EQUIVALENCE OF PDAs AND CFGs

iACADEMY
SCHOOL OF COMPUTING • SCHOOL OF BUSINESS • SCHOOL OF DESIGN

SCHOOL OF COMPUTING
MITCH M. ANDAYA

INTRODUCTION

- A context-free grammar (CFG) can generate the strings of the context-free language it represents.
- A pushdown automaton (PDA) can recognize a context-free language.
- Are PDAs and CFGs equivalent?
- A context-free language is any language that can be generated by some context-free grammar.

INTRODUCTION

- Since PDAs recognize context-free languages, can it be claimed that a language is context-free if and only if some pushdown automaton recognizes it?

- To answer the question, it must be shown that:

1. If a language is context-free, then some pushdown automaton recognizes it.

2. If a pushdown automaton recognizes some language, then it is context-free.

INTRODUCTION

- A context-free language L has a context-free grammar G that can be used to generate or derive the strings of the language.

- To show that context-free language L has some pushdown automaton that recognizes it, a procedure must be established to convert context-free grammar G of language L into its equivalent pushdown automaton P.

CONVERTING CFG TO PDA

- Case Study:

  Assume a certain context-free language L_1 is represented by grammar G_1:

$$S \rightarrow AB$$
$$A \rightarrow 0A \mid 0$$
$$B \rightarrow 0B1 \mid \varepsilon$$

CONVERTING CFG TO PDA

Recall how a grammar is used to generate a string of the language.

To derive the string 0001 using the leftmost derivation:

$S \rightarrow AB$

$A \rightarrow 0A \mid 0$

$B \rightarrow 0B1 \mid \varepsilon$

$$
\begin{aligned}
S &\rightarrow \overset{0A}{A}B \\
&\rightarrow 0\overset{0}{A}B \\
&\rightarrow 00\overset{0B1}{B} \\
&\rightarrow 000\overset{\varepsilon}{B}1 \\
&\rightarrow 0001
\end{aligned}
$$

CONVERTING CFG TO PDA

To show that language L_1 has a pushdown automaton that recognizes it, a PDA P_1 must be constructed from grammar G_1. This PDA should be able to determine if an input string belongs to context-free language L_1.

To do this, P_1 must have the capability to determine whether there is a series of substitutions using the rules of grammar G_1 that can generate the input string.

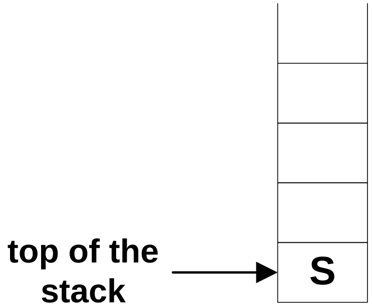
P_1 should be able to "simulate" the leftmost derivation of the input string. The stack is used to record the steps of the derivation.

CONVERTING CFG TO PDA

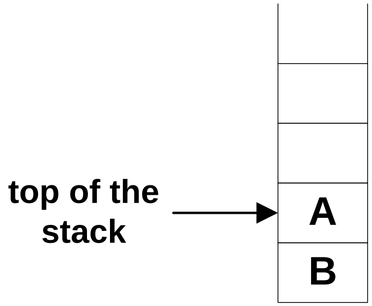
S → AB
→ 0AB
→ 00B
→ 000B1
→ 0001

As an example of how the stack is used in simulating the leftmost derivation of the string 0001 using grammar G_1:

1. First, push the start variable *S* onto the stack. The stack contents will be:

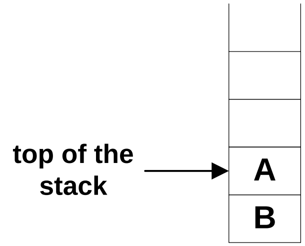


2. Using the rule $S \rightarrow AB$, remove *S* from the stack and replace it with *AB*. The stack contents will now be:



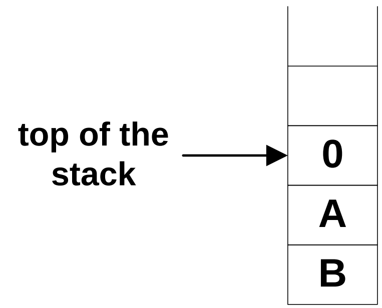
iACADEMY
SCHOOL OF COMPUTING • SCHOOL OF BUSINESS • SCHOOL OF DESIGN

CONVERTING CFG TO PDA

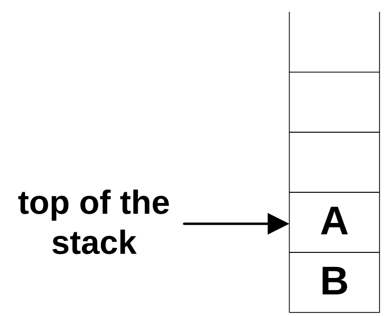


S → AB
→ 0AB
→ 00B
→ 000B1
→ 0001

3. Using the rule *A* → 0*A*, remove *A* from the stack and replace it with 0*A*. The stack contents will now be:

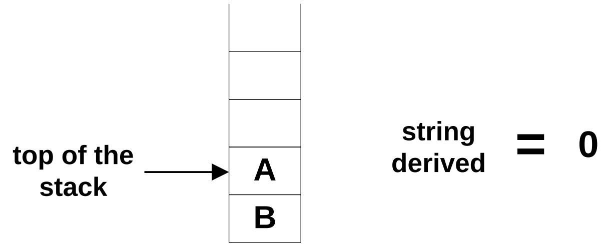


4. Remove 0 from the stack. The stack contents will now be:



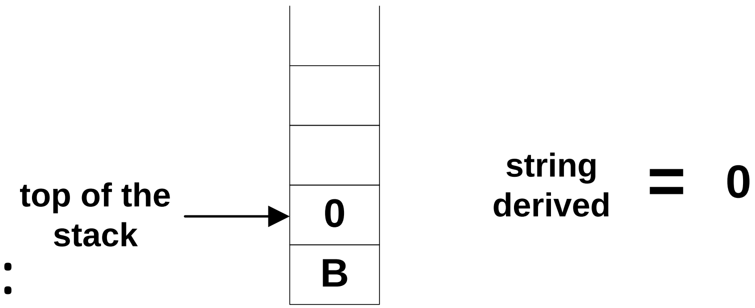
string derived = 0

CONVERTING CFG TO PDA

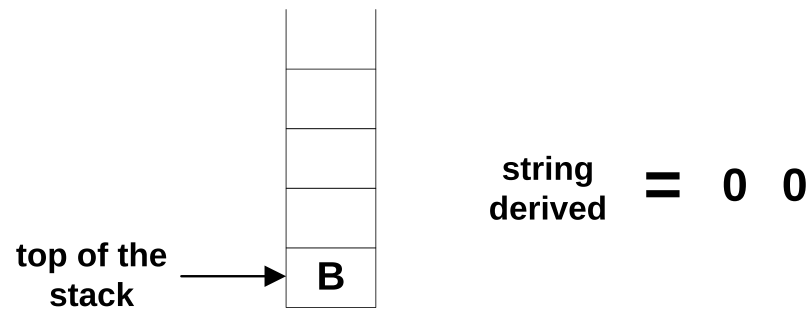


S → AB
→ 0AB
→ 00B
→ 000B1
→ 0001

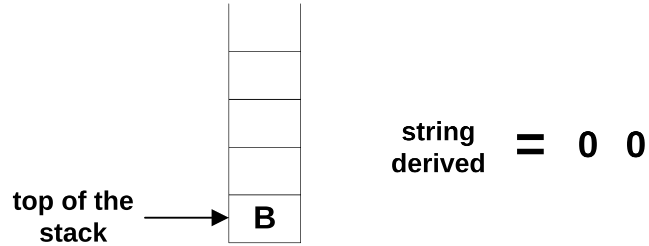
5. Using the rule $A \to 0$, remove A from the stack and replace it with 0. The stack contents will now be:



6. Remove 0 from the stack. The stack contents will now be:

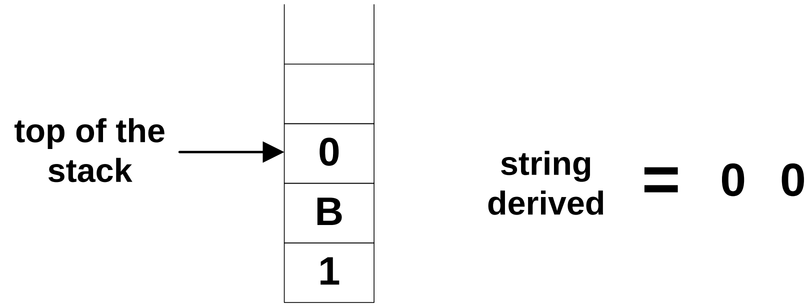


CONVERTING CFG TO PDA

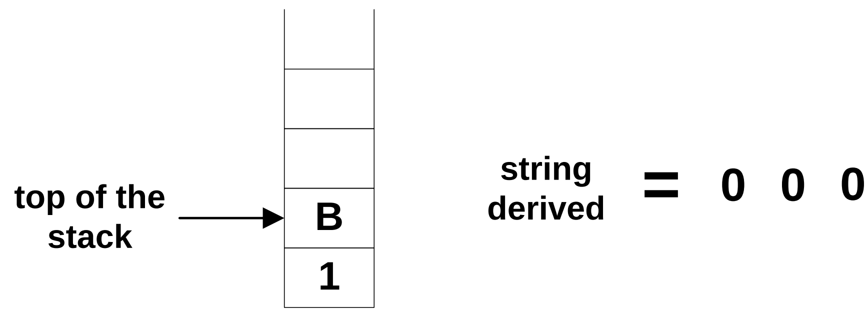


$S \rightarrow AB$
$\rightarrow 0AB$
$\rightarrow 00B$
$\rightarrow 000B1$
$\rightarrow 0001$

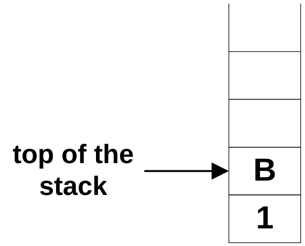
7. Using the rule $B \rightarrow 0B1$, remove B from the stack and replace it with $0B1$. The stack contents will now be:



8. Remove 0 from the stack. The stack contents will now be:



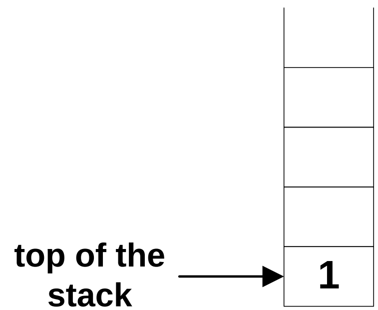
CONVERTING CFG TO PDA



string derived = 0 0 0

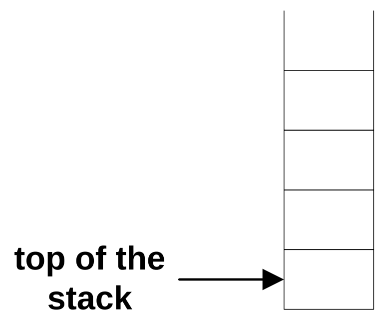
S → AB
→ 0AB
→ 00B
→ 000B1
→ 0001

9. Using the rule $B \rightarrow \varepsilon$, remove B from the stack. The stack contents will now be:



string derived = 0 0 0

10. Remove 1 from the stack. The stack contents will now be:



string derived = 0 0 0 1

CONVERTING CFG TO PDA

- Hence, to determine if an input string is to be accepted, the PDA P_1 that should be able to perform the following actions:

1. P_1 pushes the stack empty symbol $\$$ onto the stack.

2. P_1 then pushes the start symbol of the grammar onto the stack.

3. The following actions are then repeated:

    a. If the top of stack is a variable, pop that variable from the stack.

