# **Course:** Automata Theory

# **Lecture 8:** Models of Computation – Push Down Stack Machines

**Lecturer:** Martha Gichuki

# Course description

- The course begins with an introduction to logic and formal grammar where learners will do a recap on sets, logic and truth tables, sequences, relations and functions
- A coverage of finite state machines, Push Down automata and Turing Machines (The Church's thesis) will culminate the study of various models of computation.
- Formal language and grammar will then follow to enable learners differentiate regular and context free languages.
- An evaluation of the computability and complexity of practical computational problems which are the foundations of automata theory will then be done and the outcome will be problem description.

# Learning outcomes:

## Lecture 8: Models of Computation – Push Down Stack Machines

At the end of the lecture the learner will be able to:

- Describe Pushdown Stack Machines
- Describe Pushdown Stack Machines using State Diagrams
- Formally describe various Pushdown Stack Machines

# Pushdown Automata (PDA)

- Such machines are identical to DFAs (or NFAs), except that **they additionally carry memory in form of a stack**.
- Recall that a stack is a data structure that follows the rule of Last In First Out (LIFO) while reading elements/symbols in the stack

- With the Push Down Stack machine, the transition function will now also depend on the **symbol(s) on top of the stack.**
- **Push** action is equivalent to **writing a symbol** into the stack
- **Pop action is equivalent to** reading a symbol from the stack

# Why is the symbol on top of the stack important?

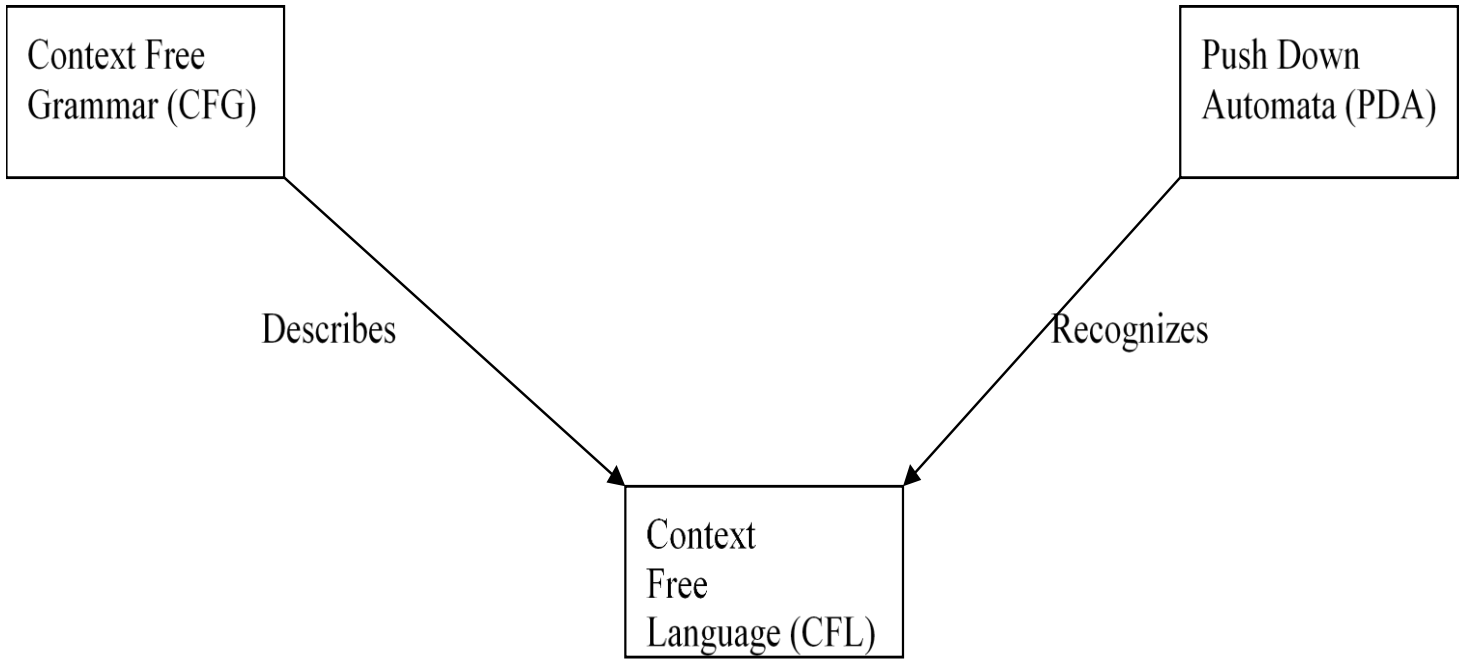
- The symbol on top of the stack will specify how the stack is to be changed at each transition.
- The symbol is replaced with another symbol meaning we pop it out and push in another symbol to take its place.

