Theory of Computing:

7. Pushdown Automata

Professor Imed Bouchrika
National School of Artificial Intelligence
imed.bouchrika@ensia.edu.dz

Outline :

- Revision:
  - Context-Free Languages
  - Chomsky normal form
- Pushdown Automata
- Constructing PDA
- Notations and Formalism
- Examples
- Converting CFG <---> PDA

Context-Free Languages

- Context-free languages are those that can be **generated** by context-free grammar.
- Example : Language L = { 0^n1^n | n >= 0 }

Context-Free Languages

- Context-free languages are those that can be **generated** by context-free grammar.
- Example : Language L = { 0^n1^n | n >= 0 }
    - $S \rightarrow 0S1 \mid e$

Context-Free Languages

- Context-free languages are those that can be **generated** by context-free grammar.
- Example : Language L = { palindromes }

Context-Free Languages

- Context-free languages are those that can be **generated** by context-free grammar.
- Example : Language L = { palindromes }
  - $S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$

Context-Free Languages

- Context-free languages are those that can be **generated** by context-free grammar.
- Example : Language L = { even-length palindromes }

Context-Free Languages

- Context-free languages are those that can be **generated** by context-free grammar.
- Example : Language L = { even-length palindromes }
  - $S \rightarrow aSa \mid bSb \mid \varepsilon$

Context-Free Languages

- Context-free languages are those that can be **generated** by context-free grammar.
- Example : Language L = { odd-length palindromes }

Context-Free Languages

- Context-free languages are those that can be **generated** by context-free grammar.
- Example : Language L = { odd-length palindromes }
  - $S \rightarrow aSa \mid bSb \mid a \mid b$

Context-Free Languages

- Context-free languages are those that can be **generated** by context-free grammar.
- Example : Language L = { 0^n1^n | n >= 0 }
    - $S \rightarrow 0S1 \mid e$

Ambiguous Context Free Grammar

Ambiguous Context Free Grammar

- Converting the following to Chomsky normal form:
  - $A \rightarrow BAB \mid B \mid \varepsilon$
  - $B \rightarrow 00 \mid \varepsilon$

Ambiguous Context Free Grammar

- Converting the following to Chomsky normal form:
  - $A \rightarrow BAB \mid B \mid \varepsilon$
  - $B \rightarrow 00 \mid \varepsilon$

  - **$S \rightarrow A$**
  - $A \rightarrow BAB \mid B \mid \varepsilon$
  - $B \rightarrow 00 \mid \varepsilon$

Ambiguous Context Free Grammar

- Converting the following to Chomsky normal form:

  - $S \rightarrow A$
  - $A \rightarrow BAB \mid B \mid \boldsymbol{\varepsilon}$
  - $B \rightarrow 00 \mid \boldsymbol{\varepsilon}$

Ambiguous Context Free Grammar

- Converting the following to Chomsky normal form:

  - S→A
  - A → BAB | B | ε
  - B → 00 | **ε**

  - S→A
  - **A → BAB | B | ε | BA | AB | A**
  - B → 00

Ambiguous Context Free Grammar

- Converting the following to Chomsky normal form:

  - S→A
  - **A → BAB | B | ε | BA | AB | A**
  - B → 00

  - S→A | **ε**
  - A → BAB | B | BA | AB | A | **BB**
  - B → 00

Ambiguous Context Free Grammar

- Converting the following to Chomsky normal form:

  - $S \rightarrow A \mid \boldsymbol{\varepsilon}$
  - $A \rightarrow BAB \mid B \mid BA \mid AB \mid \mathbf{A} \mid BB$
  - $B \rightarrow 00$

  - $S \rightarrow A \mid \boldsymbol{\varepsilon}$
  - $A \rightarrow BAB \mid B \mid BA \mid AB \mid BB$
  - $B \rightarrow 00$

Ambiguous Context Free Grammar

- Converting the following to Chomsky normal form:

    - S→A | **ε**
    - A → BAB | **B** | BA | AB | BB
    - B → 00

    - S→A | **ε**
    - A → BAB | **00** | BA | AB | BB
    - B → 00

Ambiguous Context Free Grammar

- Converting the following to Chomsky normal form:

  - S→**A** | **ε**
  - A → BAB | 00 | BA | AB | BB
  - B → 00

  - S→**BAB | 00 | BA | AB | BB** | **ε**
  - A → BAB | 00 | BA | AB | BB
  - B → 00

Ambiguous Context Free Grammar

- Converting the following to Chomsky normal form:

  - S→A | ε
  - A → BAB | 00 | BA | AB | BB
  - **B → 00**

  - S→BAB | **UU** | BA | AB | BB | ε
  - A → BAB | **UU** | BA | AB | BB
  - B → **UU**
  - **U →0**

Ambiguous Context Free Grammar

- Converting the following to Chomsky normal form:

  - $S \rightarrow \mathbf{BAB} \mid 00 \mid BA \mid AB \mid BB \mid \varepsilon$
  - $A \rightarrow BAB \mid 00 \mid BA \mid AB \mid BB$
  - $B \rightarrow UU$
  - $U \rightarrow 0$

  - $S \rightarrow \mathbf{BA_b} \mid UU \mid BA \mid AB \mid BB \mid \varepsilon$
  - $A \rightarrow BAB \mid UU \mid BA \mid AB \mid BB$
  - $B \rightarrow UU$
  - $U \rightarrow 0$
  - $\mathbf{A_b \rightarrow AB}$

Context-Free Languages

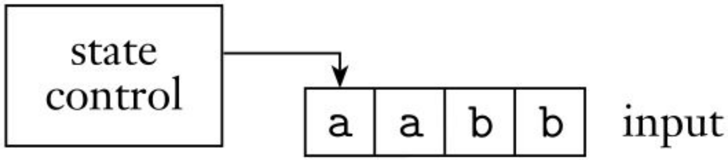
- As with regular languages, there are various approaches to describe the language :
    - Build Automata that accepts or **recognize**s a string from the language
    - Design Regular expressions that **describe** the language
    - Write the regular grammar to **generate** the language

Context-Free Languages

- For context-free languages :
  - Context-free grammar : to **generate** language.
  - Can we design an automaton to **recognize** a word from the language ?

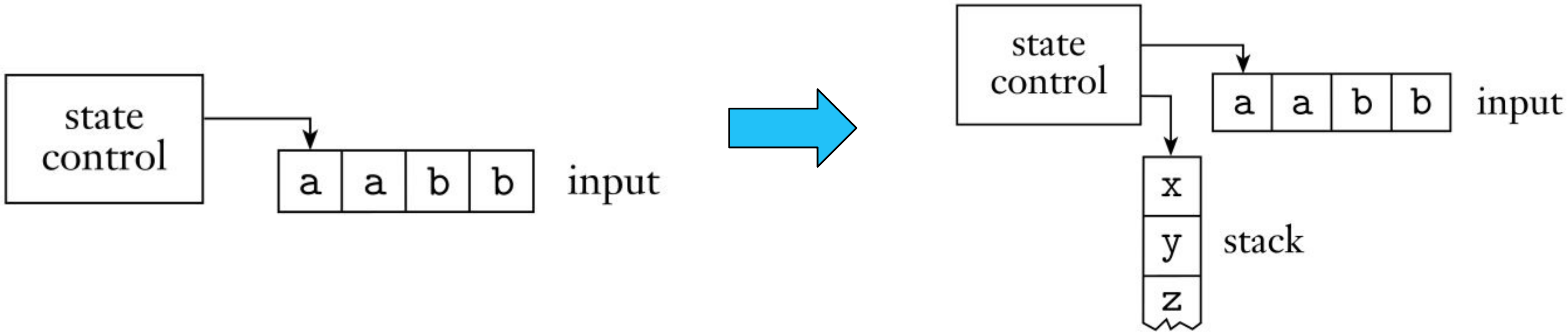
Pushdown Automata

- Can NFA represent a context-free language : ?
- **No :** We need a machine with some extra memory



Pushdown Automata

- Can NFA represent a context-free language : ?
- **No :** We need a machine with some extra memory : **Stack**



Pushdown Automata

- Pushdown Automata (PDA) : can be considered as NFA with a **stack**
- The stack stores information on the **last-in first-out** principle.
    - Items are added on top by **pushing**;
    - items are removed from the top by **popping**
- **Only the top** of the stack is visible at any point in time.

Pushdown Automata

- A pushdown automaton (PDA) has a fixed set of states (like FA), but it also has **one** stack with (theoretically) **infinite** storage.
- When symbol is read, depending on :
  1. *State of automaton*
  2. *Symbol on top of stack*
  3. *Symbol read,*

Pushdown Automata

- A pushdown automaton (PDA) has a fixed set of states (like FA), but it also has one stack with (theoretically) **infinite** storage.
- When symbol is read, depending on :
  1. *State of automaton*
  2. *Symbol on top of stack*
  3. *Symbol read,*

The automaton would:

- *Updates its state*
- *(optionally)* ***pop****s or* ***push****es a symbol.*

Pushdown Automata

- A pushdown automaton (PDA) has a fixed set of states (like FA), but it also has one stack with (theoretically) **infinite** storage.
- When symbol is read, depending on :
  1. *State of automaton*
  2. *Symbol on top of stack*
  3. *Symbol read,*

The automaton would:

- *Updates its state*
- *(optionally)* ***pop****s or* ***push****es a symbol.*

The automaton may also pop or push without reading input.

Constructing Pushdown Automaton

- **Important notes:**
  - The stack is recommended to be initialized with a special symbol or marker either the **$** or $\mathbf{Z_0}$ symbols
  - The special symbol is used to indicate the bottom of the stack.
  - There are different notations and definitions used for PDA depending on the textbook being used :
    - Sipser's Book :
      - does not have a start stack symbol
      - does not allow transitions to push multiple symbols onto the stack.

Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }

Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = $\{a^n b^n\}$
- **Informal English Description :**

  *Read symbols from the input. As each **a** is read, push it onto the stack.*

  *As soon as **b**s are seen, pop an **a** off the stack for each **b** read.*

  *If reading the input is finished exactly when the stack becomes empty of **a**s, accept the input.*

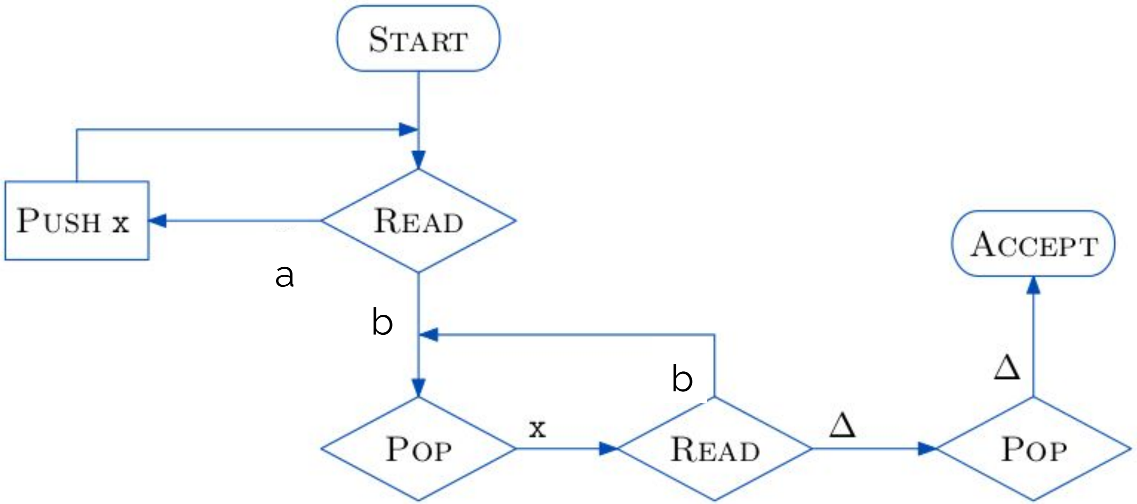
  *If the stack becomes empty while **b**s remain or if the **b**s are finished while the stack still contains **a**s or if any **a**s appear in the input following **b**s, **reject the input***

Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = $\{a^n b^n\}$
- **Design the informal algorithm**
    1. Initialize the Stack with a special marker
    2. while next input character is **a** do
        - push **a**
    3. while next input character is **b** do
        - pop **a**
    4. The special marker is on top of the stack.

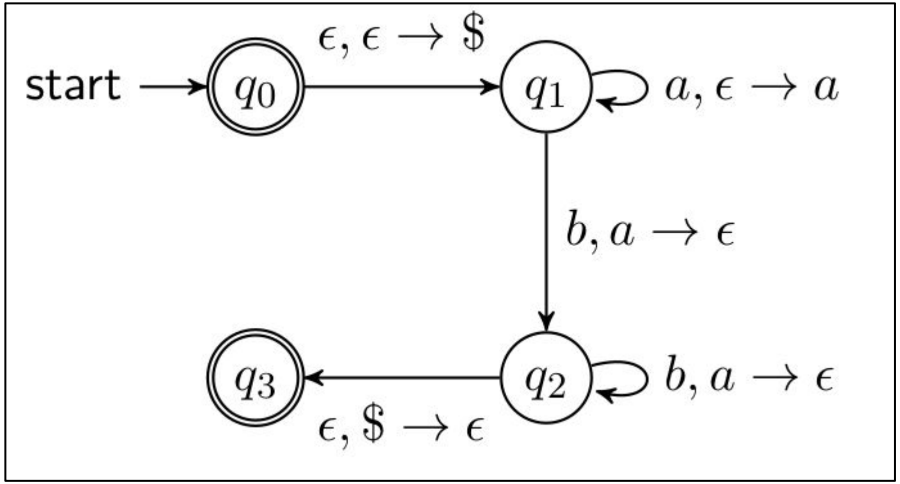
Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }
- **Flowchart** can be used to simplify the concept



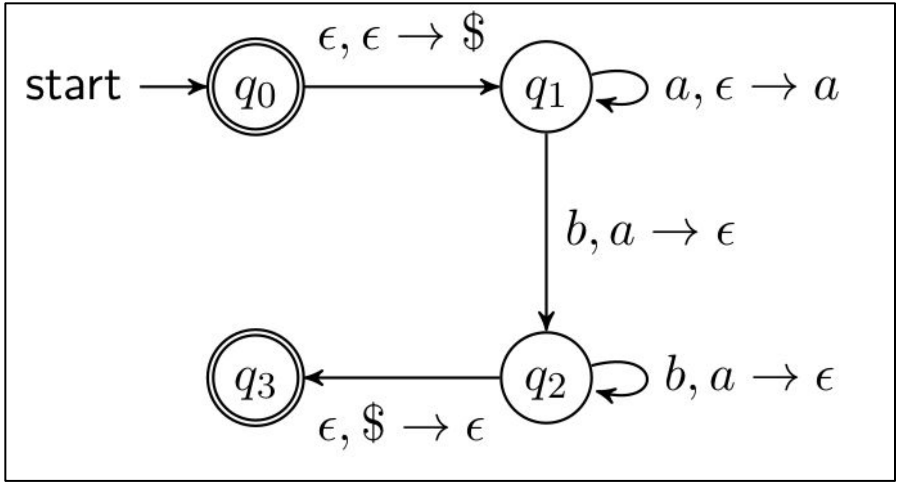
Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }
- State Diagram