       Then, select one of the rules for that variable and push the right-hand side string of that rule onto the stack.

iACADEMY
SCHOOL OF COMPUTING • SCHOOL OF BUSINESS • SCHOOL OF DESIGN

CONVERTING CFG TO PDA

b. If the top of stack is a terminal, pop that terminal from the stack.

   Then, compare that terminal with the next symbol of the input string. If they match, repeat step 3 and processing continues. If they do not match, reject the string.

c. If the top of stack is the stack empty symbol $ (the stack is empty) and there are no more input symbols. Accept the input string (the PDA goes to a final state).

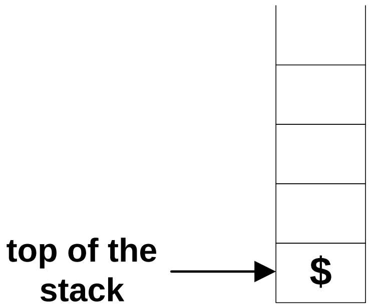
   If the stack is empty and there are input symbols remaining, or if the stack is not empty and there are no more input symbols, reject the input string.

CONVERTING CFG TO PDA

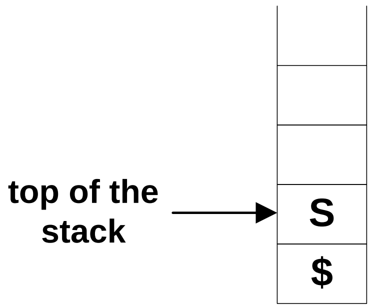
$S \rightarrow AB$
$\rightarrow 0AB$
$\rightarrow 00B$
$\rightarrow 000B1$
$\rightarrow 0001$

- Based on the series of actions given, the following will now be performed by P_1 in order to determine if the string 0001 belongs to language L_1:

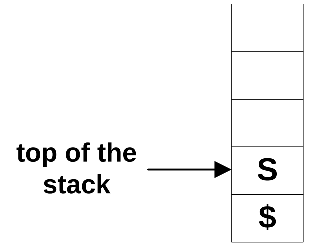
1. P_1 will push the stack empty symbol $ onto the stack. The stack contents will now be:



2. P_1 will push the start symbol S onto the stack. The stack contents will now be:



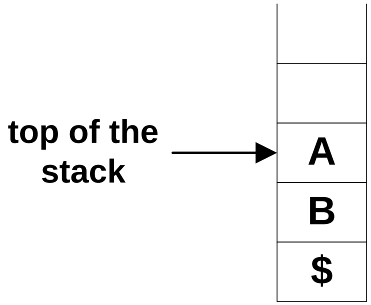
CONVERTING CFG TO PDA



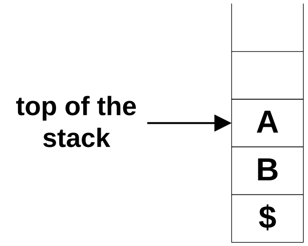
$S \rightarrow AB$
$\rightarrow 0AB$
$\rightarrow 00B$
$\rightarrow 000B1$
$\rightarrow 0001$

3. Since the symbol at the top of the stack is a variable (S), pop it out from the stack and push the right-hand side string of a rule for S ($S \rightarrow AB$) onto the stack.

   The stack contents will now be:



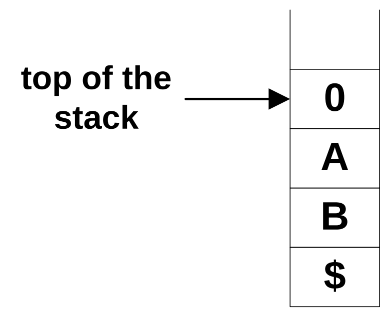
CONVERTING CFG TO PDA



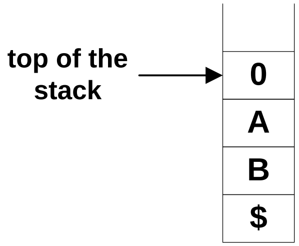
$S \rightarrow AB$
$\rightarrow 0AB$
$\rightarrow 00B$
$\rightarrow 000B1$
$\rightarrow 0001$

4. Since the symbol at the top of the stack is a variable (*A*), pop it from the stack and push the right-hand side string of a rule for the variable *A* ($A \rightarrow 0A$) onto the stack.

   The stack contents will now be



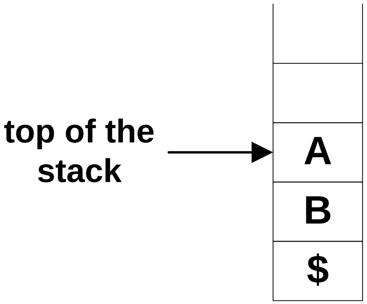
CONVERTING CFG TO PDA



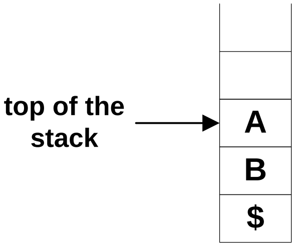
$$
\begin{aligned}
S &\rightarrow AB \\
&\rightarrow 0AB \\
&\rightarrow 00B \\
&\rightarrow 000B1 \\
&\rightarrow 0001
\end{aligned}
$$

5. Since the symbol at the top of the stack is a terminal (0), pop this from the stack and compare it with the first incoming input symbol (0). Since they match, the computation continues.

   The stack contents will now be



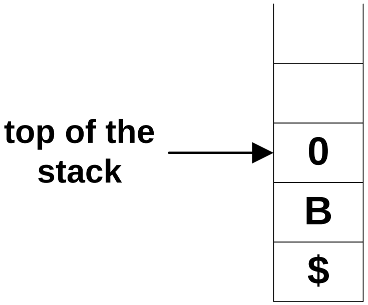
CONVERTING CFG TO PDA



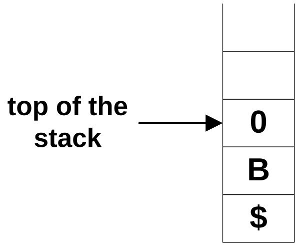
S → AB
→ 0AB
→ 00B
→ 000B1
→ 0001

6. Since the symbol at the top of the stack is a variable (*A*), pop it from the stack and push the right-hand side string of a rule for *A* (*A* → 0) onto the stack.

   The stack contents will now be



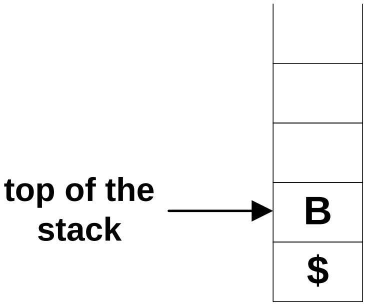
CONVERTING CFG TO PDA



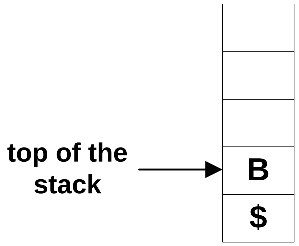
$S \rightarrow AB \rightarrow 0AB \rightarrow 00B \rightarrow 000B1 \rightarrow 0001$

7. Since the symbol at the top of the stack is a terminal (0), pop this from the stack and compare it with the second incoming input symbol (0). Since they match, the computation continues.

   The stack contents will now be



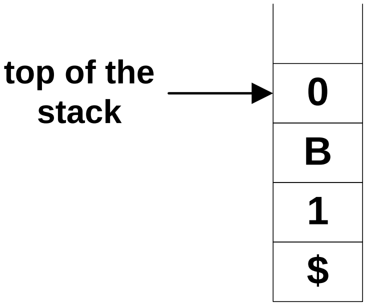
CONVERTING CFG TO PDA



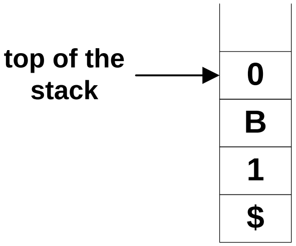
S → AB
→ 0AB
→ 00B
→ 000B1
→ 0001

8. Since the symbol at the top of the stack is a variable (*B*), pop it out from the stack and push the right-hand side string of a rule for the variable *B* (*B* → 0*B*1) onto the stack.

   The stack contents will now be



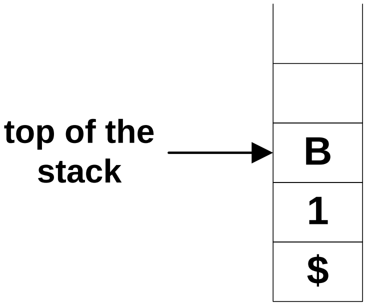
CONVERTING CFG TO PDA



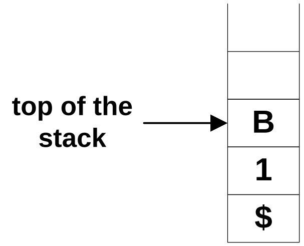
$S \rightarrow AB$
$\rightarrow 0AB$
$\rightarrow 00B$
$\rightarrow 000B1$
$\rightarrow 0001$

9. Since the symbol at the top of the stack is a terminal (0), pop this from the stack and compare it with the third incoming input symbol (0). Since they match, the computation continues.

   The stack contents will now be

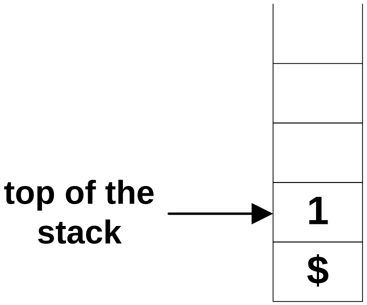


CONVERTING CFG TO PDA



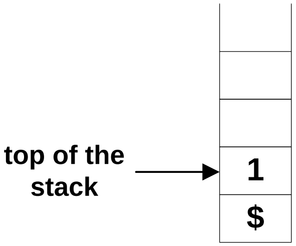
S → AB
→ 0AB
→ 00B
→ 000B1
→ 0001

10. Since the symbol at the top of the stack is a variable (*B*), pop it from the stack and push the right-hand side string of a rule for *B* (*B* → ε) onto the stack. But since the right-hand side string of the selected rule is the empty string, there is no need to push anything.