# Push Down Automata (PDA) are extensions of Finite Automata

- The family of languages accepted by the DFA and NFA is called the **family of regular languages**.
- More powerful automata can accept **more complicated languages.**
- Such automata include, Push Down Automatons (PDAs) and Turing Machines (TMs)

# What language does a PDA recognize?

- Non-deterministic PDAs accept languages  known as the Context-Free Languages (CFL).
- A Context Free Grammar (CFG) defines a Context Free Language (CFL) which is recognized by a Push Down Automata (PDA)



# Turing Machines (TMs)

- These are the **most powerful computational machines**.
- They possess an **infinite memory in the form of a tape**, and **a head which can read and change the tape, and move in either direction along the tape**.
- We shall discuss them in detail in our next lecture

# Turing Machines (TMs)…

- Turing machines are equivalent to algorithms and are the theoretical basis for modern computers.
- Turing machines decide recursive languages and recognize the recursively enumerable languages.
- Turing Machines have the essential feature of unrestricted access to limited memory, distinguishing them from weaker models like Finite Automata and Push Down Automata.

# Formal Definition of a Pushdown Automaton (PDA)

- The formal definition of a PDA is **similar to that of a Finite Automaton**, except for the additional memory of a **stack**.
- ***At the heart of any formal definition of an automaton is the transition function, for that is what describes its behaviour.***

# Formal Definition of the PDA

- A Pushdown Automaton is a **6-tuple ($Q, \Sigma, \Gamma, \delta, q_0, F$),**

where $Q$, $\Sigma$, $\Gamma$ and $F$ are all finite sets.

i. $Q$ is the set of finite states

ii. $\Sigma$ is the input alphabet,

iii. $\Gamma$ is the stack alphabet

iv. $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow P(Q \times \Gamma_\epsilon)$ is the transition function