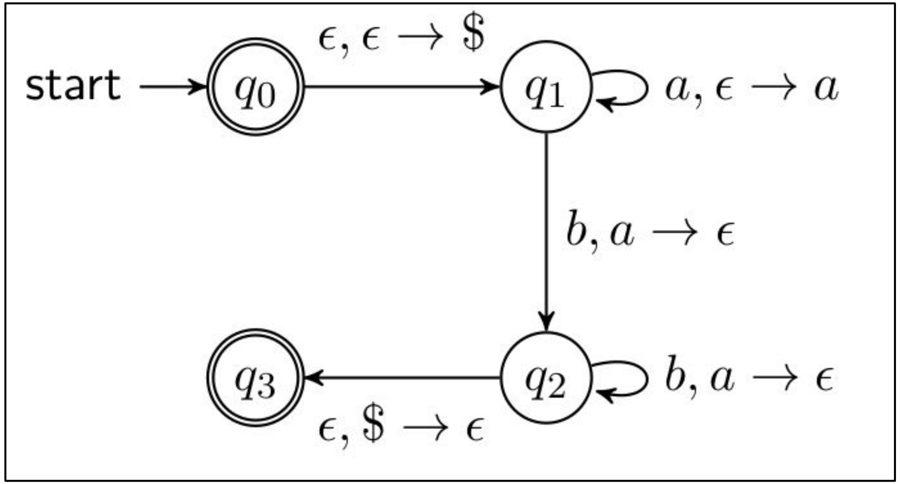
Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }
- State Diagram
    - **States : q0, q1, q2 , q3**



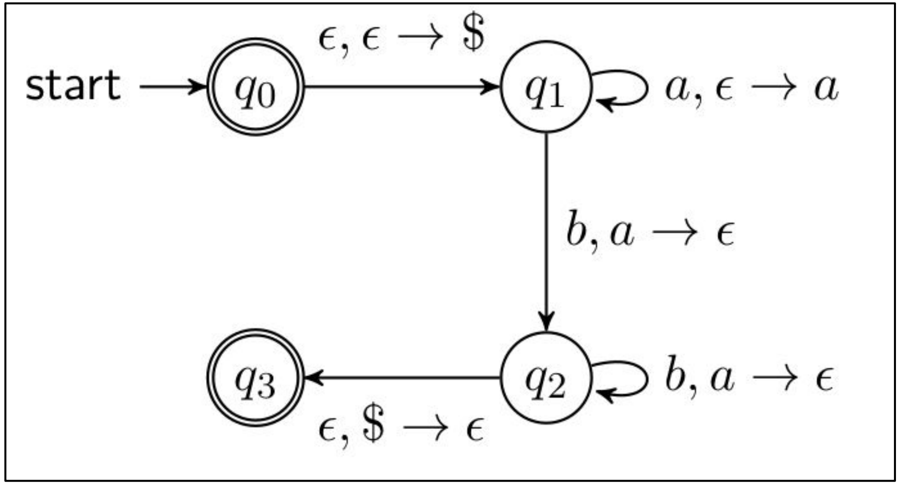
Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }
- State Diagram
  - **Start State : q0**



Constructing Pushdown Automaton

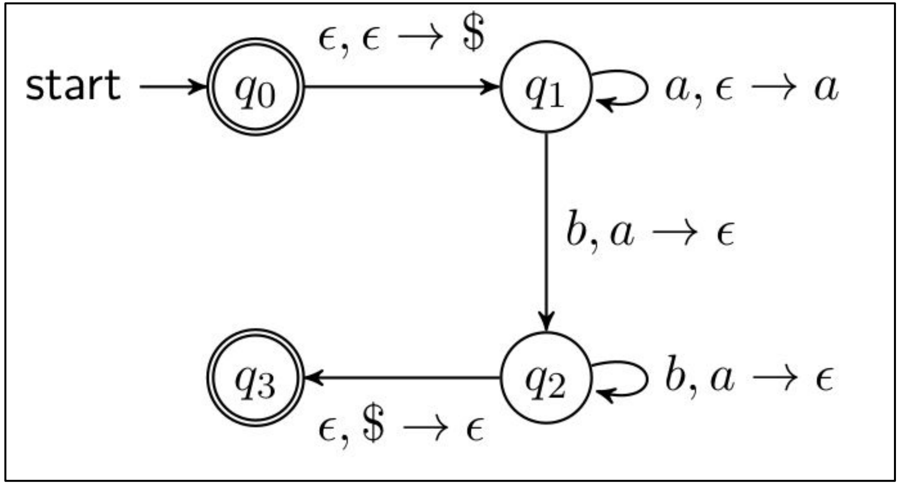
- Let's Construct a PDA that accepts all strings from the language L = $\{a^n b^n\}$
- State Diagram
  - **Accepting States : q0 , q3**



Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = $\{a^n b^n\}$
- State Diagram
  - **Transitions :**

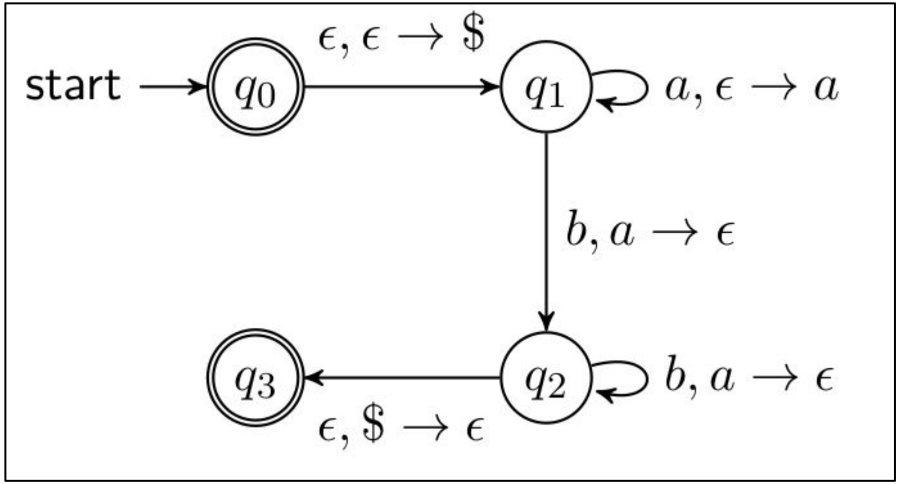
    A , B → C
  - *Means that when*
    - *You read symbol **A**,*
    - *Pop **B***
    - *Push **C***



Constructing Pushdown Automaton

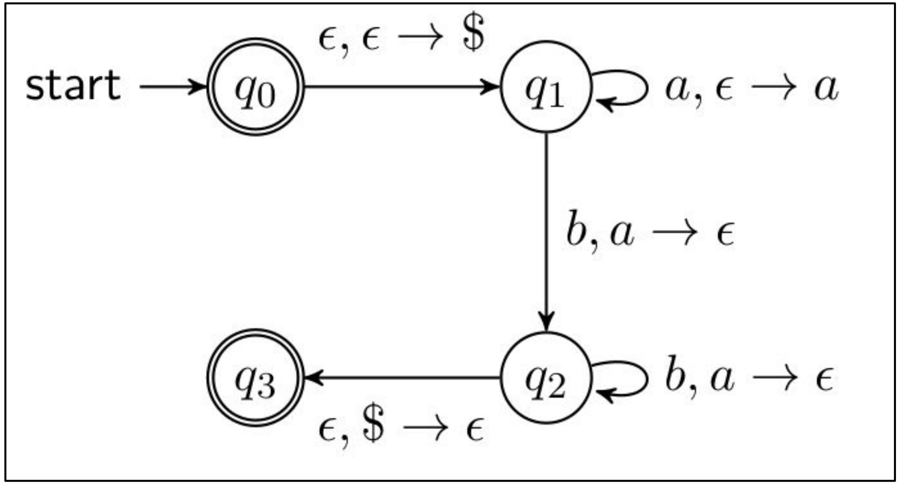
- Let's Construct a PDA that accepts all strings from the language L = $\{a^n b^n\}$
- State Diagram
  - **Transitions :**

    A , B → C
  - *Special Cases*
    - **A , ε → B**
      - ***Push B***
    - **A , B → ε**
      - ***Pop B***



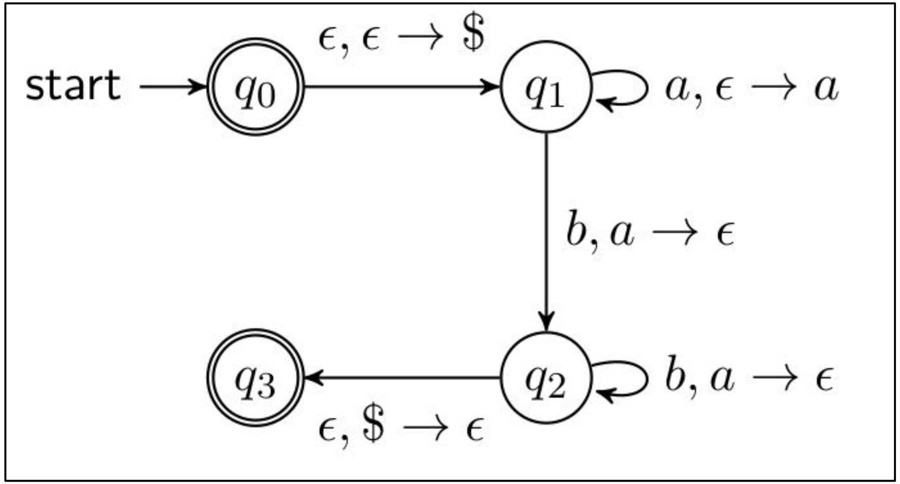
Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }
- State Diagram
  - **First Transition ε, ε → $:**
    - When nothing, place **$** into The top the of stack.
    - **$** is used to serve as a special marker



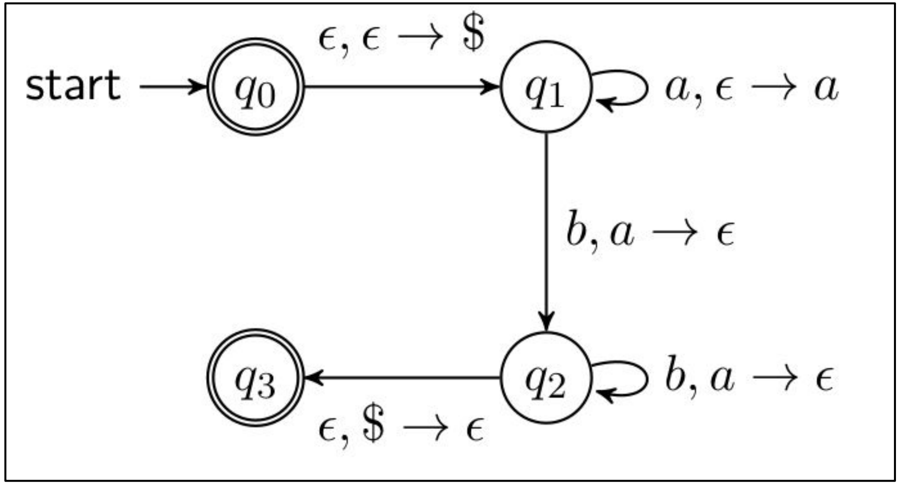
Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }
- State Diagram
    - **First Transition ε, ε → $:**
        - (q0, ε, ε) → (q1, $)



Constructing Pushdown Automaton

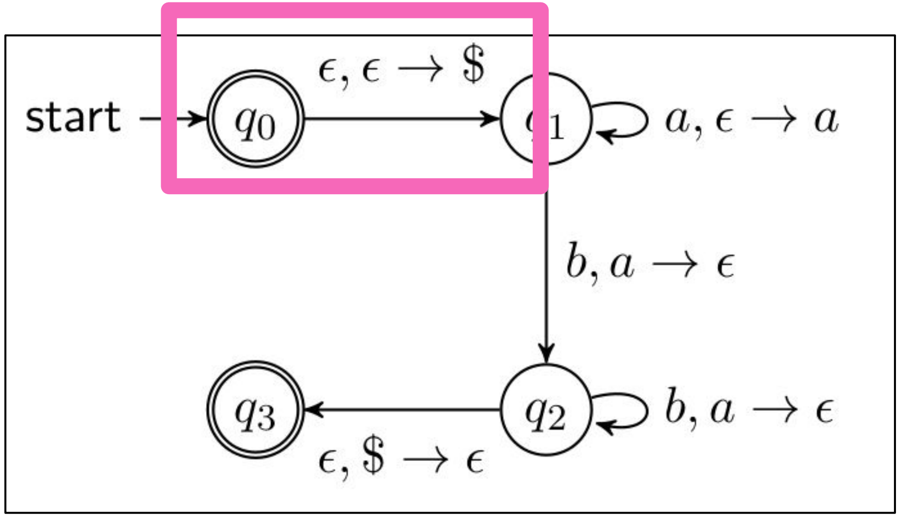
- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }
- Simulation : **aaabbb**



Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }
- Simulation : **aaabbb**

Step	State	Stack	Input	Action
1	q_0		$aaabbb$	push \$

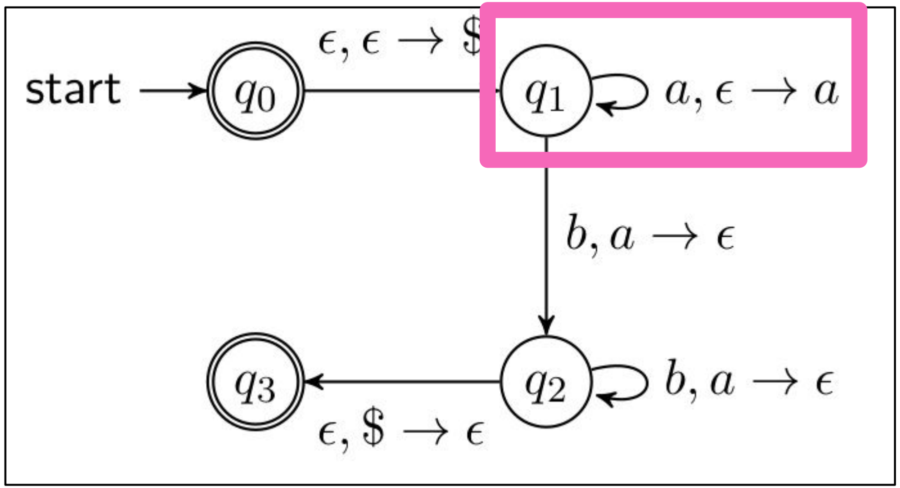


Taken from the slides of Pramod Ganapathi , State University of New York at Stony Brook

Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }
- Simulation : **aaabbb**

Step	State	Stack	Input	Action
1	q_0		$aaabbb$	push $\$$
2	q_1	$\$$	$aaabbb$	push a

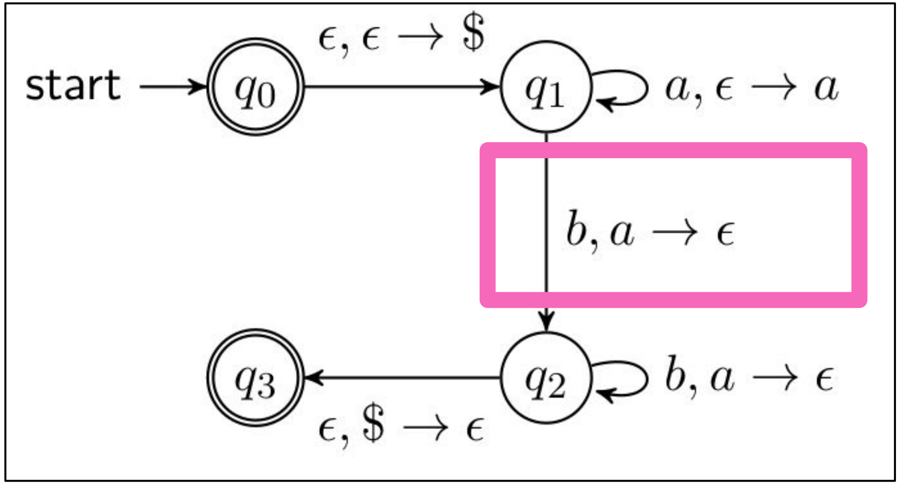


Taken from the slides of Pramod Ganapathi , State University of New York at Stony Brook

Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }
- Simulation : **aaabbb**

Step	State	Stack	Input	Action
1	q_0		$aaabbb$	push \$
2	q_1	\$	$aaabbb$	push a
3	q_1	$\$a$	$aabbb$	push a
4	q_1	$\$aa$	$abbb$	push a
5	q_1	$\$aaa$	bbb	pop a

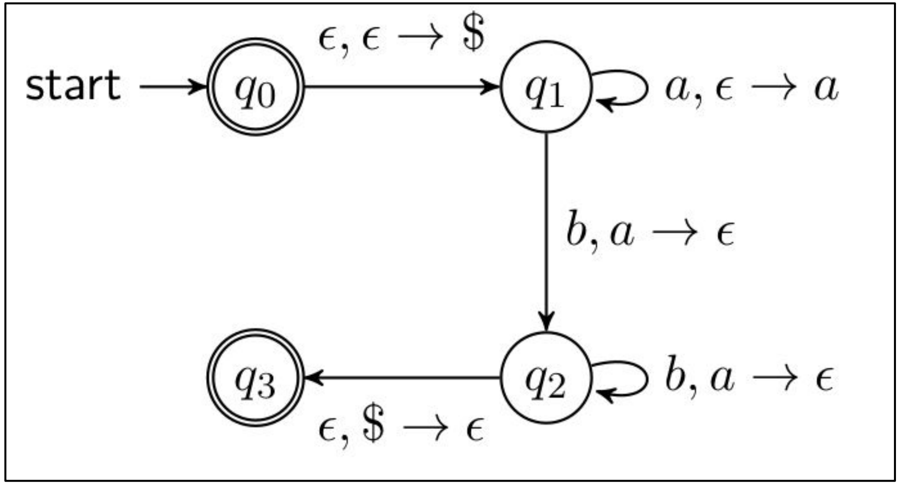


Taken from the slides of Pramod Ganapathi , State University of New York at Stony Brook

Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }
- Simulation : **aaabbb**

Step	State	Stack	Input	Action
1	q_0		$aaabbb$	push \$
2	q_1	\$	$aaabbb$	push a
3	q_1	\$$a$	$aabbb$	push a
4	q_1	\$$aa$	$abbb$	push a
5	q_1	\$$aaa$	bbb	pop a
6	q_2	\$$aa$	bb	pop a
7	q_2	\$$a$	b	pop a
8	q_2	\$		pop \$
9	q_3			accept

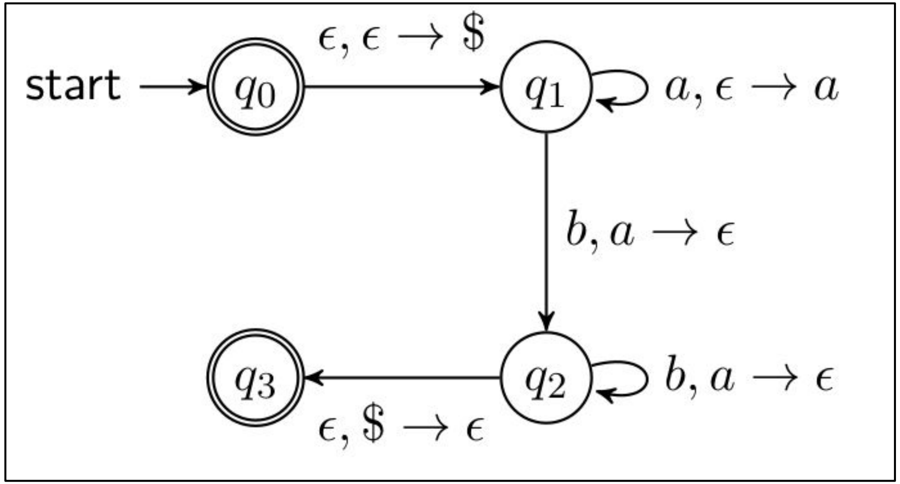


Taken from the slides of Pramod Ganapathi , State University of New York at Stony Brook

Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = $\{a^n b^n\}$
- Simulation : **aababb**

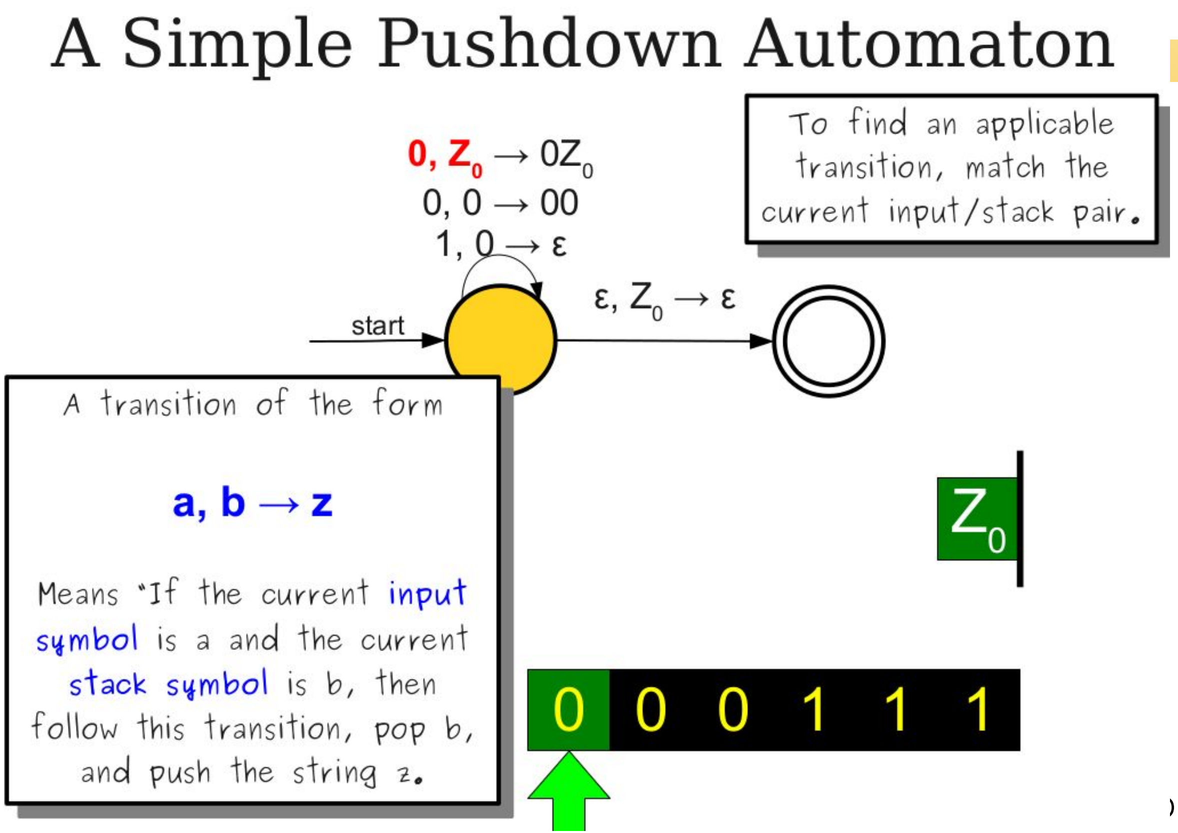
Step	State	Stack	Input	Action
1	q_0		$aababb$	push $\$$
2	q_1	$\$$	$aababb$	push a
3	q_1	$\$a$	$ababb$	push a
4	q_1	$\$aa$	$babb$	pop a
5	q_2	$\$a$	abb	crash
6	q_ϕ	$\$a$	bb	
7	q_ϕ	$\$a$	b	
8	q_ϕ	$\$a$		reject



Taken from the slides of Pramod Ganapathi , State University of New York at Stony Brook

Constructing Pushdown Automaton

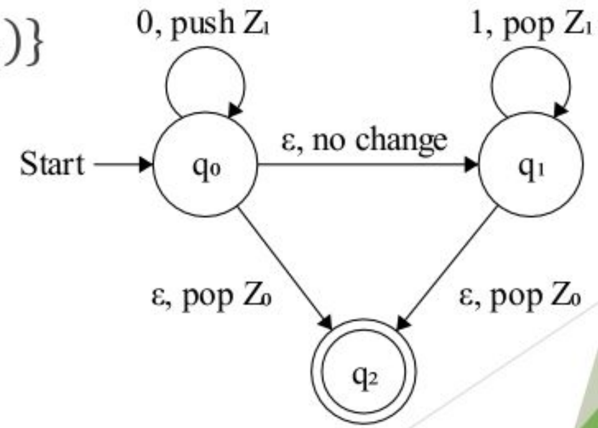
- Let's Construct a PDA tha
- Lecture notes from the University of Stanford



Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {$a^n b^n$ }
- Lecture notes from UNC

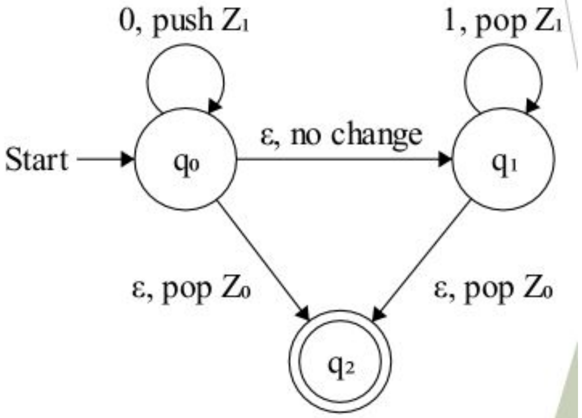
  - $M = (\{q_0, q_1, q_2\}, \{0, 1\}, \{Z_0, Z_1\}, \delta, q_0, Z_0, \{q_2\})$
  - $\delta(q_0, 0, Z_0) = \{(q_0, Z_1 Z_0)\}$
  - $\delta(q_0, 0, Z_1) = \{(q_0, Z_1 Z_1)\}$
  - $\delta(q_0, \varepsilon, Z_0) = \{(q_2, \varepsilon), (q_1, Z_0)\}$
  - $\delta(q_0, \varepsilon, Z_1) = \{(q_1, Z_1)\}$
  - $\delta(q_1, 1, Z_1) = \{(q_1, \varepsilon)\}$
  - $\delta(q_1, \varepsilon, Z_0) = \{(q_2, \varepsilon)\}$



Constructing Pushdown Automaton

- Let's Construct a PDA that accepts all strings from the language L = {a^n b^n }
- Lecture notes from
  UNC
- The symbol : ⊢
  means :
  Goes to

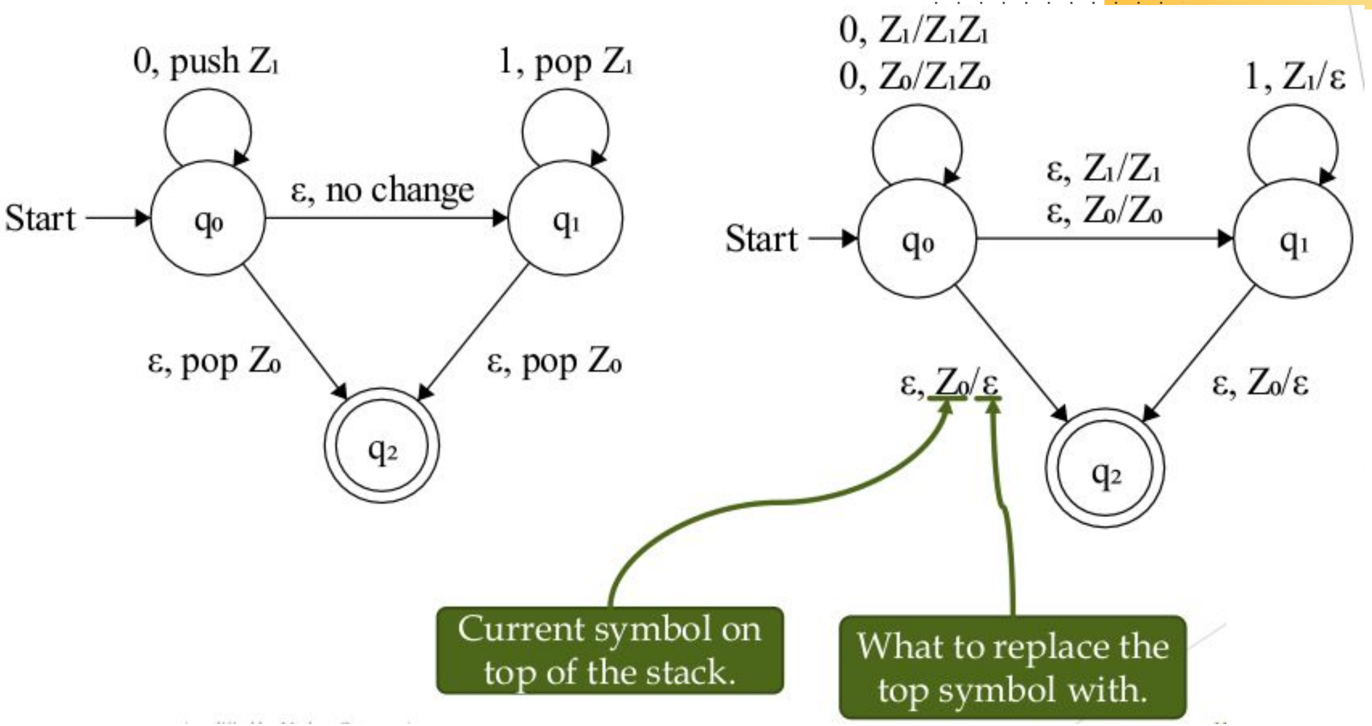
$$
\begin{aligned}
(q_0, 0011, Z_0) &\vdash (q_0, 011, Z_1 Z_0) \\
&\vdash (q_0, 11, Z_1 Z_1 Z_0) \\
&\vdash (q_1, 11, Z_1 Z_1 Z_0) \\
&\vdash (q_1, 1, Z_1 Z_0) \\
&\vdash (q_1, \varepsilon, Z_0) \\
&\vdash (q_2, \varepsilon, \varepsilon)
\end{aligned}
$$



Constructing Pushdown Automaton

- Let's Construct a
- Lecture notes

  From UNC



Constructing Pushdown Automaton

- Let’s Construct a PDA that accepts all strings from the language L = {a^n b^n }
- When to accept or reject ?