The stack contents will now be

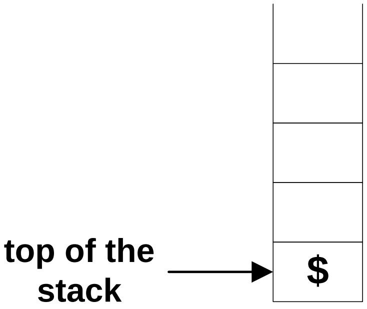
CONVERTING CFG TO PDA



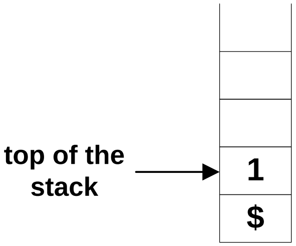
$S \rightarrow AB$
$\rightarrow 0AB$
$\rightarrow 00B$
$\rightarrow 00B1$
$\rightarrow 0001$

11. Since the symbol at the top of the stack is a terminal (1), pop this from the stack and compare it with the fourth incoming input symbol (1). Since they match, the computation continues.

    The stack contents will now be

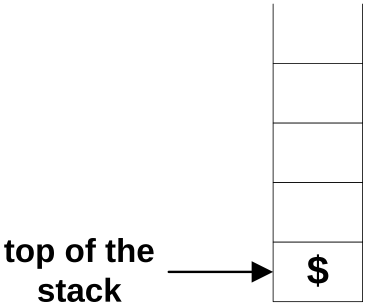


CONVERTING CFG TO PDA



S → AB
→ 0AB
→ 00B
→ 000B1
→ 0001

12. Since the stack is empty and there are no more input symbols left, the string is accepted by PDA P_1.



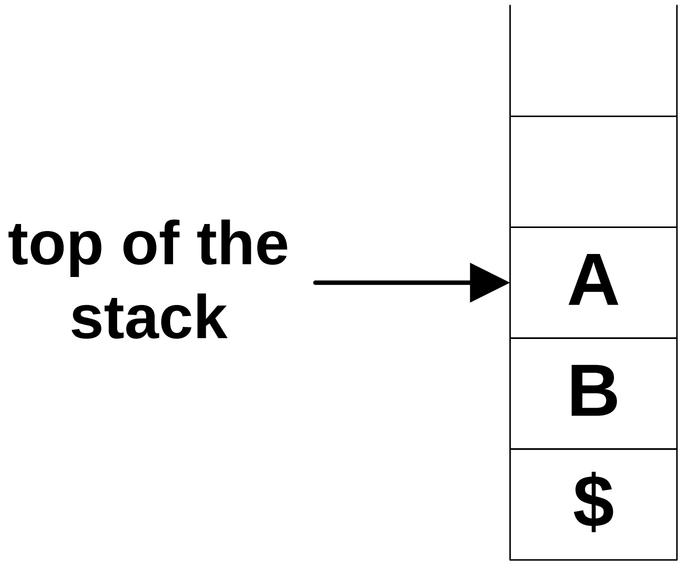
CONVERTING CFG TO PDA

- One problem in the series of actions outlined is that there will be situations where there will be several choices on which rule to use in substituting a certain variable.

- As an example, take note that there are two rules for variable A in grammar G_1.

  One is the rule $A \rightarrow 0A$ while the other one is the rule $A \rightarrow 0$.

CONVERTING CFG TO PDA



$S \rightarrow AB$

$A \rightarrow 0A \mid 0$

$B \rightarrow 0B1 \mid \varepsilon$

Which rule to use?

CONVERTING CFG TO PDA

- Hence, if the symbol at the top of the stack is *A*, the PDA has two options:

   1. One option is to pop variable *A* from the stack and then push the right-hand side string of the rule $A \rightarrow 0A$ onto the stack. In other words, pop *A* and then push $0A$.

   2. The other option is to pop variable *A* from the stack and then push the right-hand side string of the rule $A \rightarrow 0$ onto the stack. In other words, pop *A* and then push 0.

- The same case is also true for variable *B*.

CONVERTING CFG TO PDA

- The question now is which option will the PDA choose? One option may cause the PDA to accept the string while the other may not.

- The answer lies in the nondeterministic nature of PDAs.

  The PDA will make nondeterministic guesses on which rule to use by doing several computations at the same time.

CONVERTING CFG TO PDA

- In the given example, the PDA will make two copies of itself.

  One copy will proceed with the computation using the rule $A \rightarrow 0A$ while the second copy will continue processing the input string using the rule $A \rightarrow 0$.

- In order words, the PDA will try to use all possible rules for substituting a certain variable by making nondeterministic guesses.

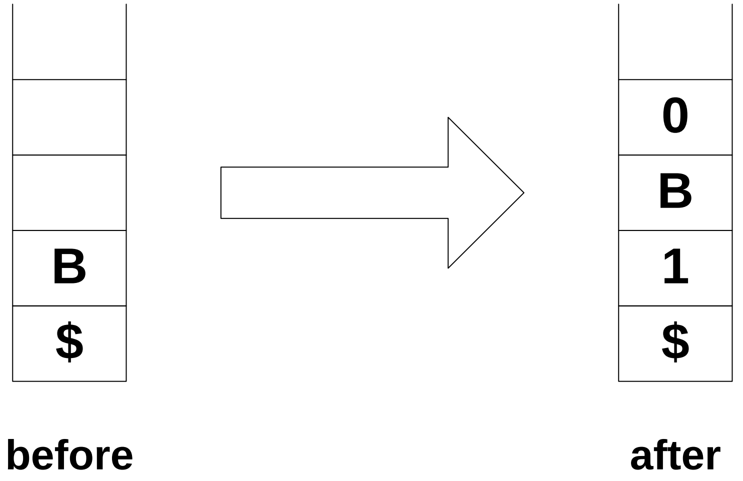
  If the string is indeed part of the language of the PDA, then one or more copies of the PDA will end up in a final state. If the string does not belong to the language, then all copies of the PDA will just die.

CONVERTING CFG TO PDA

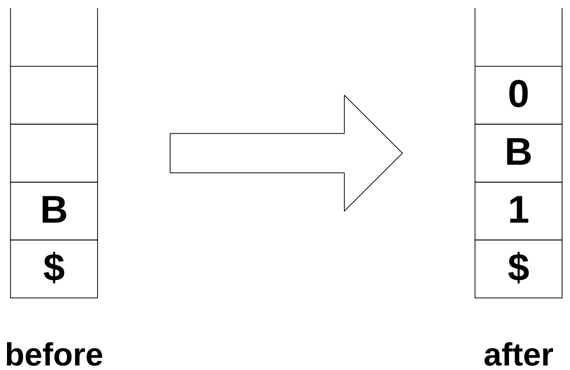
- Observe that the PDA to be constructed requires that an entire string is pushed onto the stack.

- Example:

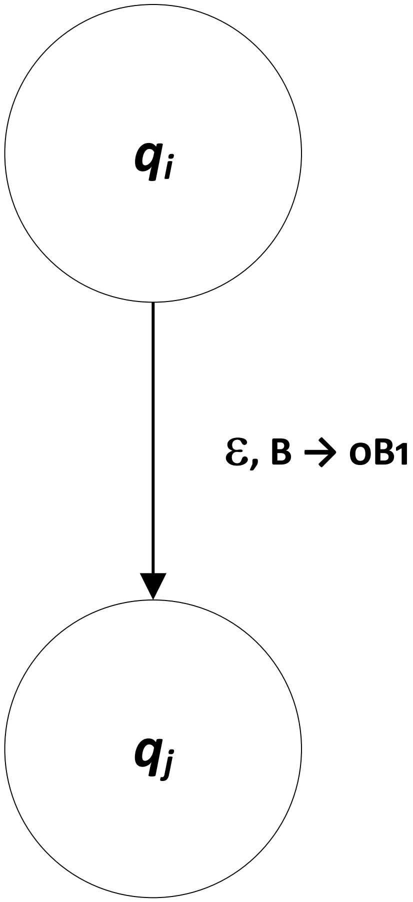
  In step 8 of the previous example, when the symbol at the top of the stack was *B*, it was popped out of the stack and the string 0*B*1 was pushed onto the stack.



CONVERTING CFG TO PDA



- If this action is represented by a state diagram, it would look like:



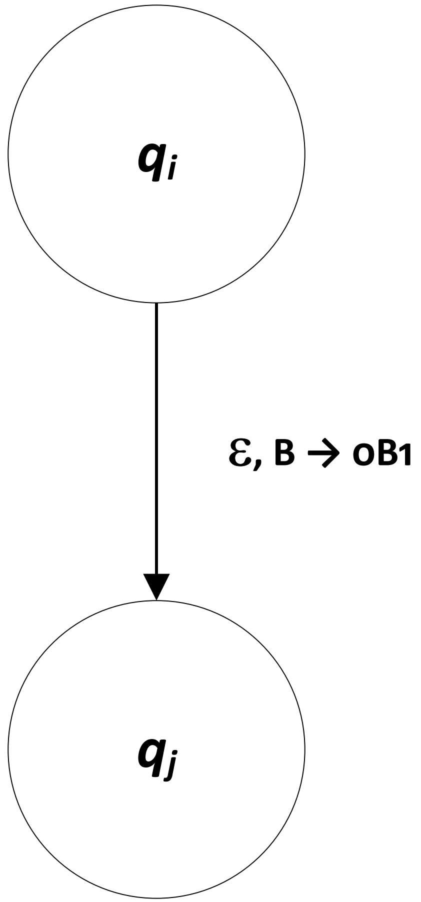
CONVERTING CFG TO PDA

- The transition edge from state q_i to state q_j is ε, $B \rightarrow$ oB1.

  Recall that this label means that even without an input symbol, if the symbol at the top the stack is B, then the PDA will pop B from the stack and then move from state q_i to state q_j, and at the same time push the string oB1 onto the stack.

  This transition can be written as:

$$\delta(q_i, \varepsilon, B) = (q_j, \text{o}B1)$$

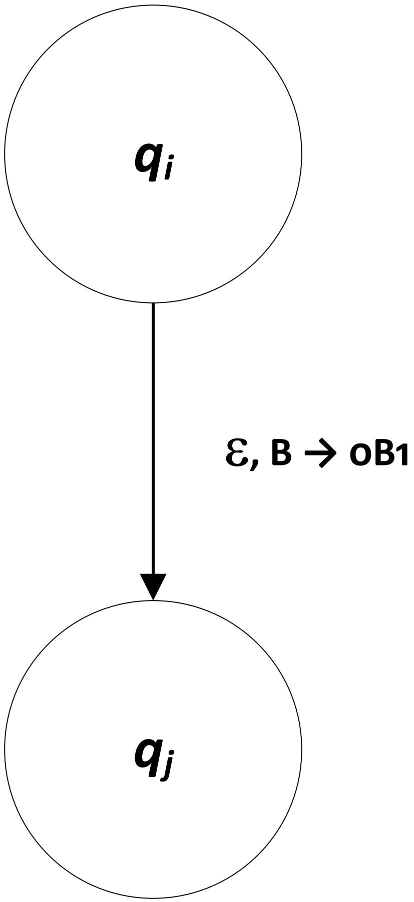


CONVERTING CFG TO PDA

- However, PDAs can push only one symbol onto the stack as it moves from one state to another.

