get

<sXs> → r<sλs>

For rule 3 we have

<sXs> → l<sXs><sXs>

Notice how things are simpler when you have only one state—you don't have to consider combinations of states in the generated rules.

Renaming <sλs> = S (and collapsing the unit production S → <sλs>), and <sXs> = B, the grammar becomes:

S → lB | λ
B → rS | lBB

As a sanity check, we can generate "lrllrlrr" as follows:

S => lB => lrS => lrlB => lrllBB => lrllrSB => lrllrlBB => lrllrlrSB => lrllrlrB => lrllrlrrS
⇨ lrllrlrr