Formalism for Pushdown Automata

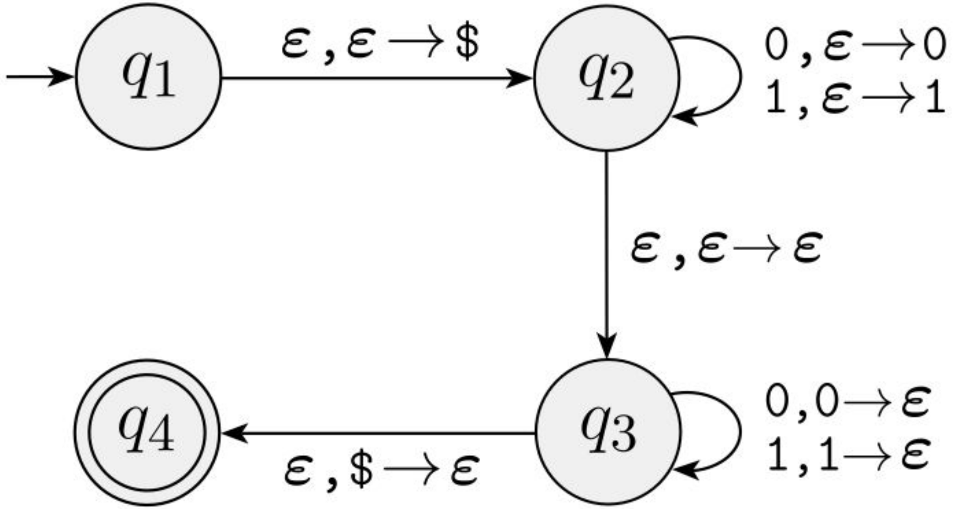
- A pushdown automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets such that:
  1. Q is the set of states,
  2. Σ is the input alphabet,
  3. Γ is the stack alphabet,
  4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$ is the transition function,
  5. $q_0 \in Q$ is the start state
  6. $F \subseteq Q$ is the set of accept states.

Examples for Creating Pushdown Automata

- Let's Construct a PDA that accepts all strings from the language
- $L = \{ww^R \mid w \in \{0,1\}^* \}$

Examples for Creating Pushdown Automata

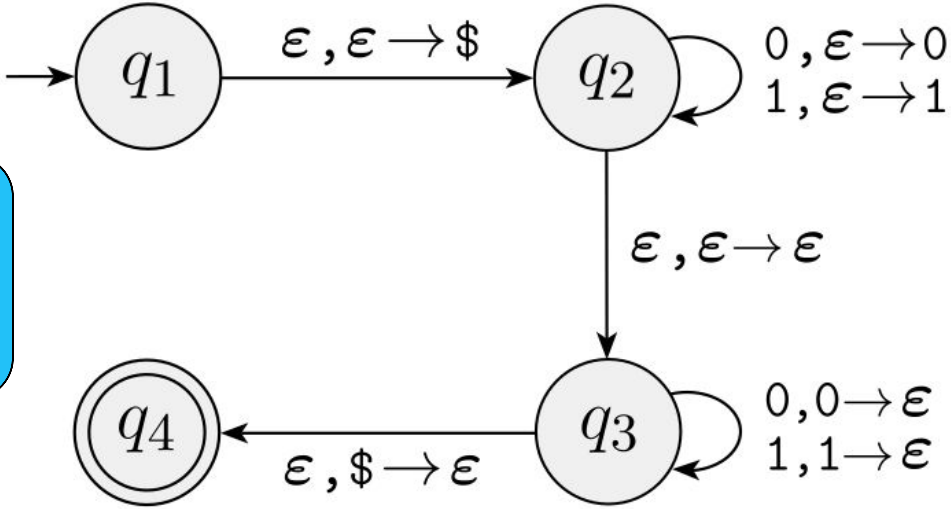
- Let's Construct a PDA that accepts all strings from the language
- L = $\{ww^R \mid w \in \{0,1\}^* \}$



Examples for Creating Pushdown Automata

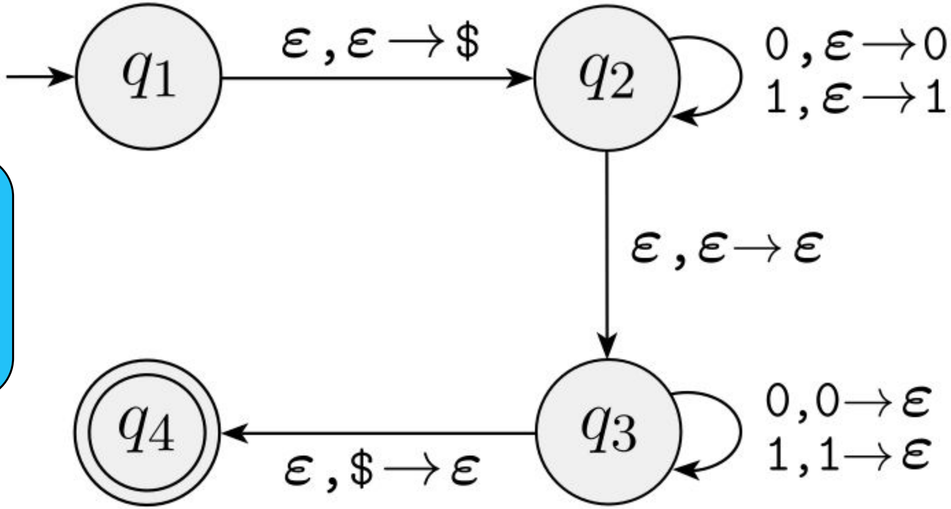
- Let's Construct a PDA that accepts all strings from the language
- $L = \{ww^R \mid w \in \{0,1\}^* \}$

Let's simulate recognizing the word :
1001



Examples for Creating Pushdown Automata

- Let's Construct a PDA that accepts all strings from the language
- L = $\{ww^R \mid w \in \{0,1\}^* \}$



Let's simulate recognizing the word :
0011

Examples for Creating Pushdown Automata

- Let's Construct a PDA that accepts all strings from the language
- $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

Examples for Creating Pushdown Automata

- Let's Construct a PDA that accepts all strings from the language
- $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

Algorithm:

1. While next input character is a do push a

2. Nondeterministically, guess whether a's =b's or a's =c's

Case 1 : a's=b's

- While next input is b do pop a

- While next input character is c do nothing

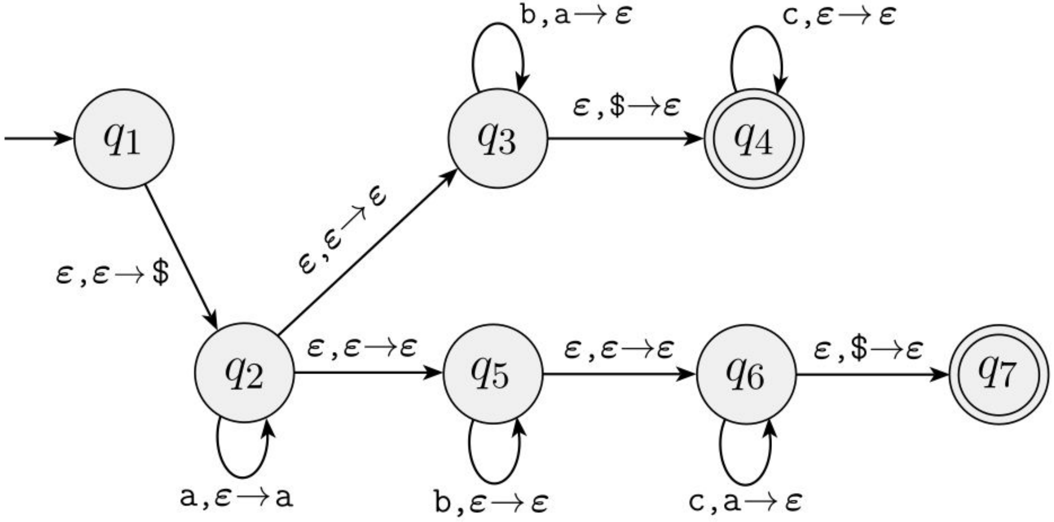
Case 2 : a's=c's

- While next input is b do nothing

- While next input character is c do pop a

Examples for Creating Pushdown Automata

- Let's Construct a PDA that accepts all strings from the language
- $L = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$



Examples for Creating Pushdown Automata

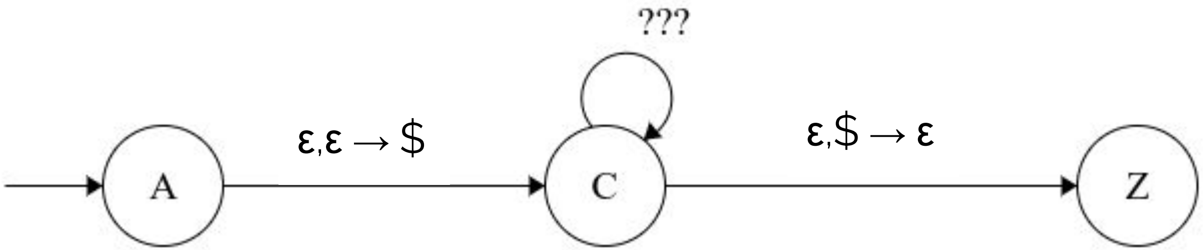
- Let's Construct a PDA that accepts all strings from the language
- $L = \{ww \mid w \in \{0,1\}^* \}$

Examples for Creating Pushdown Automata

- Let's Construct a PDA that accepts all strings from the language
- L = {w such that $w = w^R$ and w has an even length}

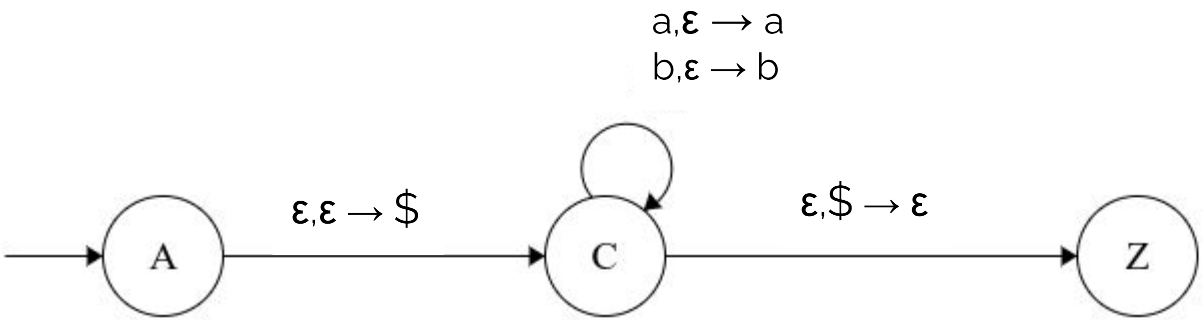
Examples for Creating Pushdown Automata

- Let's Construct a PDA that accepts all strings from the language
- L = {w such that w = w^R and w has an even length}



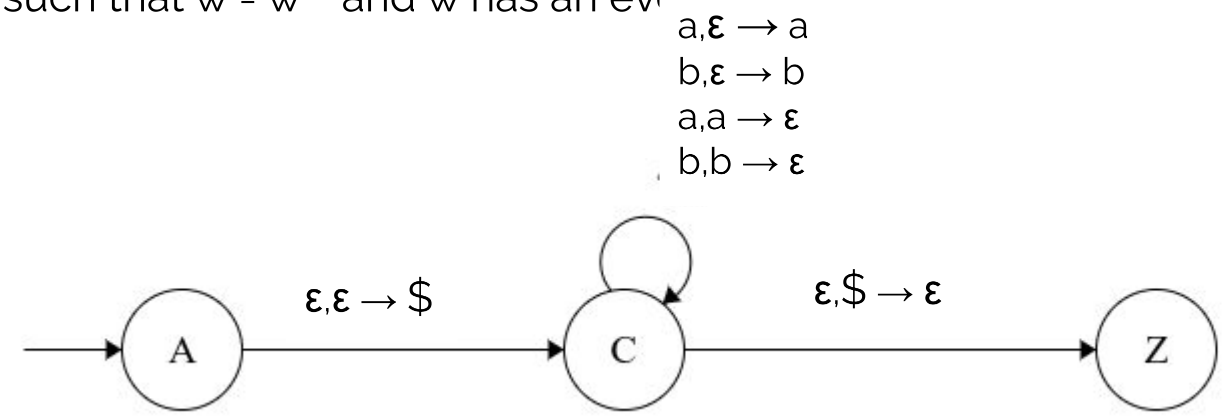
Examples for Creating Pushdown Automata

- Let's Construct a PDA that accepts all strings from the language
- L = {w such that w = w^R and w has an even length}



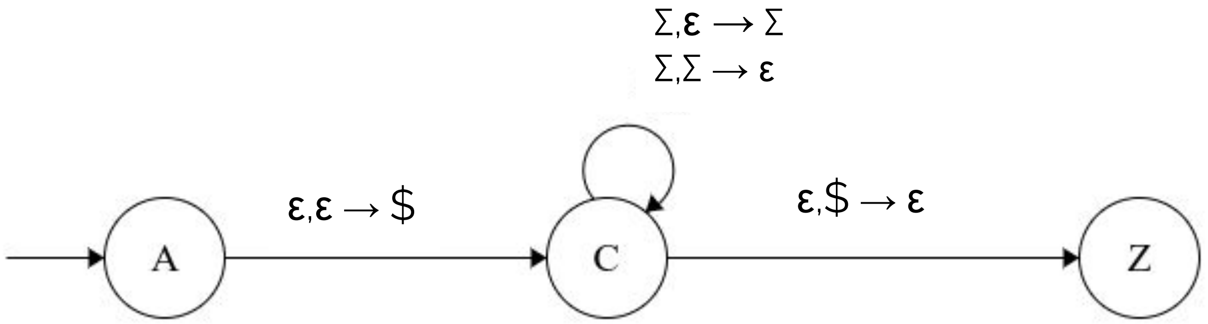
Examples for Creating Pushdown Automata

- Let's Construct a PDA that accepts all strings from the language
- L = {w such that w = w^R and w has an ev



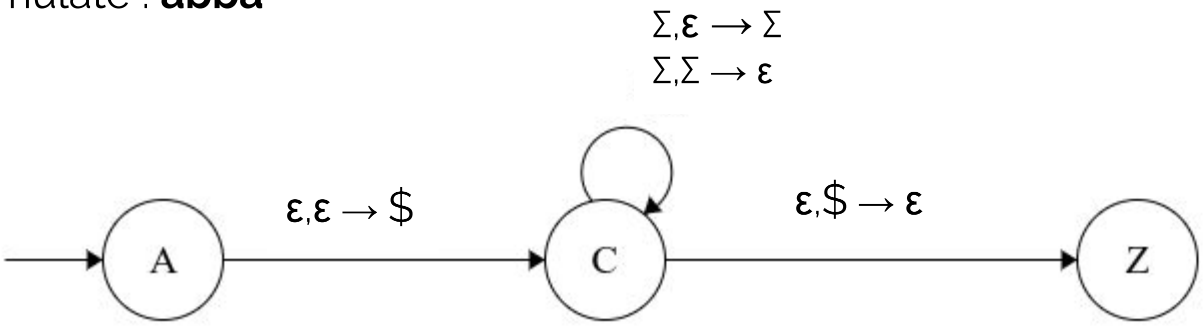
Examples for Creating Pushdown Automata

- Let's Construct a PDA that accepts all strings from the language
- L = {w such that w = w^R and w has an even length}



Examples for Creating Pushdown Automata

- Let's Construct a PDA that accepts all strings from the language
- L = {w such that w = w^R }
- Let's simulate : **abba**



Examples for Creating Pushdown Automata

- Let's Construct a PDA that accepts all strings from the language
- L = {w such that w = w^R }
- Let's simulate : **abba**

$\Sigma, \varepsilon \rightarrow \Sigma$

$\Sigma, \Sigma \rightarrow \varepsilon$

Nondeterminism is extremely important where the machine would try all possible derivations until it gets the correct one.