- There is no provision for the PDA to push an entire string in only one state transition.

- The pushing of an entire string onto the stack can be implemented by introducing intermediate states between states q_i and q_j.



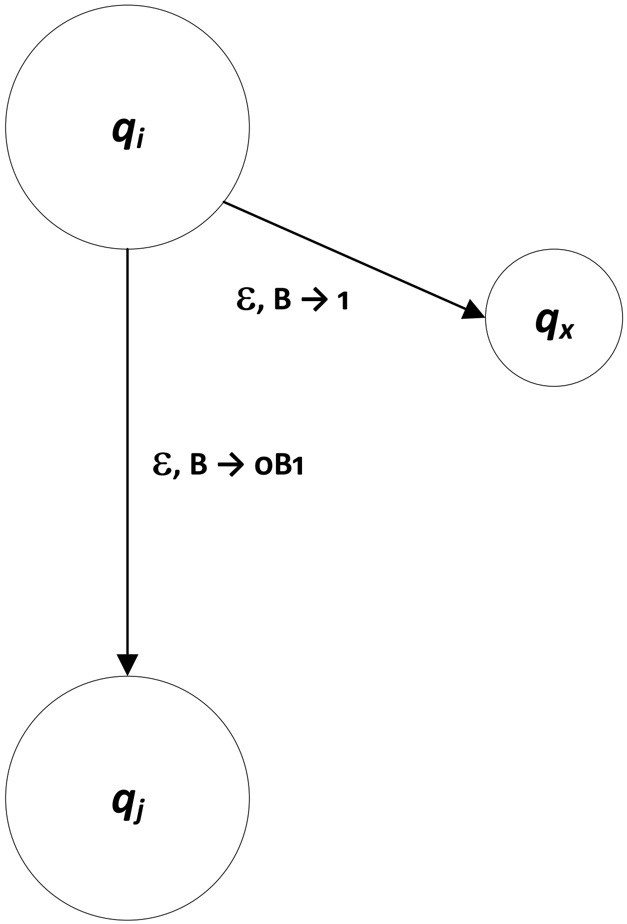
CONVERTING CFG TO PDA

- Instead of having the PDA move from state q_i to state q_j and push the string 0B1 onto the stack, the PDA will perform the following:

1. If the symbol at the top of the stack is B and there is no input symbol, move from state q_i to an intermediate state q_x and then push 1 onto the stack.

   In other words:

$$\delta(q_i, \varepsilon, B) = (q_x, 1)$$

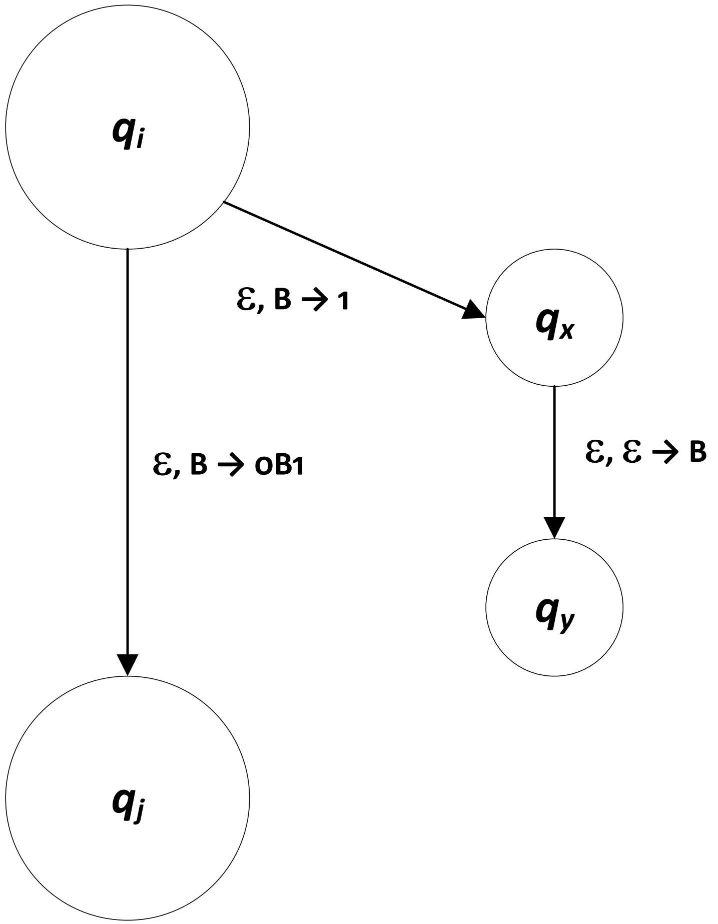


CONVERTING CFG TO PDA

2. Then, from intermediate state q_x, move right away to another intermediate state q_y while pushing B onto the stack.

   In other words:

   $$\delta(q_x, \varepsilon, \varepsilon) = (q_y, B)$$

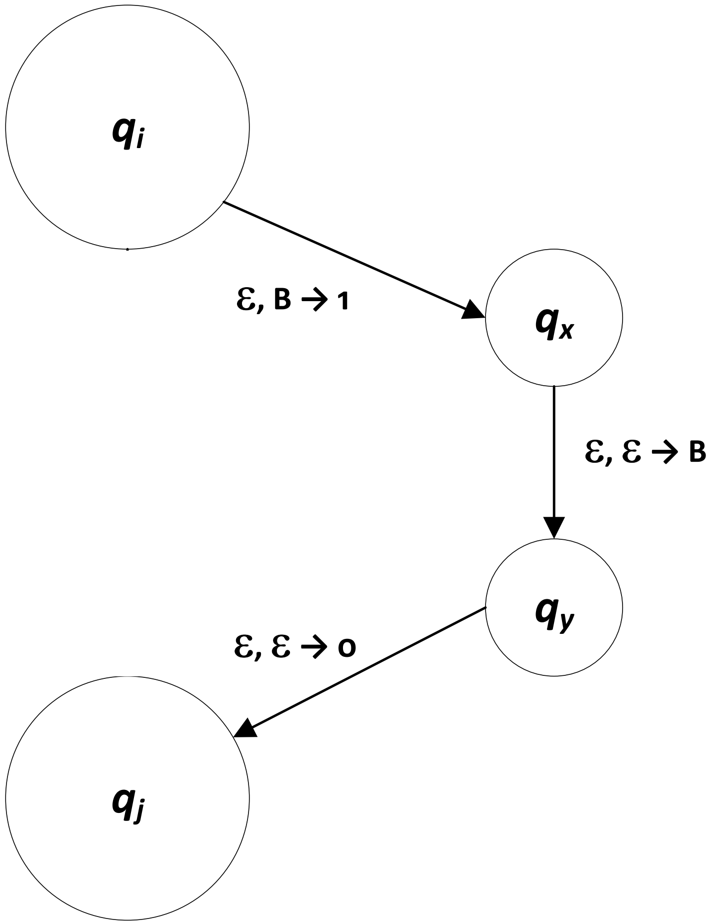


CONVERTING CFG TO PDA

3. Last, from intermediate state q_y, move right away to the destination state q_j while pushing o onto the stack.

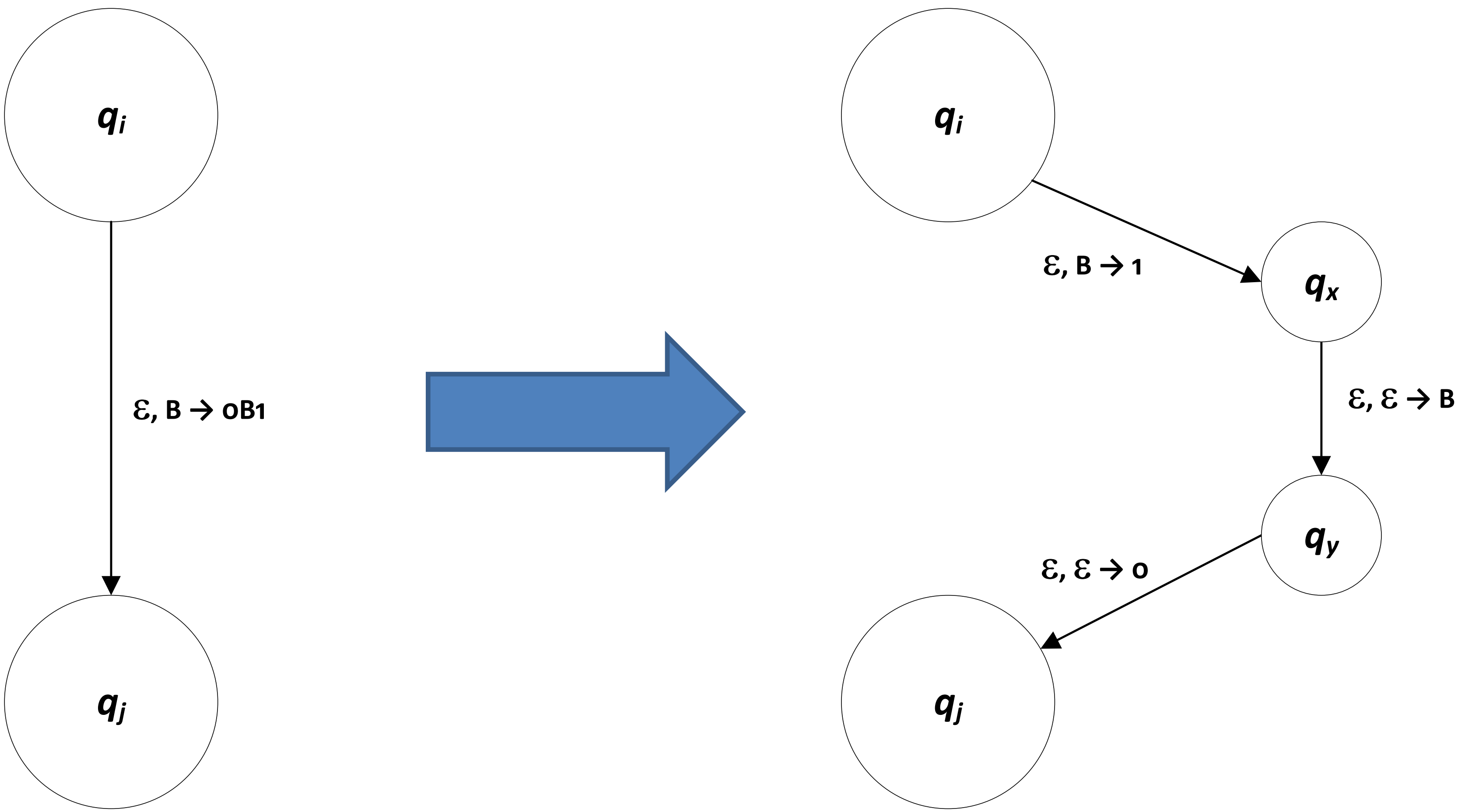
In other words:

$$\delta(q_y, \varepsilon, \varepsilon) = (q_j, o)$$



iACADEMY
SCHOOL OF COMPUTING • SCHOOL OF BUSINESS • SCHOOL OF DESIGN

CONVERTING CFG TO PDA



CONVERTING CFG TO PDA

- Recall the procedure presented earlier on how the PDA to be constructed processes input strings.