Deterministic PDA

- For each state in the PDA, and for any combination of a **current input symbol** and a **current stack symbol**, there is at most one transition defined
- In other words, there is **precisely at most** one legal sequence of transitions that can be followed for any input
- What about the e-transitions ?

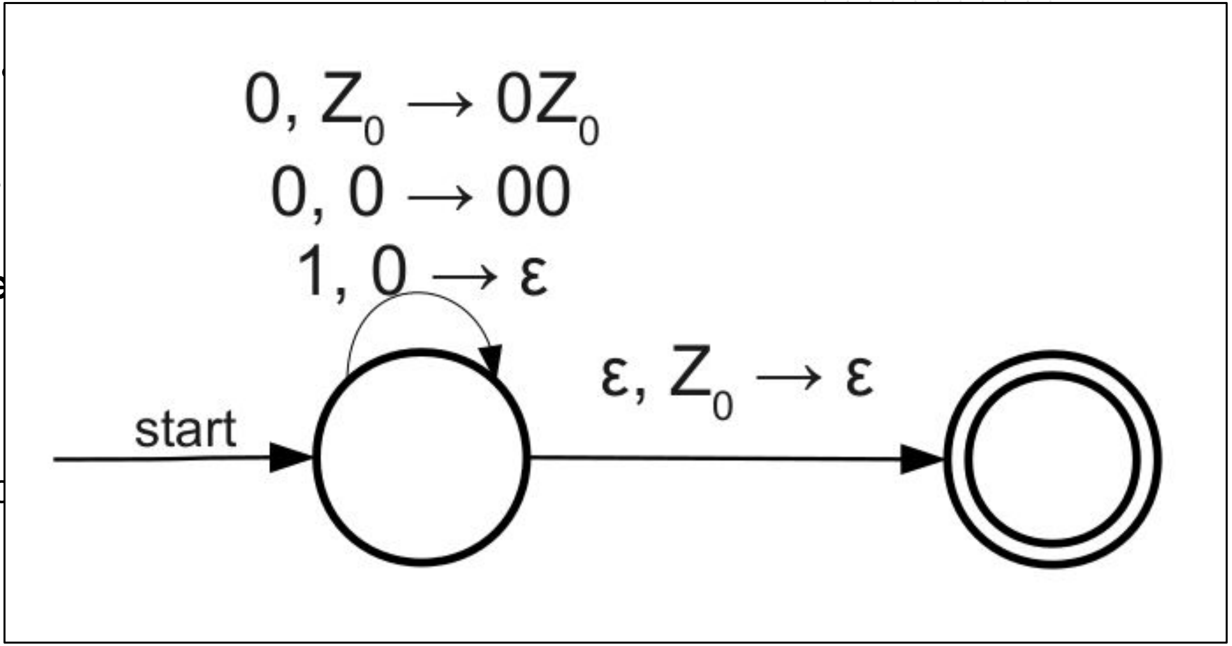
Taken from the slides of Keith Schwarz at Stanford

Deterministic PDA

- For each state in the PDA,

  current stack symbol, the
- In other words, there is **pre**

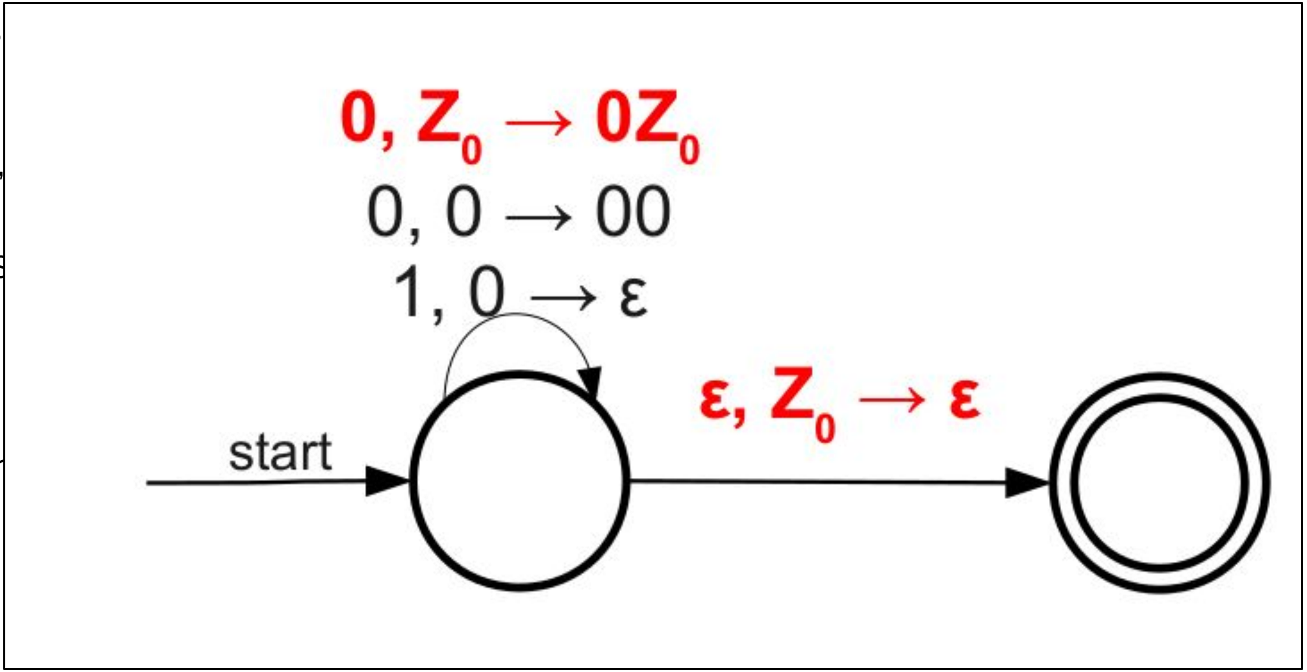
  followed for any input
- What about the e-transitic



Taken from the slides of Keith Schwarz at Stanford

Deterministic PDA

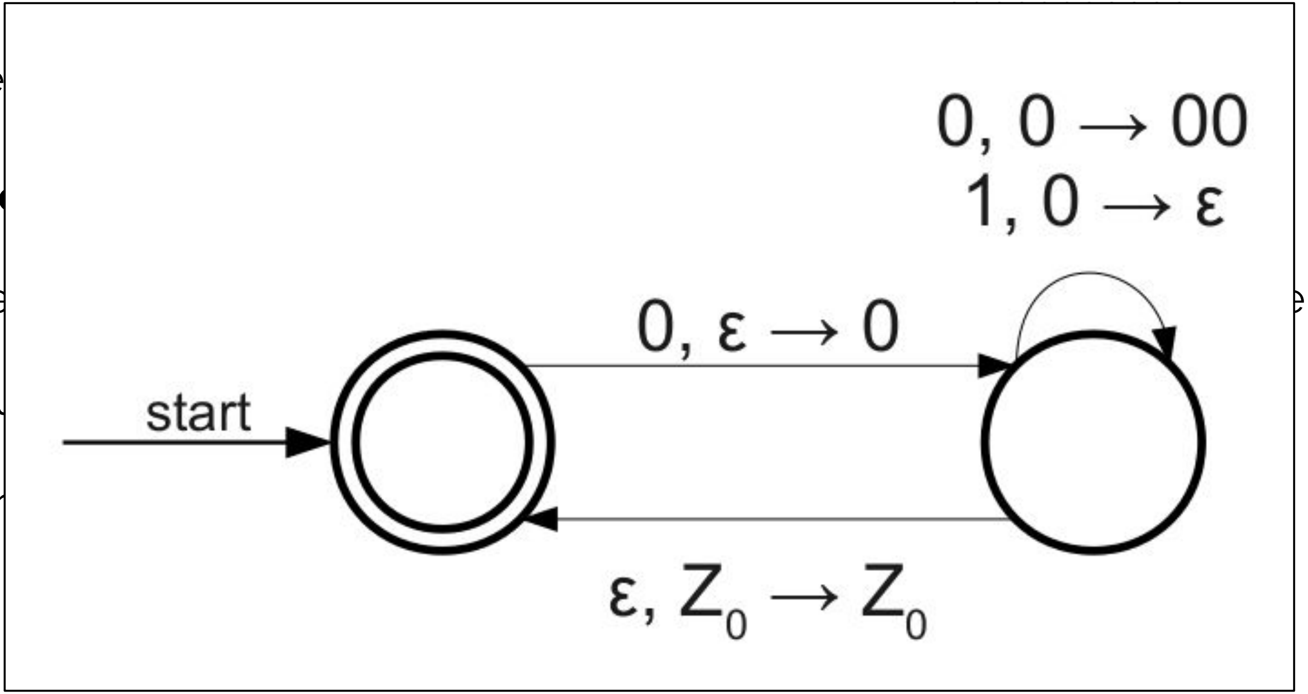
- For each state in the P
  current stack symbol,
- In other words, there is
  followed for any input
- What about the e-tran



Taken from the slides of Keith Schwarz at Stanford

Deterministic PDA

- For each state in the
  current stack symb
- In other words, there
  followed for any inpu
- What about the e-tr



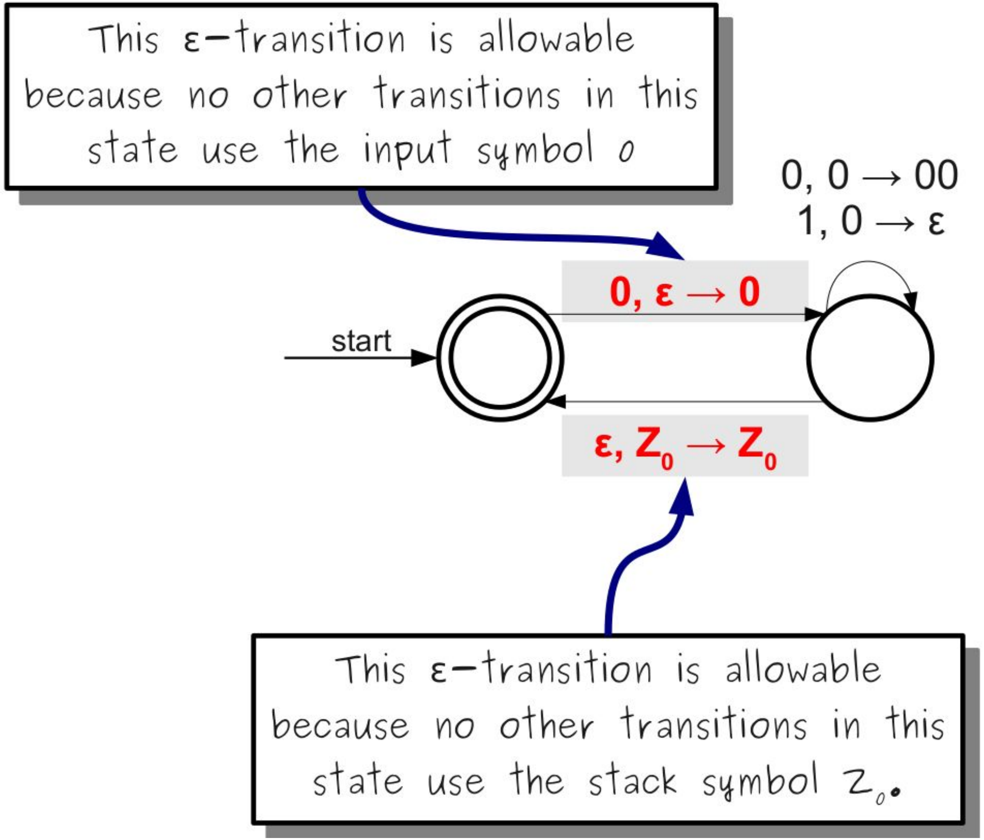
Taken from the slides of Keith Schwarz at Stanford

Deterministic PDA

- For each state in the PDA, and for any

  current stack symbol, there is at mos
- In other words, there is **precisely at m**

  followed for any input
- What about the e-transitions ?



Taken from the slides of Keith Schwarz at Stanford

Deterministic PDA

- If we can find a DPDA for a CFL, then we can recognize strings in that language efficiently.
- Can we guarantee that we can always find a DPDA for a CFL?

Deterministic PDA

- If we can find a DPDA for a CFL, then we can recognize strings in that language efficiently.
- Can we guarantee that we can always find a DPDA for a CFL?
  - As DFA and NFA are equivalent and each NFA has its DFA equivalent
  - Do PDA and DPDA have the same power ?
    - Does any CFL represented by a DPA, has an DPDA equivalent ?

Deterministic PDA

- Simple example: **The language of palindromes.**
- Design the algorithm for the DPDA
- How do you know when you've read half the string?
    - It is deterministic, the machine does not have the power for guessing or branching …

Equivalence

- A language is context free if and only if some pushdown automaton recognizes it.

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.

Equivalence : CFG $\rightarrow$ PDA

- Simple Idea :
  - **Push Variables into the stack**
  - **Replace Top Variable by its Production rules into the stack**

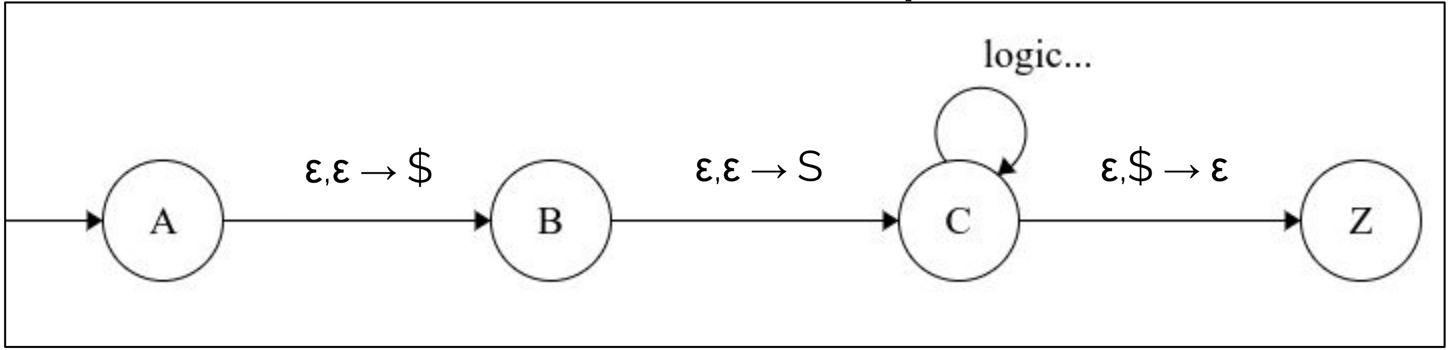
Equivalence : CFG $\rightarrow$ PDA

- Given the following grammar :
    - $S \rightarrow aS \mid \boldsymbol{\varepsilon}$
- **For simplification, we derive the following word:**
    - **aaa**

Equivalence : CFG → PDA

- Given the following grammar :
  - S → aS | **ε**
    - **Initialize the stack with the marker symbol $**
    - **Place the start variable at the top of the Stack**