1. Before any computation starts, P_1 pushes the stack empty symbol $\$$ onto the stack.

2. P_1 pushes the start symbol of the grammar onto the stack.

3. Repeat the following actions:

    a) If the top of stack is a variable, pop that variable from the stack.

       Then, select one of the rules for that variable and push the right-hand side string of that rule onto the stack.

CONVERTING CFG TO PDA

b) If the top of stack is a terminal, pop that terminal from the stack.

Then, compare that terminal with the next symbol from the input.

If they match, repeat step 3 and processing continues. If they do not match, reject the string.

c) If the top of stack is the stack empty symbol $ (the stack is empty) and there are no more input symbols, accept the input string (the PDA goes to a final state).

CONVERTING CFG TO PDA

- This PDA will therefore have three major states representing the actions it has to perform in determining whether a string belongs to a certain language:

1. q_{start} – this is the start state where the stack empty symbol *$* and then the start variable S of the grammar will be pushed onto the stack. This takes care of steps 1 and 2 listed above.

2. q_{loop} – this state is where the major processing occurs. This takes care of steps 3a, 3b, and 3c listed above

3. q_{final} – this is the final or accept state. The PDA goes to this state from state q_{loop} when the stack is empty and there are no more input symbols.

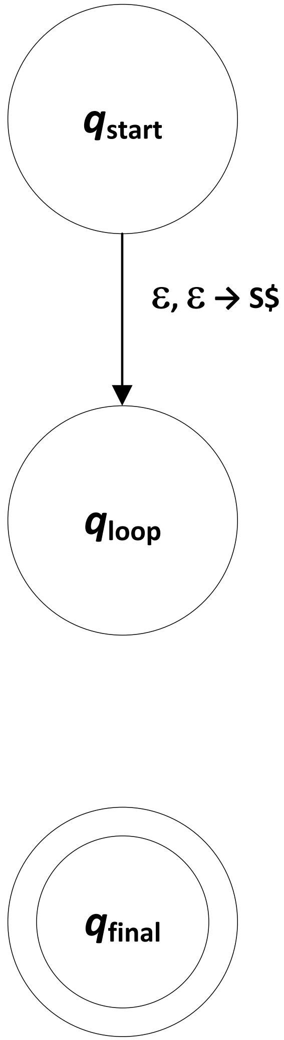
CONVERTING CFG TO PDA

- The state diagram of the PDA to be constructed should be as follows:

  The start state q_{start} will have a transition edge to state q_{loop} with the label:

  $$\varepsilon, \varepsilon \rightarrow S\$$$

  This transition simply states that when the PDA is at state q_{start}, it will move right away to state q_{loop} and at the same time push the string *S$* onto the stack.



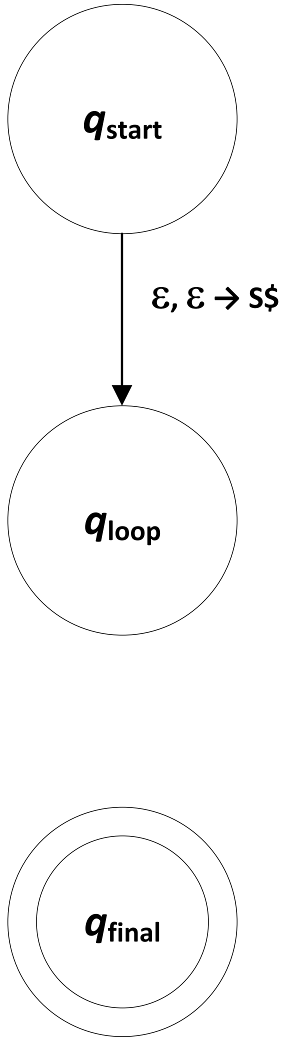
CONVERTING CFG TO PDA

The transition edges for state q_{loop} should be able to handle the following situations:

1. The symbol at the top of the stack is a variable (step 3a).

2. The symbol at the top of the stack is a terminal (step 3b).

3. The symbol at the top of the stack is the stack empty symbol $ indicating that the stack is already empty (step 3c).



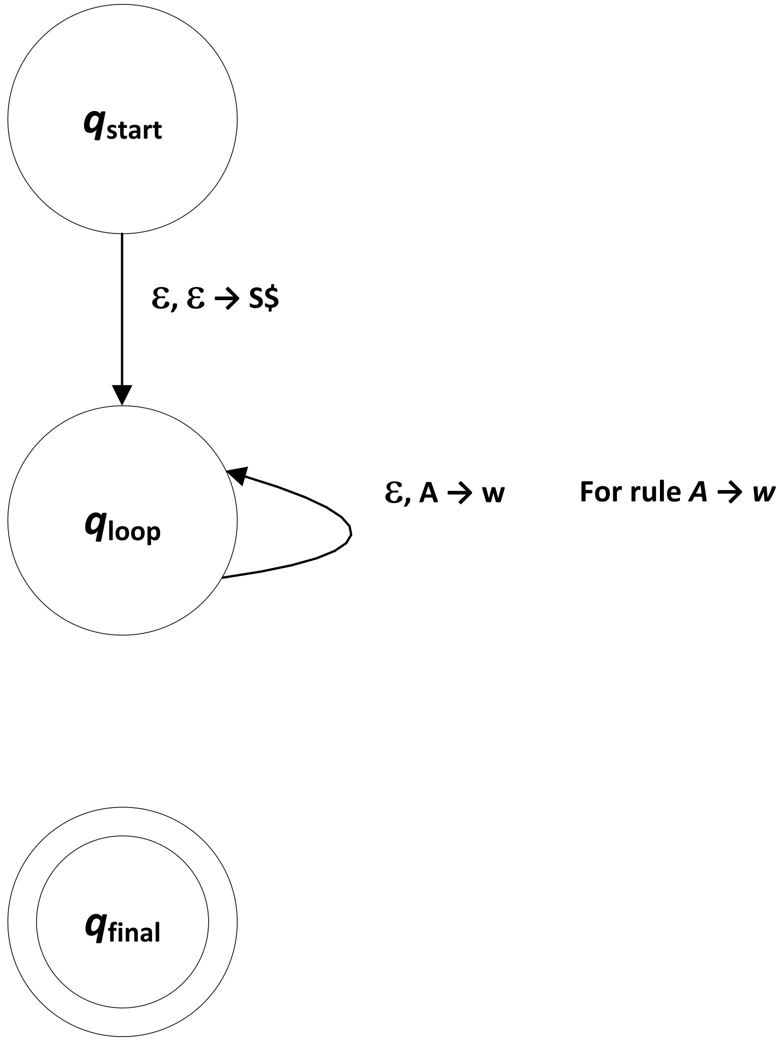
CONVERTING CFG TO PDA

Case 1:

If the top of the stack symbol is a variable *A*, there should be a transition edge with label

$$\varepsilon, A \rightarrow w$$

where *w* is the right-hand side string of the rule *A* → *w* used to replace *A*.

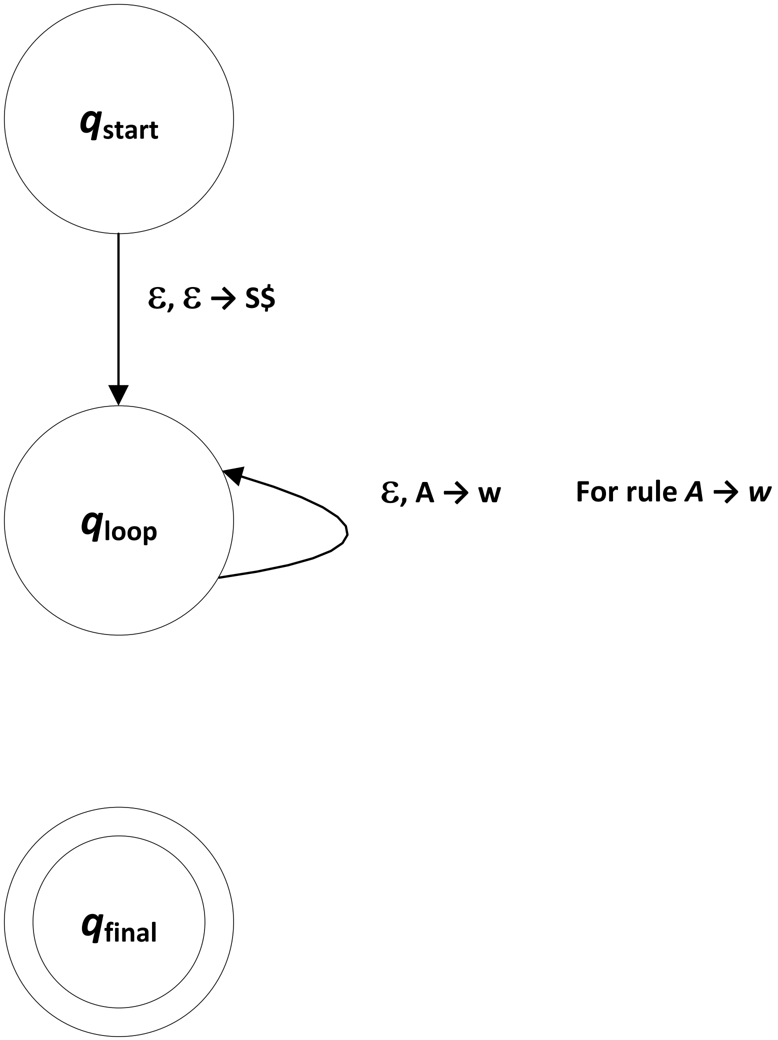


CONVERTING CFG TO PDA

Hence, if the symbol at the top of the stack is a variable A, the PDA moves from state q_{loop} to q_{loop} and pushes the right-hand side string (w) of a rule for variable A.

As mentioned earlier, if there are several rules for variable A, the nondeterministic nature of the PDA will allow it to try out all possible rules.

So there must be a transition edge of this type for each rule for each variable.