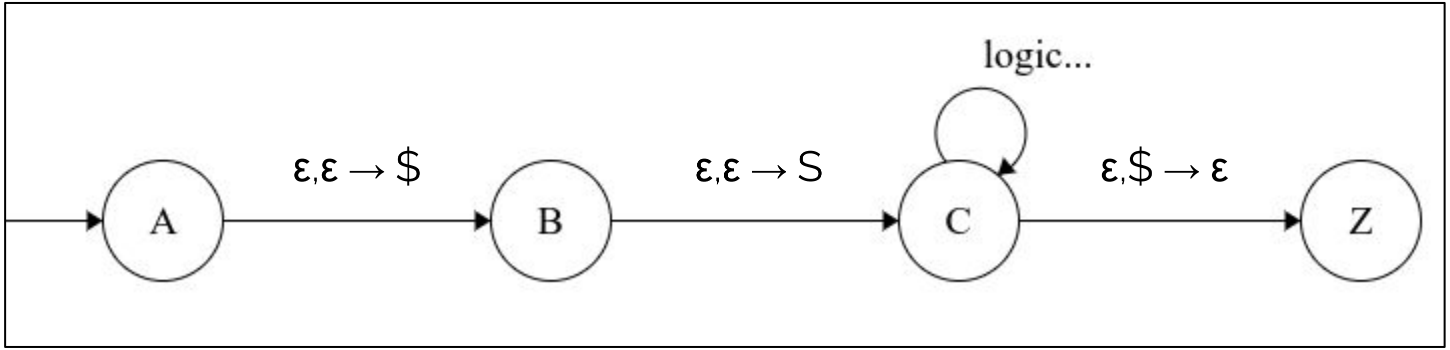
S
$



Equivalence : CFG → PDA

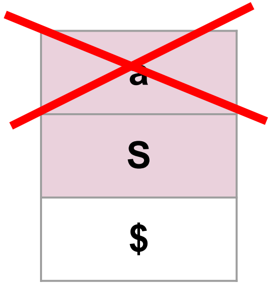
- Given the following grammar :
  - S → **aS** | **ε**
    - **We replace the variable with its production rule** S → **aS Within the stack**

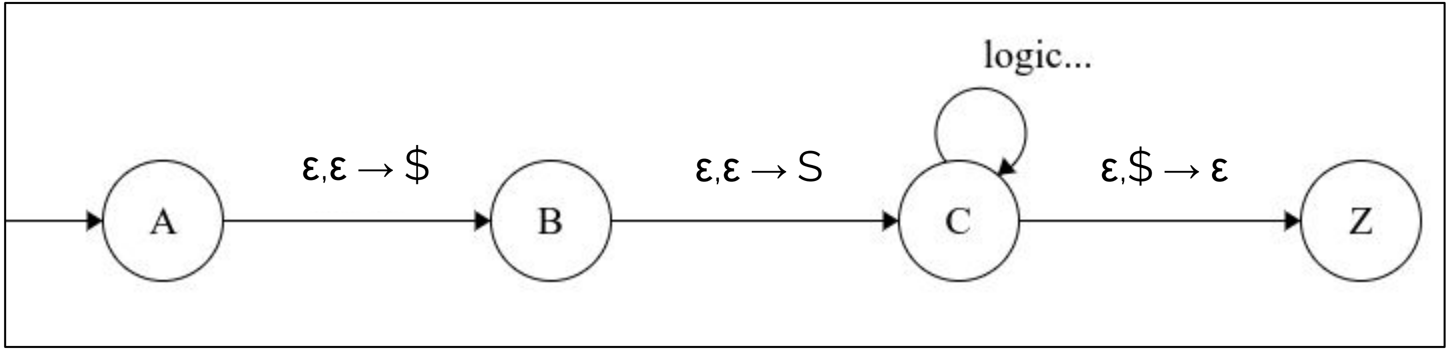
a
S
$



Equivalence : CFG → PDA

- Given the following grammar :
  - S → **aS** | **ε**
    - **Terminal Symbol "a" must be POPPED from the stack**

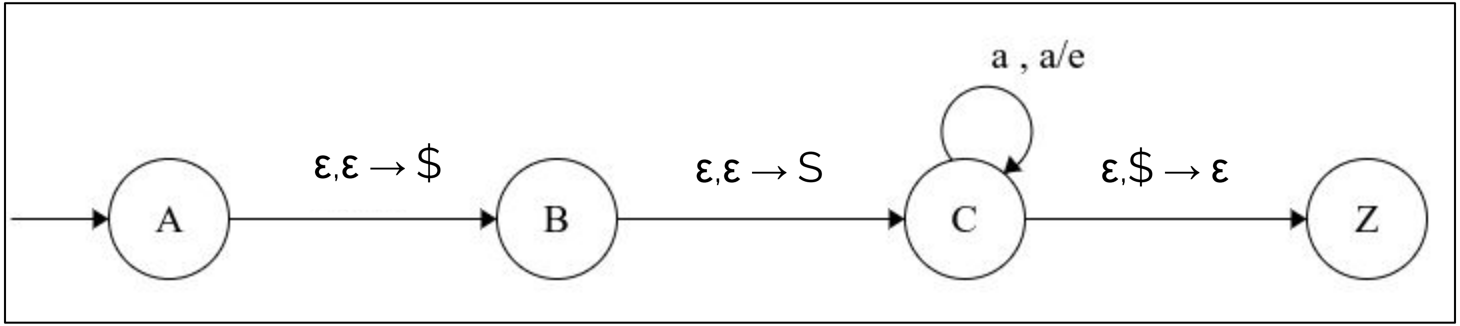




Equivalence : CFG → PDA

- Given the following grammar :
    - S → **aS** | **ε**
        - **The transition should be made as : a, a → ε**

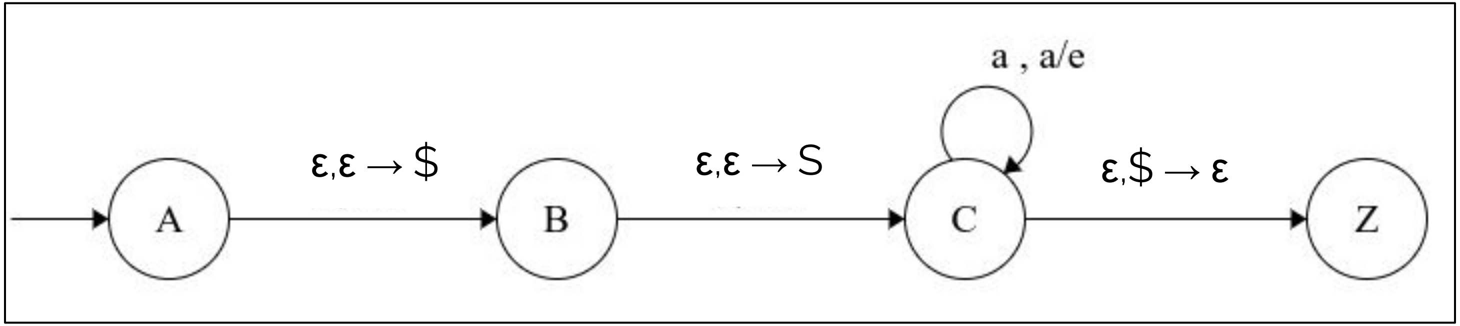
S
$



Equivalence : CFG → PDA

- Given the following grammar :
  - S → **aS** | **ε**
    - **For Variable S ? how can we do its transition ?**

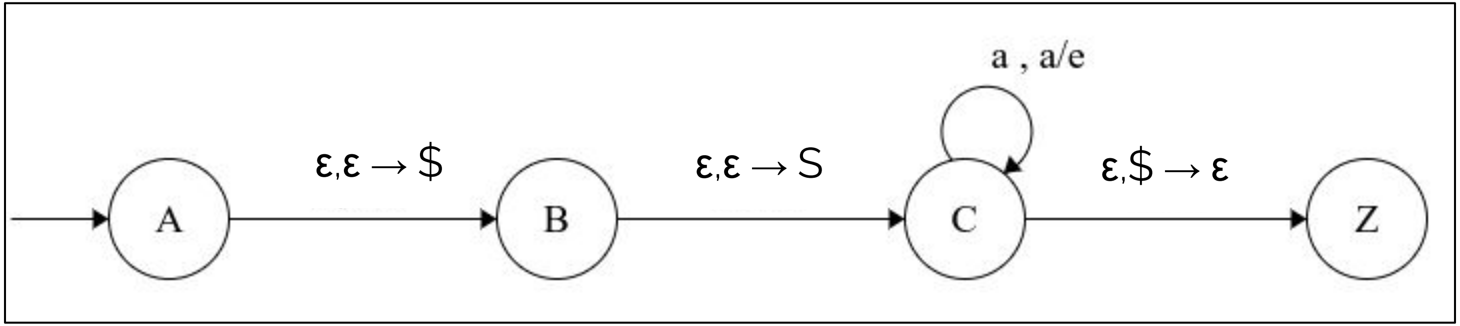
S
$



Equivalence : CFG → PDA

- Given the following grammar :
    - S → **aS** | **ε**
        - **For Variable S ? how can we do its transition ?**
            - **ε , S → a S**

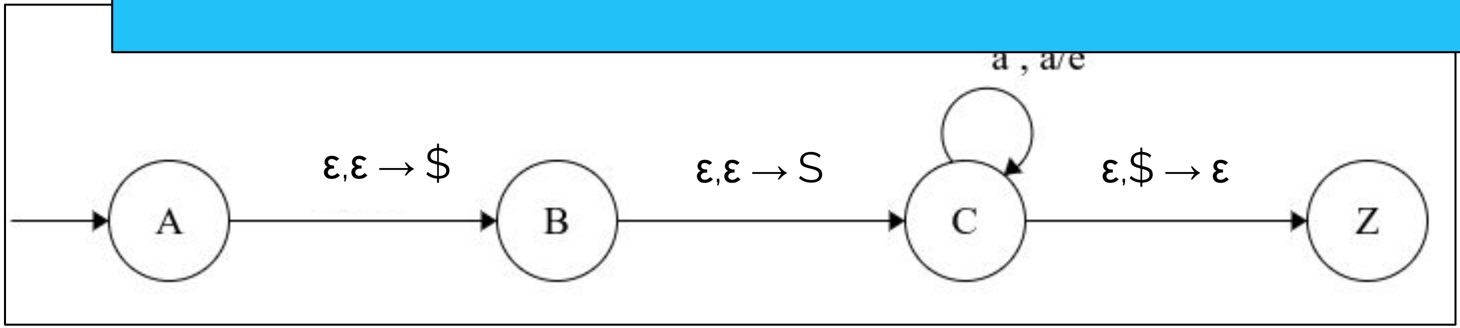
S
$



Equivalence : CFG → PDA

- Given the following grammar :

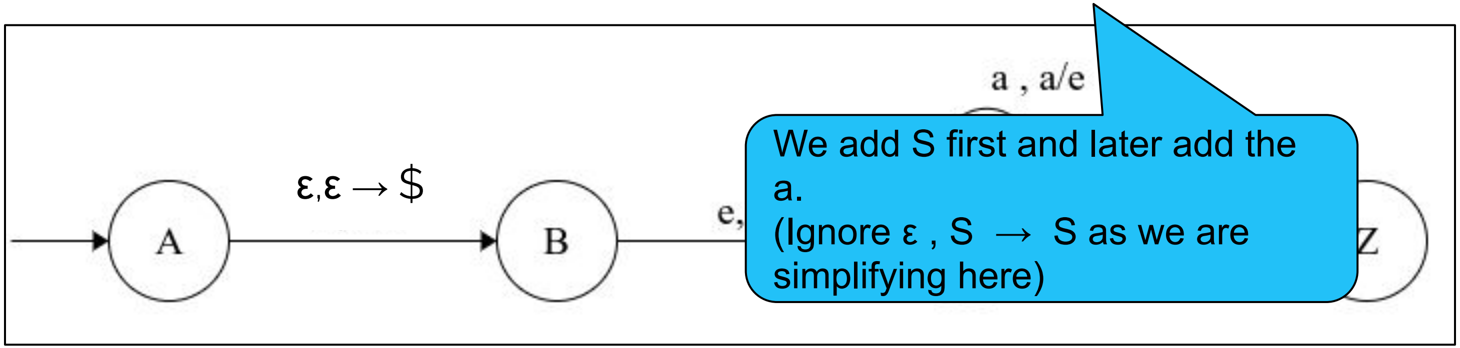
There are certain textbook restricting adding multiple values into the stack in one go.



Equivalence : CFG → PDA

- Given the following grammar :
  - S → **aS** | **ε**
    - **For Variable S ? how can we do its transition ?**
      - **ε , S → a S ⇒ two transitions : ε , S → S + ε , ε → a**

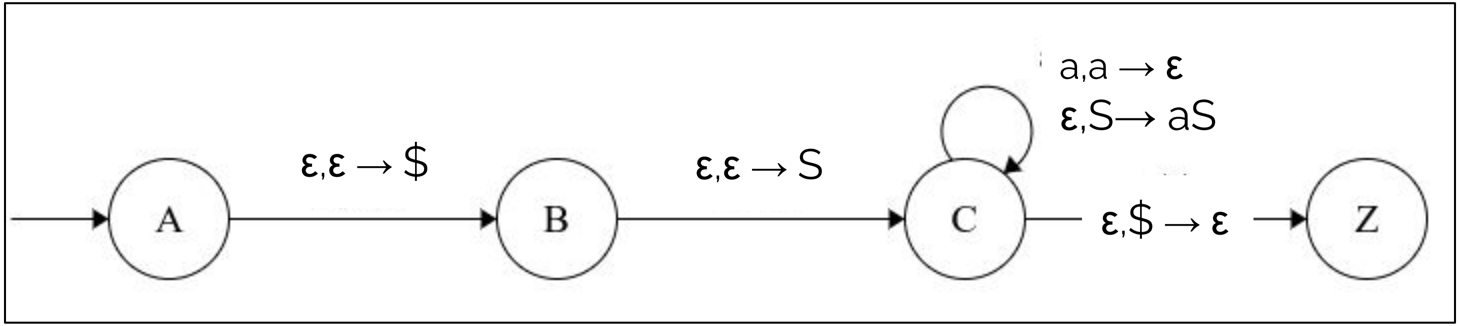
a
S
$



Equivalence : CFG → PDA

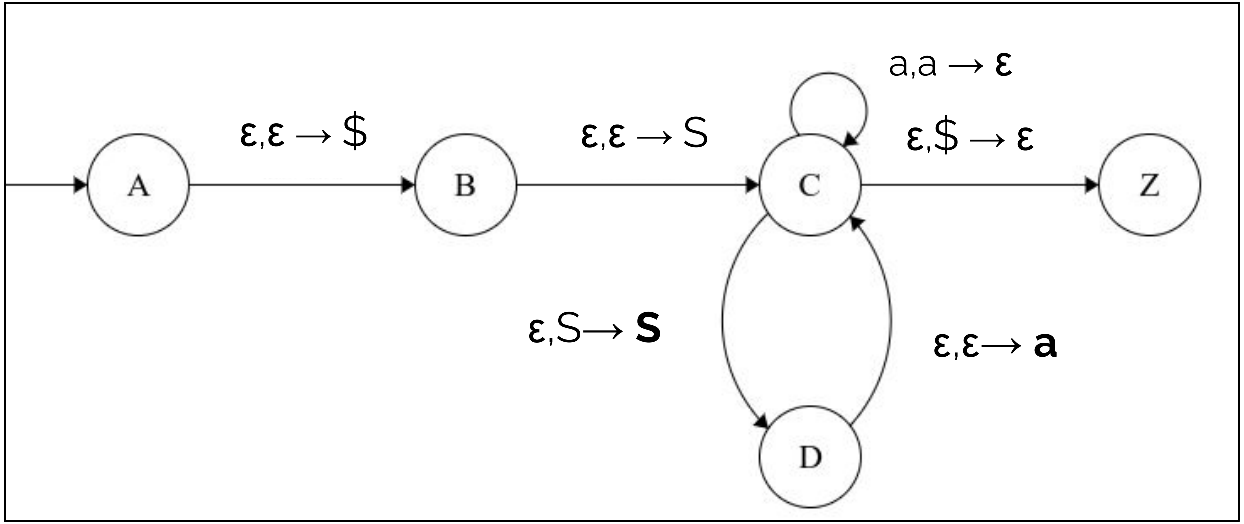
- Given the following grammar :
  - $S \rightarrow \mathbf{aS} \mid \boldsymbol{\varepsilon}$
    - **For Variable S ? how can we do its transition ?**
      - **$\varepsilon , S \rightarrow a\,S \Rightarrow$ two transitions : $\varepsilon , S \rightarrow S$ + $\varepsilon , \varepsilon \rightarrow a$**

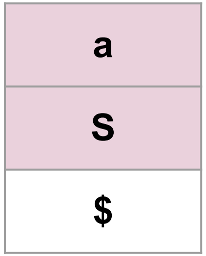
a
S
$



Equivalence : CFG → PDA

- Given the following grammar :
    - S → **aS** | **ε**



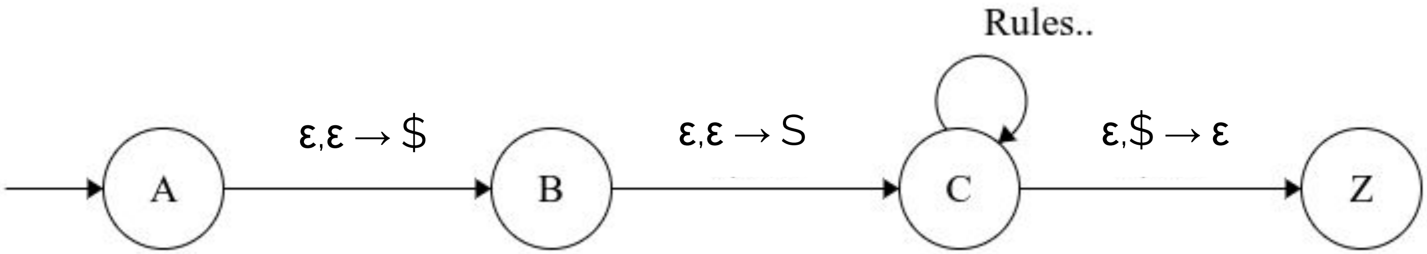


Equivalence : CFG → PDA

- Given the following grammar :
  - $S \rightarrow X \mid \varepsilon$
  - $X \rightarrow aXb \mid \varepsilon$

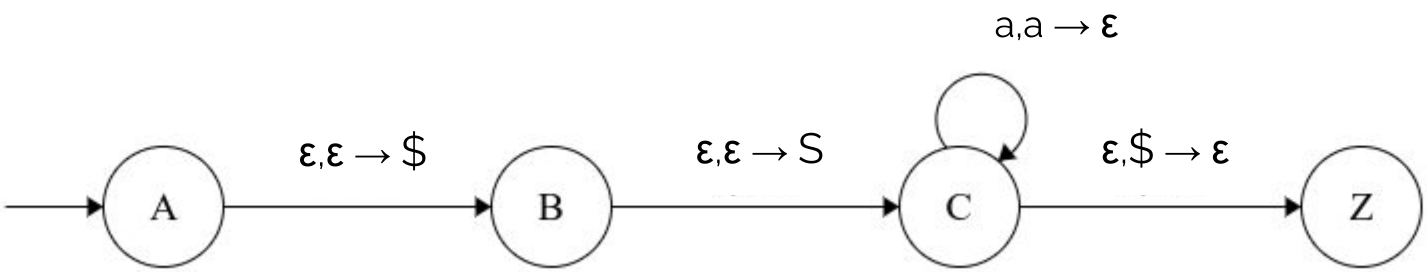
Equivalence : CFG → PDA

- Given the following grammar :
  - $S \rightarrow X \mid \varepsilon$
  - $X \rightarrow aXb \mid \varepsilon$



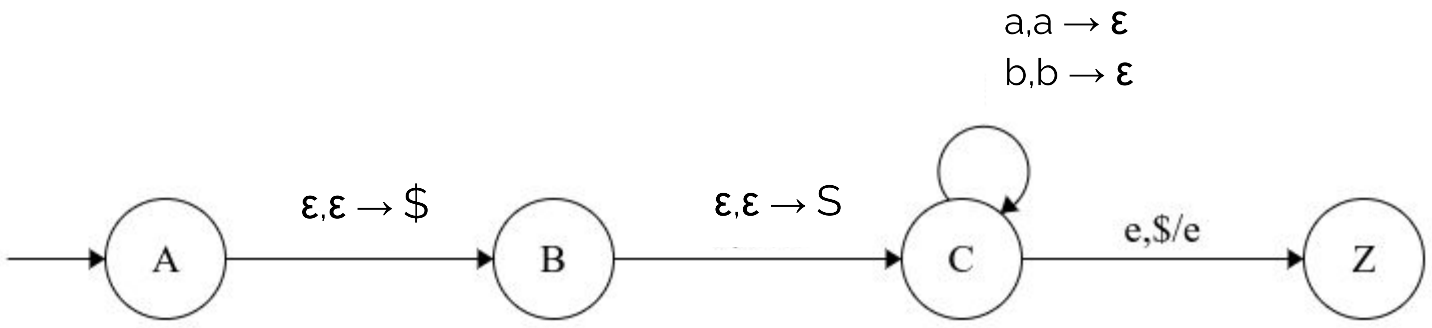
Equivalence : CFG → PDA

- Given the following grammar :
  - $S \rightarrow X \mid \varepsilon$
  - $X \rightarrow aXb \mid \varepsilon$



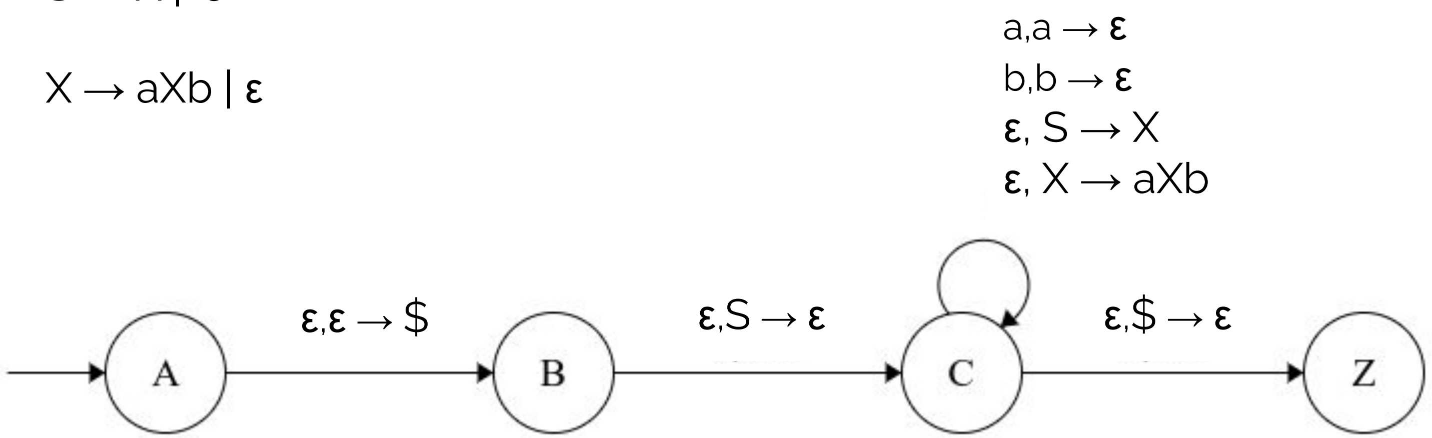
Equivalence : CFG → PDA

- Given the following grammar :
  - $S \rightarrow X \mid \varepsilon$
  - $X \rightarrow aXb \mid \varepsilon$



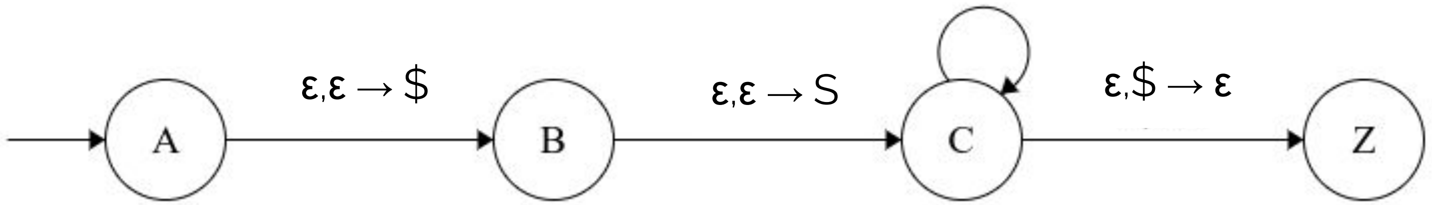
Equivalence : CFG → PDA

- Given the following grammar :
    - $S \rightarrow X \mid \varepsilon$
    - $X \rightarrow aXb \mid \varepsilon$



Equivalence : CFG → PDA

- Given the following grammar :
    - $S \rightarrow X \mid \boldsymbol{\varepsilon}$
    - $X \rightarrow aXb \mid \boldsymbol{\varepsilon}$

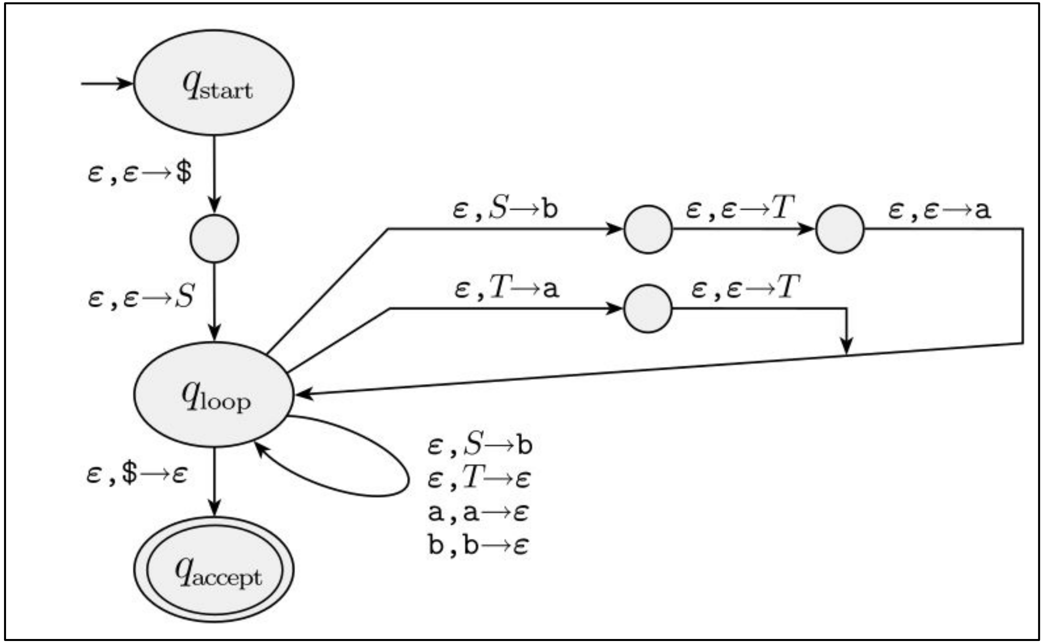


Equivalence : CFG → PDA

- Given the following grammar :
  - $S \rightarrow a\,T\,b \mid b$
  - $T \rightarrow Ta \mid \varepsilon$

Equivalence : CFG → PDA

- Given the following grammar :
  - S → a T b | b
  - T → Ta | ε



Equivalence : PDA $\to$ CFG

- A language is context free if and only if some pushdown automaton recognizes it.

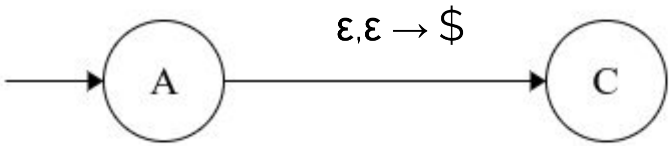
1. If a language is context free, then some pushdown automaton recognizes it.
2. **If a pushdown automaton recognizes some language, then it is context free.**

Equivalence : PDA → CFG

- The Algorithm:

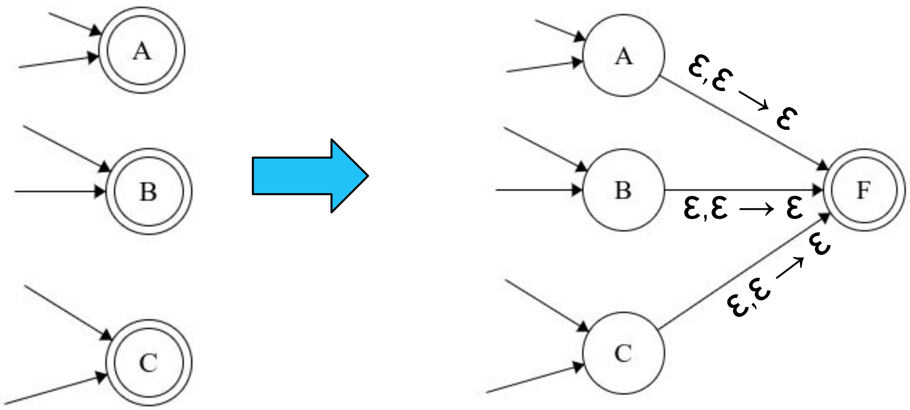
  1. Simplify the PDA:

     - *Create a new Start State and initialize the stack with **$***



Equivalence : PDA → CFG

- The Algorithm:
  1. Simplify the PDA:
     - *Should have only one accept state newly created*

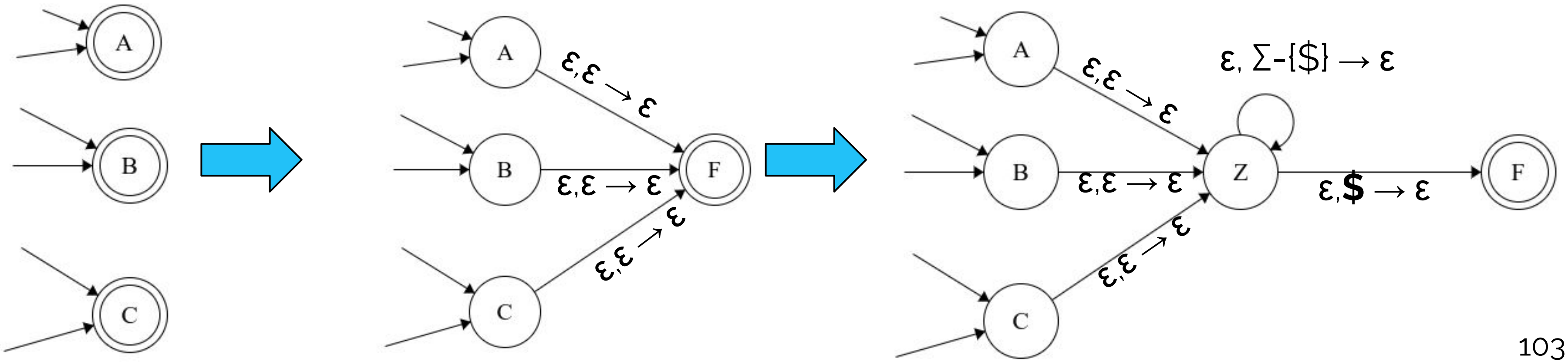


Equivalence : PDA → CFG

- The Algorithm:

  1. Simplify the PDA:

     - *The stack needs to be **emptied** just after passing the newly created final state*



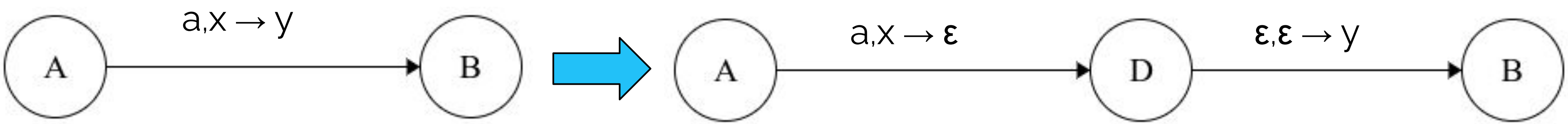
Equivalence : PDA → CFG

- The Algorithm:

    1. Simplify the PDA:

        - *Transform all transitions so that each transition would do at a time either :*

            - *push a symbol or*
            - *Pop a symbol*



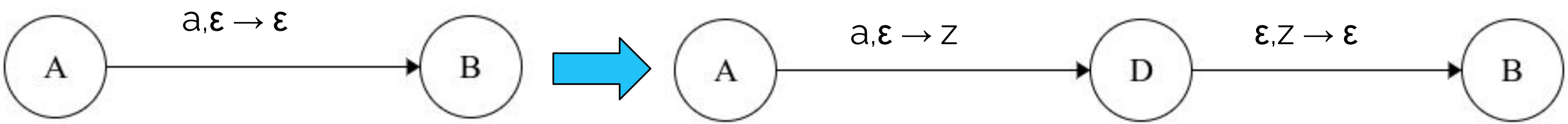
Equivalence : PDA → CFG

- The Algorithm:

  1. Simplify the PDA:

     - *Transform all transitions so that each transition would do at a time either :*

       - *push a symbol or*
       - *Pop a symbol*



Equivalence : PDA → CFG

- The Algorithm:

  2. Construct the Context Free Grammar

     - *For each **reachable/traversable** pair of states (A, B), create a variable (non-terminal symbol) V_{AB}*
     - *The start variable is V_{SF} such that S is the start state and F is the final state*
     - *Create Production Rules based on the following cases*
       - *.*
       - *...*

Equivalence : PDA → CFG

- The Algorithm:

  2. Construct the Context Free Grammar

     - *The start variable is V_{SF} such that S is the start state and F is the final stat*

Equivalence : PDA → CFG

- The Algorithm:

  2. Construct the Context Free Grammar

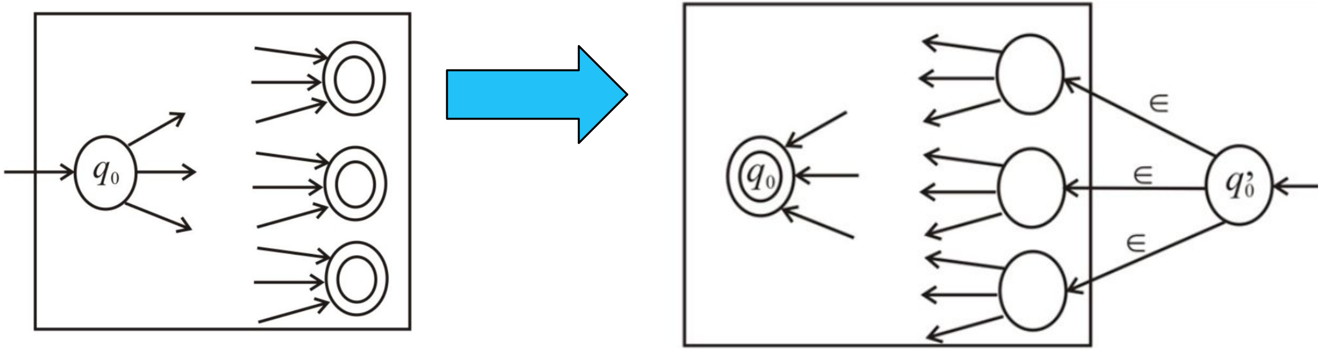
     ■ *Create Production Rules based on the following cases*

       - *if $\delta(p, a, \varepsilon) \rightarrow (r, u)$ and $\delta(s, b, u) \rightarrow (q, \varepsilon)$, add the following rule : G: $A_{pq} \rightarrow aA_{rs}b$*
       - *For each $p, q, r, s \in Q$, add the following rule to G: $A_{pq} \rightarrow A_{pr}A_{rq}$*
       - *For each $p \in Q$, add the following rule to G: $A_{pp} \rightarrow \varepsilon$*

TD5 - Solutions

- **Exercise 1**

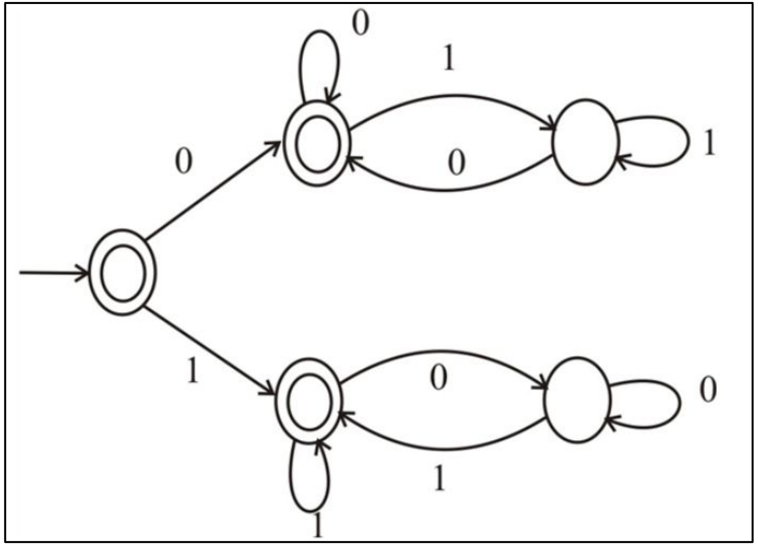
1. For any string $w = w_1 w_2 \cdots w_n$, the reverse of w, written w^R, is the string w in reverse order, $w_n \cdots w_2 w_1$. For any language A, let $A^R = \{w^R \mid w \in A\}$. Show that if A is regular, so is A^R.
   - Done by transforming the NFA for this language to as follows:
     - Invert the direction of all transitions
     - Create a new start state q0' and link them to the accepting states
     - Invert all accepting states into non-accepting states
     - Set the original start state as an accepting state.



TD5 - Solutions

- **Exercise 1**

2. Let Σ = {0,1} and let D = {w| w contains an equal number of occurrences of the substrings 01 and 10}. Thus 101 ∈ D because 101 contains a single 01 and a single 10, but 1010 not in D because 1010 contains two 10s and one 01. Show that D is a regular language.



TD5 - Solutions

- **Exercise 2**
  - Prove that the following languages are non-regular

    1. $A_1 = \{0^n 1^n 2^n \mid n \geq 0\}$
    2. $A_2 = \{www \mid w \in \{a, b\}^* \}$
    3. $A_3 = \{a2n \mid n \geq 0\}$ (Here, a2n means a string of 2^n a's.)
    4. $\{w \mid w \in \{0,1\}^*$ is not a palindrome$\}$ is not a regular

TD5 - Solutions

- **Exercise 2**

  - Prove that the following languages are non-regular

1. $A_1 = \{0^n 1^n 2^n \mid n \geq 0\}$
   a. We consider that the language A_1 is regular
   b. Therefore, there must be a pumping mechanism.
   c. We assume the pumping constant is **P**
   d. We take the string $0^p1^p2^p$ which is in the language,
   e. There are infinitely many words, and words with larger sizes that can be generated even from this word : $0^p1^p2^p \longrightarrow (0^{2p}1^{2p}2^{2p}$......)
   f. The word can be written in the form s=xyz such that |xy|<=P and |y|=k>0
      i. xy must be in the part of 0^p
      ii. Y must be only in the zero part.
      iii. If we pump Y, the new word will be in the form $0^P0^K1^P2^P = 0^{P+K}1^P2^P$
      We will **always have words not** in the language, as zeros will be more than 1 and 2.
   g. Therefore, we cannot pump more words from S to have new words in the language.
   h. ⇒ No pumping mechanism.
   i. The language is not regular

TD5 - Solutions

- **Exercise 2**

    - Prove that the following languages are non-regular

        1. $A_1 = \{0^n 1^n 2^n \mid n \geq 0\}$
        2. $A_2 = \{www \mid w \in \{a, b\}^* \}$ ⇒ $\mathbf{a^p b a^p b a^p b}$
        3. $A_3 = \{a\text{^}2\text{^}n \mid n \geq 0\}$ (Here, a2n means a string of 2^n a's.) ⇒ a^2^P
        4. $\{w \mid w \in \{0,1\}^*$ is not a palindrome$\}$ is not a regular ⇒ $0^p 1 0^p$

TD5 - Solutions

- **Exercise 2**
  - Prove that the following languages are non-regular
    1. A_3 = {a^2^n | n ≥ 0} (Here, a2n means a string of 2^n a's.) ⇒ a^2^P
       a. a^(2^p)
          i. Example for simplification only not to fix P at a given number :
             1. P=4 ⇒ aaaa …aaa (2x2x2x2 ⇒ 16 times)
             2. Next word is P+1=5 ⇒ aaa …aaa (2x2x2x2x2 ⇒ 32 times)
             3. ..
          ii. The word can be written as s=xyz such that |xy| <=P and |y|=K>0
          iii. Regardless of the value of y if we would like to generate the next word by pumping.
             1. We need to have the next word which must be in the language,
             2. $|xyz|=2^P$
             3. The next word is of course : xy^2z such that $\mathbf{|xy^2z|=2^{P+1}}$
             4. But as |xy| <=P even x is empty, and y is all the **a**s (by considering even |Y|=P at max)
                a. $|xy^2z| <= 2^p+p$ but
                b. $2^p+p < 2^{p+1}$ which is true always by induction
                c. Therefore, the new word will never be in the language.

TD5 - Solutions

- **Exercise 2**
  - Prove that the following languages are non-regular
    1. A_3 = {a^2^n | n ≥ 0} (Here, a2n means a string of 2^n a's.) ⇒ a^2^P
       a. **$2^p+p < 2^{p+1}$**
          - We assume that $2^p+p < 2^{p+1}$ is true
          - We multiply by both sides by 2
          - $2(2^p+p) < 2*2^{p+1}$
          - $2^{p+1}+2p < 2^{(p+1)+1}$
          - For P >= 1, it is always , p+1<p+p<2p,
            - Therefore, we can replace 2p with a lesser number inside the smaller side of the inequality.
          - $2^{p+1}+ p+1 < 2^{p+1}+ 2p < 2^{(p+1)+1}$
          - $2^{p+1}+ p+1 < 2^{(p+1)+1}$
          - Therefore, always true by induction

TD5 - Solutions

- **Exercise 3**
  - Let $B = \{a^k \mid k$ is a multiple of $n\}$. Show that for each $n \geq 1$, the language B_n is regular.
    - B_1 = a, we can write regular expressions = a
    - B_2 = (aa)*
    - B_3 = (aaa)*
    - ..
    - B_n= (aa…aa)* (a is repeated n times)
    - B is the union of B_1, B_2,...B_n is regular as the union of regular languages is regular.

TD5 - Solutions

- ## Exercise 3
  - For languages A and B, let the perfect shuffle of A and B be the language $\{w \mid w = a_1 b_1 \cdots a_k b_k$, where $a_1 \cdots a_k \in A$ and $b_1 \cdots b_k \in B$, each $a_i, b_i \in \Sigma\}$.
    Show that the class of regular languages is closed under a perfect shuffle.
    Example : abc ∈ A, 123 ∈ B , by perfectly shuffling → a1b2c3
  - The new language S will be constructed from A and B by taking words of the same size and taking a letter from each word in an alternating fashion.
    - If the states of the DFA for A is X={x0, x1, x2,x3…xn}
    - If the states of the DFA for B is Y={y0, y1, y2,y3…yn}
    - The DFA Machine can be constructed for the language S with the following states
      - X * Y * {A,B }
      - Examples
        - (x0,y1,A) (I am now at machine A, at state x0 whilst i was at state of y1 of B)
      - Start state would be: (x0,y0, A)
      - Accepting States would be:
        - $(x_{accept}, y_{accept}, A)$
      - Transitions would be:
        - δ((xn, yn, A) , a) → (δ(xn, a) , yn , **B**)
        - δ((xn, yn, B) , a) → (xn, δ(yn, a) , **A**)

TD6 - Solutions

In each case below, say what language is generated by the context-free grammar:

1. S → aS | bS | ε   **{a,b}***

2. S → SS | bS | a   **{a,b}*a**

3. S → SaS | b   **starts with b and ends with b + a and b are alternating**

4. S → SaS | b | ε   **does not contain bb**

5. S → T T   **contains exactly two bs**
   T → aT | T a | b

6. S → aSa | bSb | aAb | bAa   **not palindromes**
   A → aAa | bAb | a | b |ε | S

7. S → aT | bT | ε   **Even number of letters**
   T → aS | bS

8. S → aT | bT   **odd number of letters**
   T → aS | bS | ε

TD6 - Solutions

Give the context-free grammars that generate the following languages. Alphabet Σ is {0,1}.

1. {w| w contains at least three 1s}
   S→P1P1P1P
   P → 0P | 1P | e

2. {w| w starts and ends with the same symbol}
   S → 0 P 0 | 1 P 1 | 1 | 0
   P → 0P | 1 P | e

3. {w| the length of w is odd}
   S→ 0 | 1 | 00S | 10S | 10S | 11S
   Or
   S→ 0 | 1 | 0S0 | 0S1 | 1S0 | 1S1
   Or
   See previous exercise

4. {w| the length of w is odd and its middle symbol is a 0}
   S → 0 | 0S0 | 0S1 | 1S0 | 1S1

TD6 - Solutions

Give the context-free grammars that generate the following languages. Alphabet Σ is {0,1}.

1. {w| w = w^R , that is, w is a palindrome}
   S → 0S0 | 1S1 | 1 | 0 | ε

2. {w| w is not equal to w^R , that is, w is not a palindrome}
   S → 0S0 | 1S1 | 0A1 | 1A0
   A → 0A0 | 1A1 | 0 | 1 | ε | S

3. {number of 0 is the same as 1}
   S → ε | S0S1S | S1S0S
   OR
   S → 0S1 | 1S0 | SS | ε

4. All strings with more a's than b's
   S → S_1aS_1
   S_1 → $bS_1a|aS_1b|S_1S_1|aS_1|$ ε

   Test String : aabbaa :
   S—> S_1aS_1 → aS_1b aS_1 → aaS_1bb aS_1 → aaS_1bb aS_1→ aabb aS_1 → $aabbaaS_1$ → aabbaa