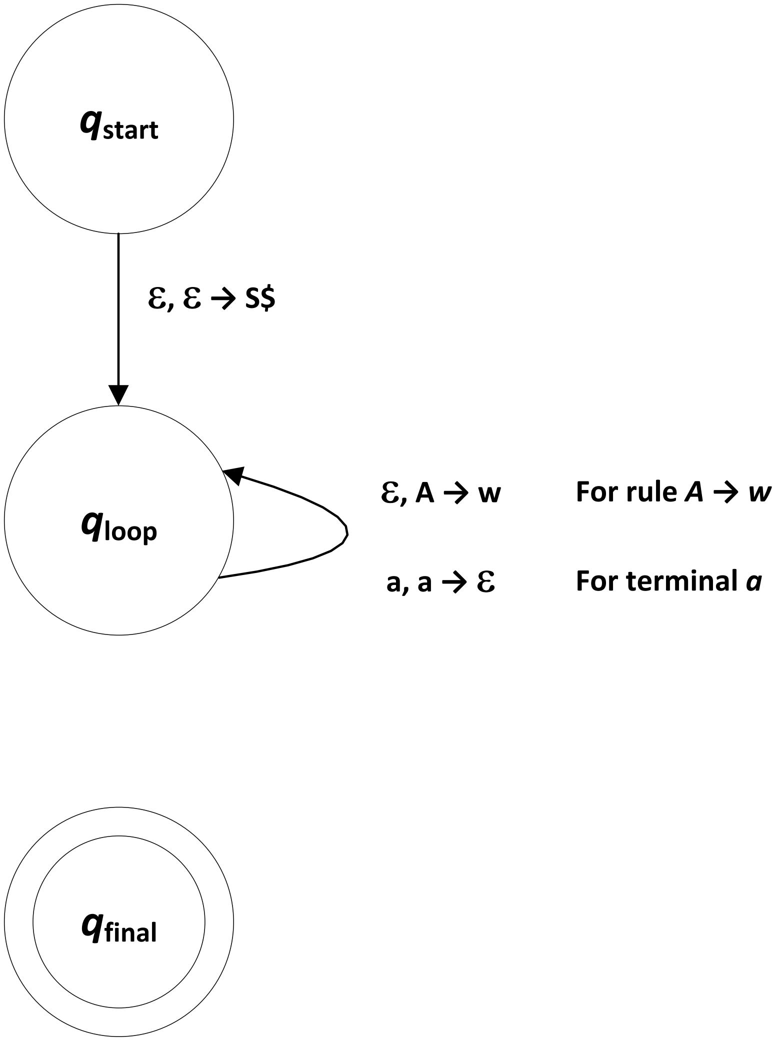
CONVERTING CFG TO PDA

Case 2:

If the top of the stack symbol is a terminal *a*, there should be a transition edge with label

$$a, a \rightarrow \varepsilon$$

Hence, if the symbol at the top of the stack is a terminal *a*, the PDA must pop this terminal and compare it to next incoming input symbol



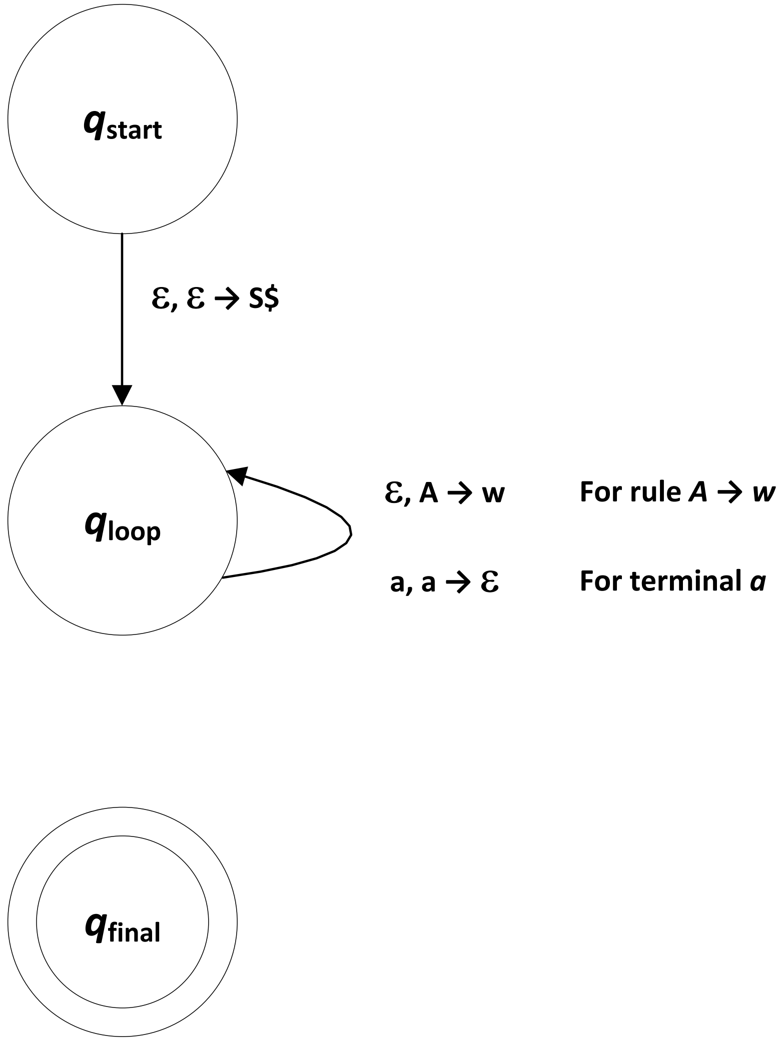
CONVERTING CFG TO PDA

If the input symbol is also *a* (they match), then the PDA simply moves from q_{loop} to q_{loop} without pushing anything onto the stack.

Since there is a match, processing therefore continues.

If the terminal at the top of the stack does not match the incoming input symbol, the PDA cannot go anywhere and processing then dies. The input string is therefore rejected.

Take note that there must be a transition edge of this type for each terminal in the grammar.



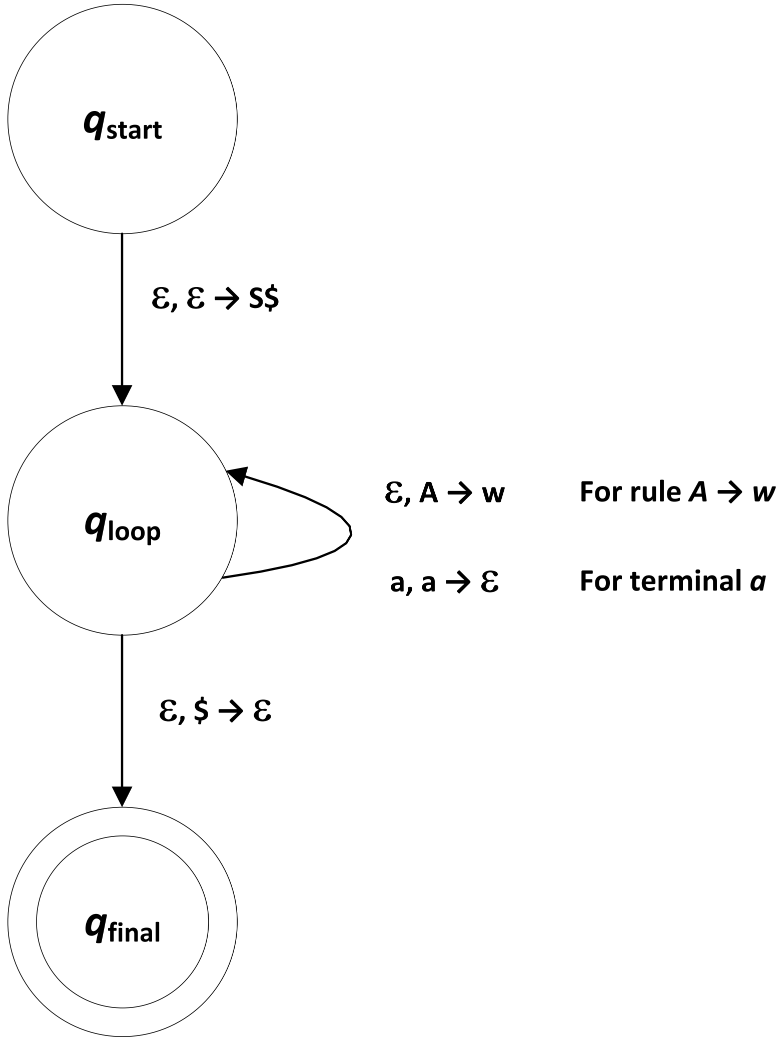
CONVERTING CFG TO PDA

Case 3:

If the top of the stack symbol is the stack empty symbol *$*, there should be a transition edge with label

$$\varepsilon, \$ \rightarrow \varepsilon$$

If the symbol at the top of the stack is *$* (the stack is empty) and there are no more input symbols, the PDA moves from q_{loop} to q_{final}. The string is therefore accepted.



CONVERTING CFG TO PDA

- Example:

Convert the context-free grammar G_1: to its equivalent PDA:

$$S \rightarrow AB$$
$$A \rightarrow 0A \mid 0$$
$$B \rightarrow 0B1 \mid \varepsilon$$

CONVERTING CFG TO PDA

$S \rightarrow AB$

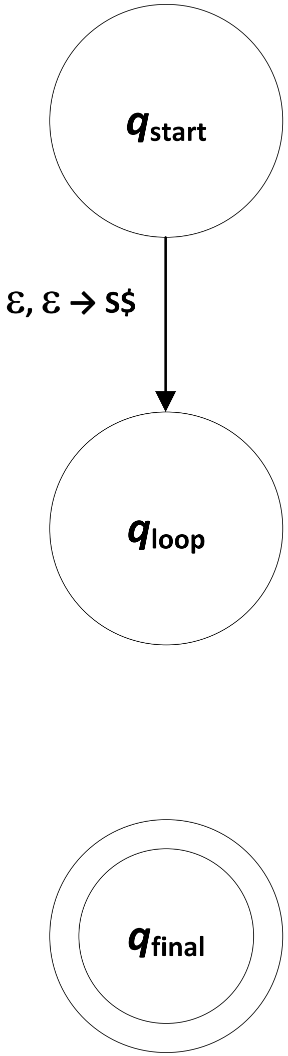
$A \rightarrow 0A \mid 0$

$B \rightarrow 0B1 \mid \varepsilon$

1. From q_{start} to q_{loop}:

$$\varepsilon, \varepsilon \rightarrow S\$$$

This allows the PDA to move right away from q_{start} to q_{loop} and push the string *S$* onto the stack.

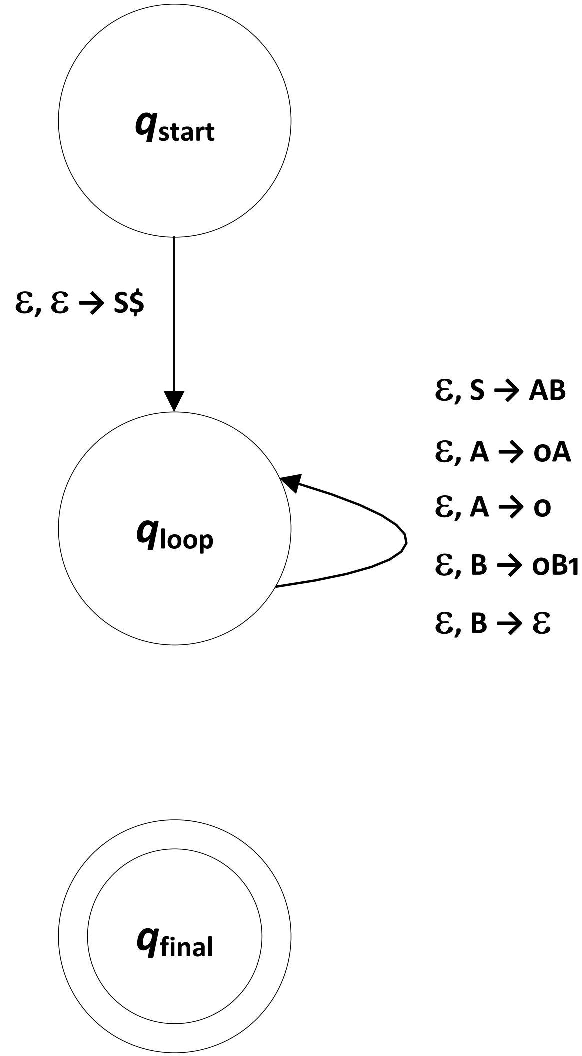


CONVERTING CFG TO PDA

$S \rightarrow AB$
$A \rightarrow 0A \mid 0$
$B \rightarrow 0B1 \mid \varepsilon$

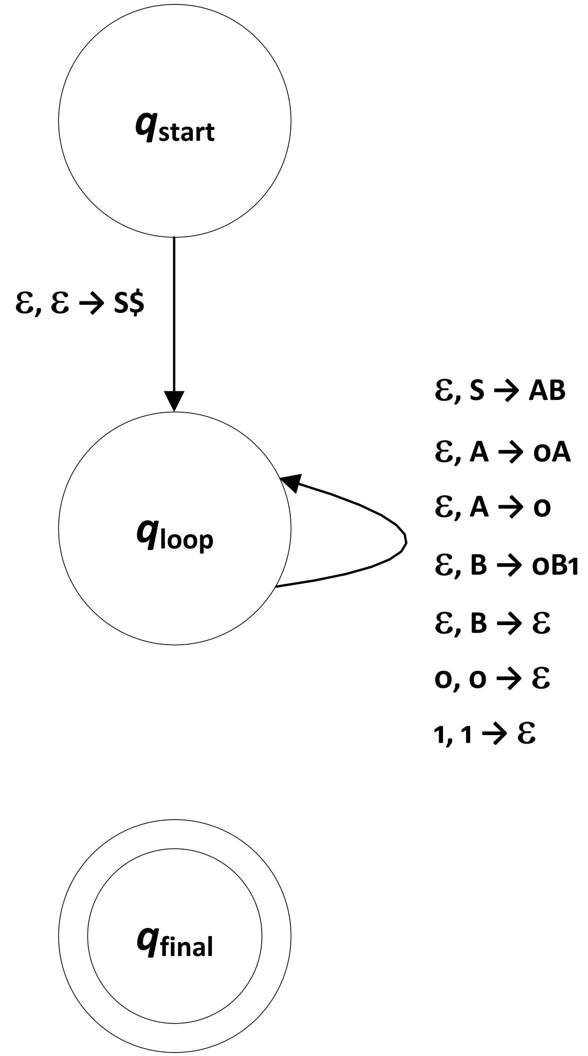
2. From q_{loop} to q_{loop}:

$\varepsilon, S \rightarrow AB$	for the rule $S \rightarrow AB$
$\varepsilon, A \rightarrow 0A$	for the rule $A \rightarrow 0A$
$\varepsilon, A \rightarrow 0$	for the rule $A \rightarrow 0$
$\varepsilon, B \rightarrow 0B1$	for the rule $B \rightarrow 0B1$
$\varepsilon, B \rightarrow \varepsilon$	for the rule $B \rightarrow \varepsilon$



CONVERTING CFG TO PDA

$S \rightarrow AB$
$A \rightarrow 0A \mid 0$
$B \rightarrow 0B1 \mid \varepsilon$



3. From q_{loop} to q_{loop}:

$0, 0 \rightarrow \varepsilon$ for the terminal 0
$1, 1 \rightarrow \varepsilon$ for the terminal 1

CONVERTING CFG TO PDA

$S \rightarrow AB$

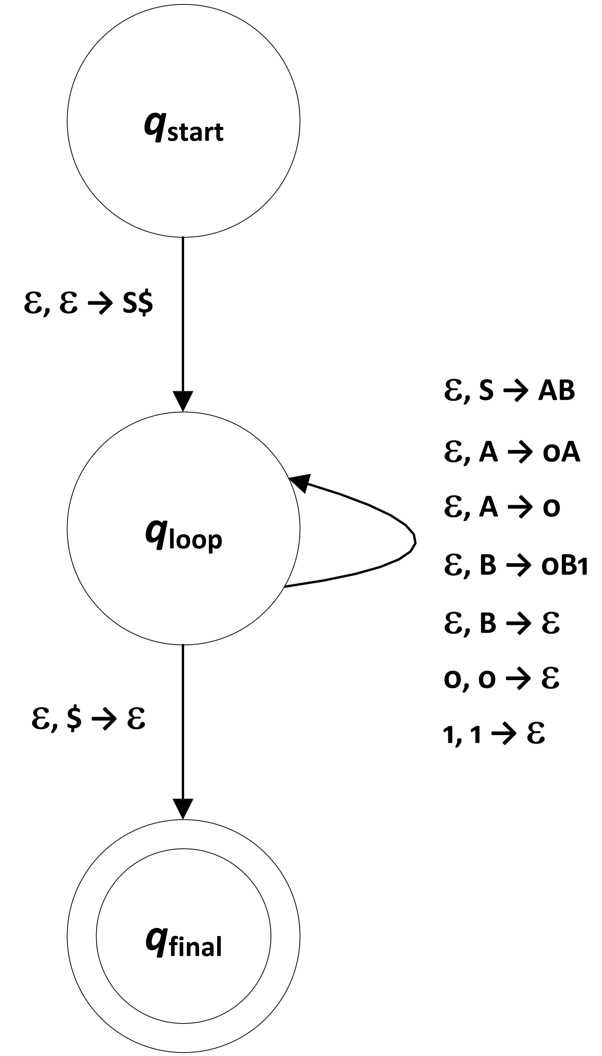
$A \rightarrow 0A \mid 0$

$B \rightarrow 0B1 \mid \varepsilon$

4. From q_{loop} to q_{final}:

$$\varepsilon, \$ \rightarrow \varepsilon$$

If the symbol at the top of the stack is *$* (stack is empty) and there are no more input symbols, the PDA moves to the final state q_{final} and the input string is accepted.



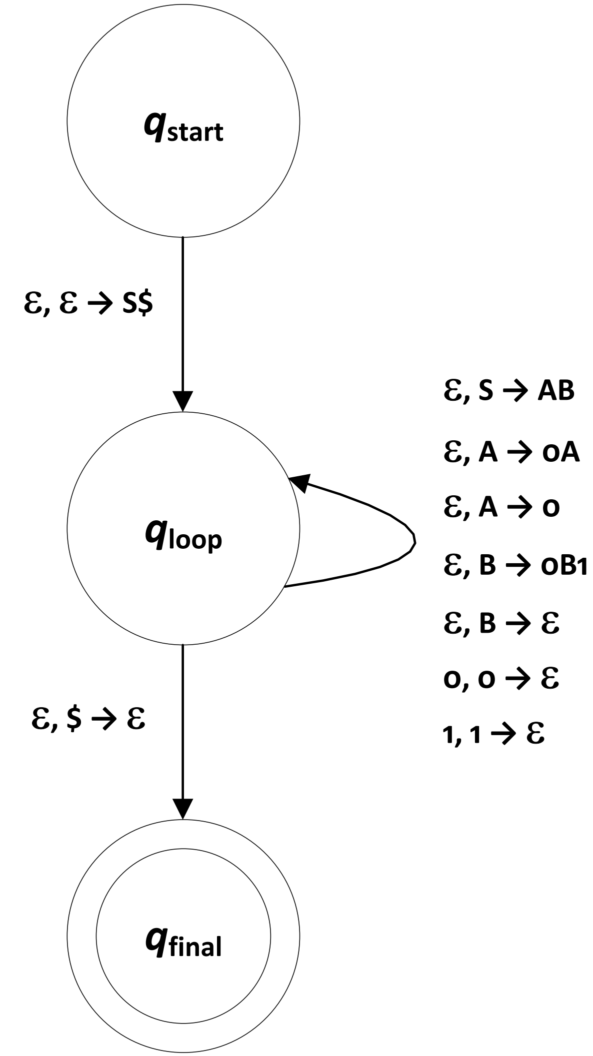
CONVERTING CFG TO PDA

- However, the PDA presented still has state transitions wherein an entire string is pushed onto the stack instead of just one symbol.

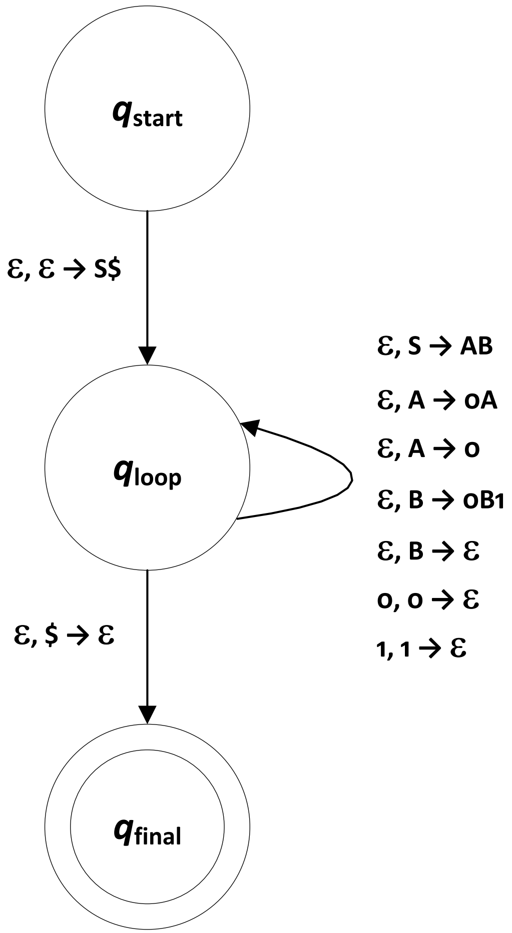
- The transitions being referred to are the ones with the labels

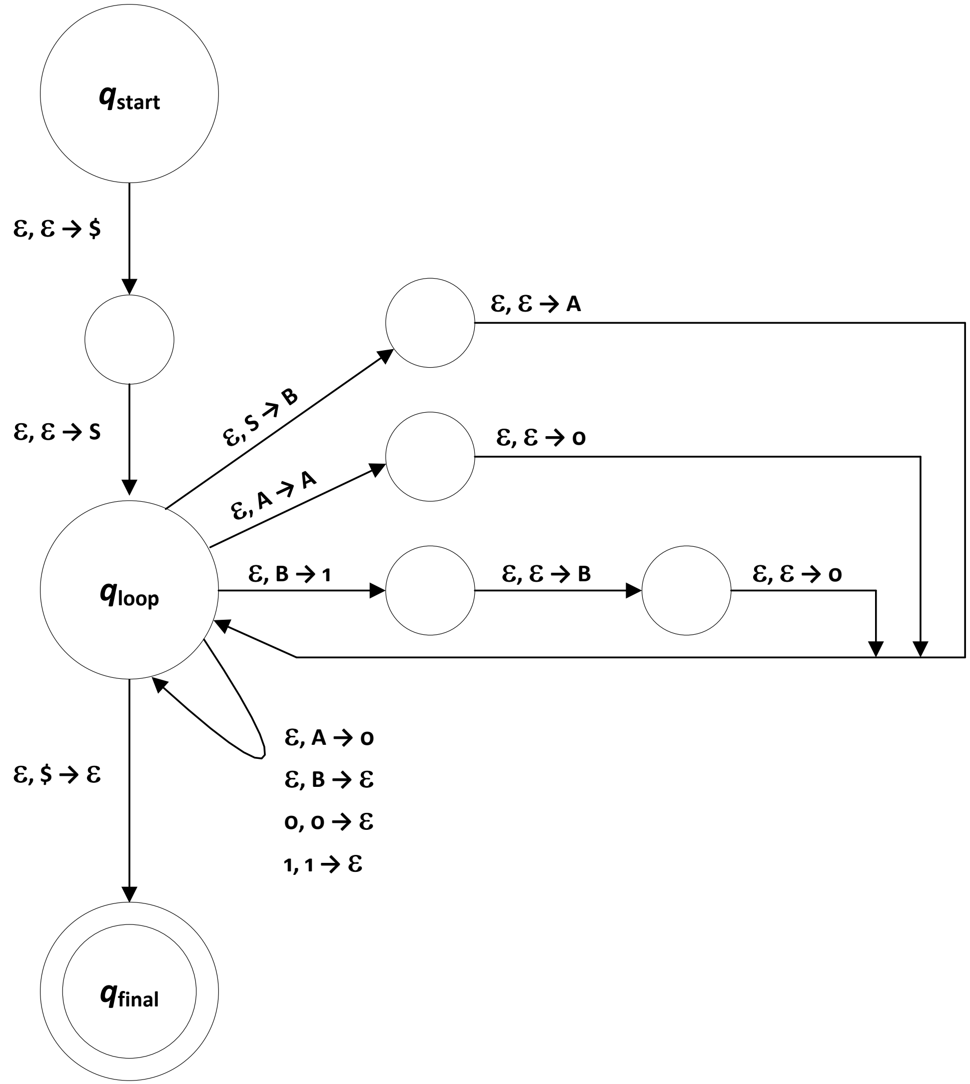
$$\varepsilon, \varepsilon \rightarrow S\$$$
$$\varepsilon, S \rightarrow AB$$
$$\varepsilon, A \rightarrow 0A$$
$$\varepsilon, B \rightarrow 0B1$$

- Therefore, PDA P_1 must be revised so that intermediate states are used to ensure that only one symbol is pushed onto the stack per state transition.



CONVERTING CFG TO PDA





CONVERTING CFG TO PDA

- Example:

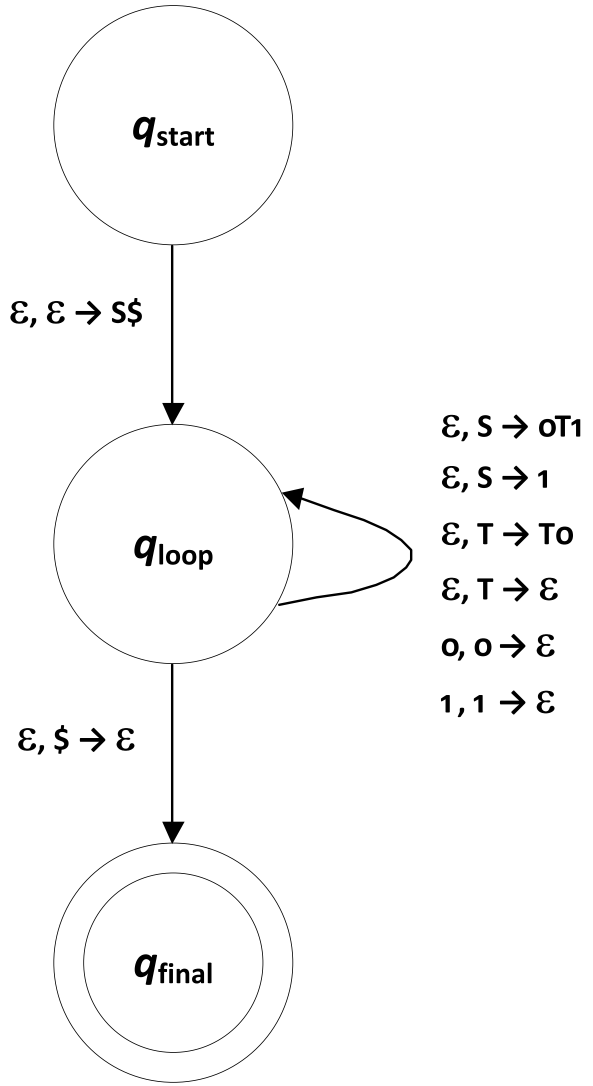
  Convert the following context-free grammar G_2 to its equivalent PDA P_2:

$$S \rightarrow 0T1 \mid 1$$
$$T \rightarrow T0 \mid \varepsilon$$

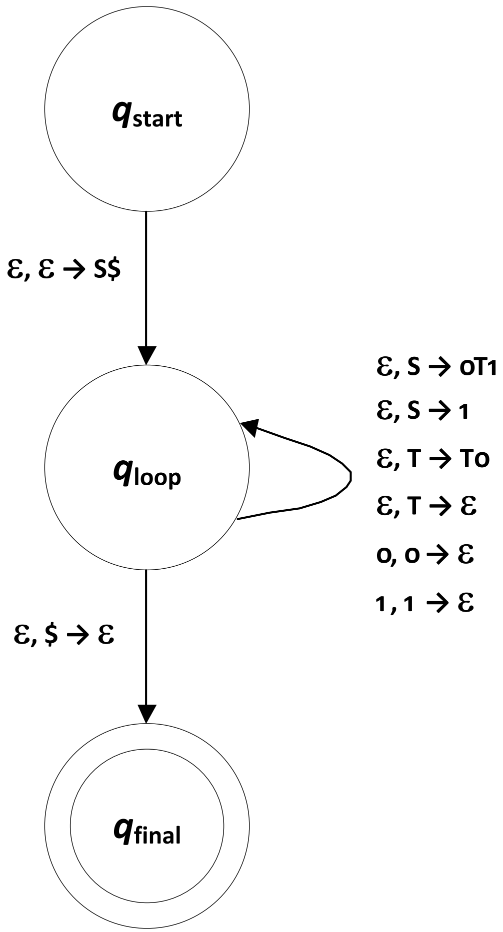
CONVERTING CFG TO PDA

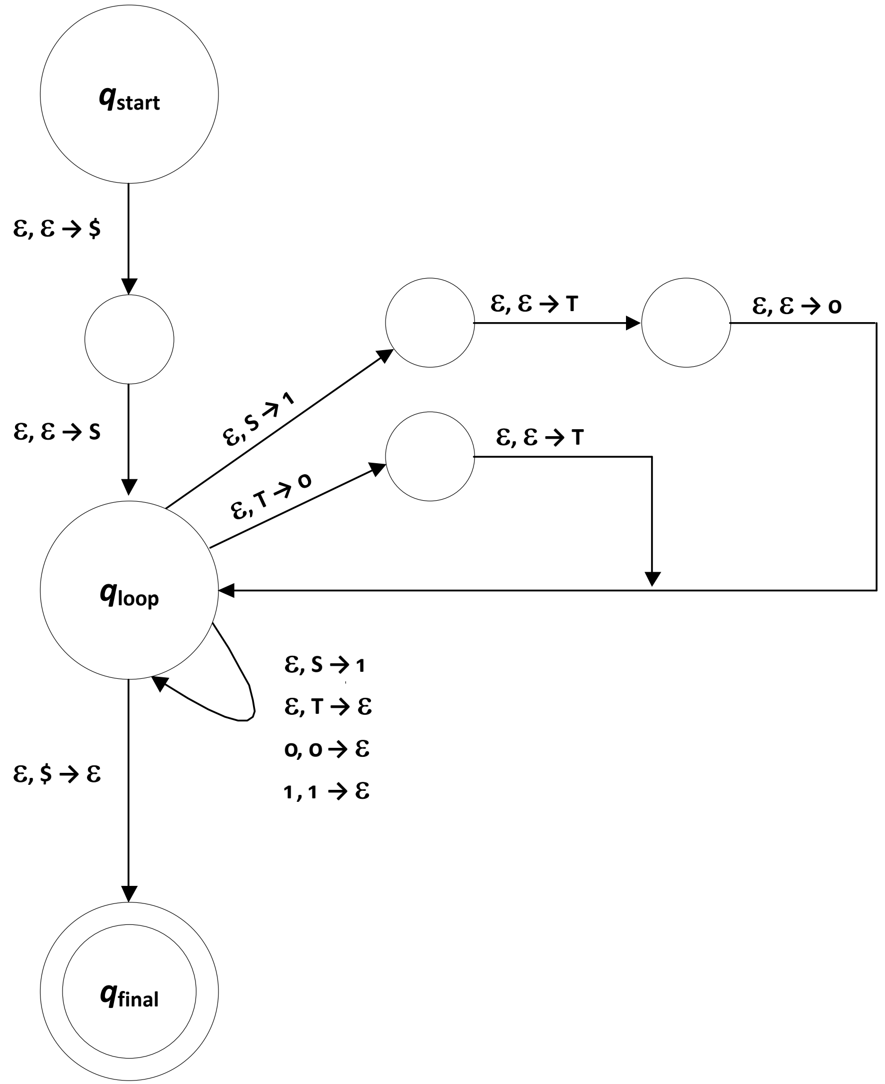
$S \rightarrow 0T1 \mid 1$

$T \rightarrow T0 \mid \varepsilon$



CONVERTING CFG TO PDA





EXERCISES

- Convert the following context-free grammars to PDAs:

1. Grammar G_3

   $S \rightarrow AB$
   $A \rightarrow aAb \mid \varepsilon$
   $B \rightarrow cB \mid \varepsilon$

2. Grammar G_4

   $S \rightarrow SS \mid (S) \mid \varepsilon$

3. Grammar G_5

   $S \rightarrow \varepsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$