v. $q_0 \in Q$ is the start state, and

vi. $F \subseteq Q$ is the set of accept states

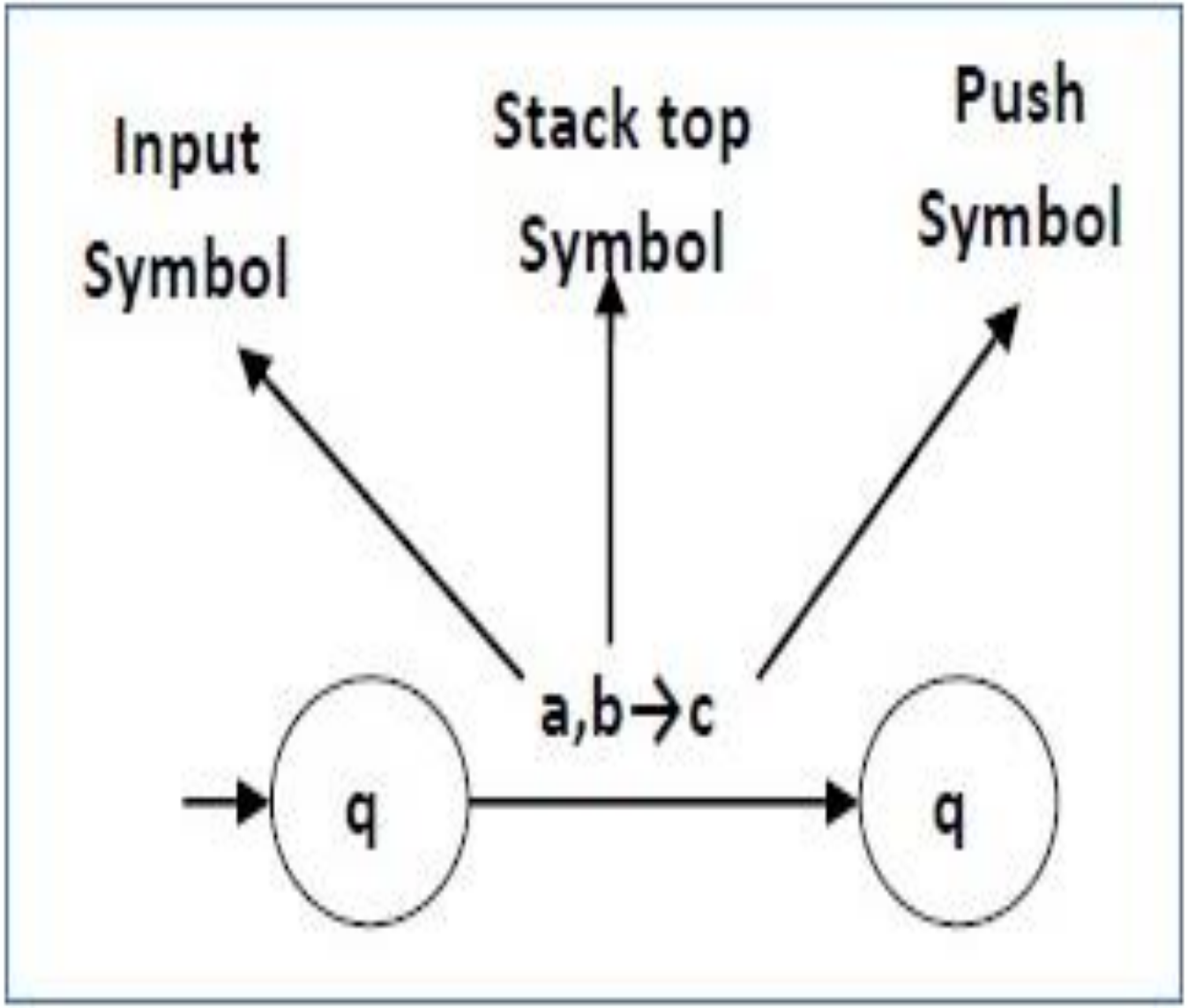
**How does the PDA Compute**?....

- ✓Generally, given the function **a, b → c** ;
- ✓It means that "a" is from the input alphabet and that when the machine **reads** an "a" from the input, **it *may* replace the symbol "b" on top of the stack with a symbol "c".**
- ✓*Any of a, b, c may be ∈.*

## **How does the PDA Compute**?....

- ✓If **a is ∈**, the machine may **make this transition without reading any symbol from the input**.
- ✓If **b is ∈**, the machine may **make this transition without reading and popping any symbol from the stack**.
- ✓If **c is ∈**, the machine **does not write any symbol on the stack when going along this transition.**

# How does the PDA compute?...



## **How does the PDA Compute**?

A Pushdown Automaton M=(Q, $\sum$, $\Gamma$, $\delta$ , $q_0$, F), computes as follows:-

- It accepts **input w if w** can be written as w= $w_1 w_2 \ldots w_m$ where:-
  - ✓ each $w_i \in \sum_{\epsilon}$ and
  - ✓ sequences of **states** $r_0\, r_1, \ldots . r_m \in Q$ and
  - ✓ **strings** $s_0\, s_1 \ldots s_m \in \Gamma^*$ exist that satisfy **the next three conditions.**

**How does the PDA Compute**?....

**Condition (i).** $r_0 = q_0$ and $s_0 = \in$.

✓This condition signifies that M starts out properly, in the **start state** and with an **empty stack**.

## How does the PDA Compute?....

**Condition (ii).** For $i = 0,\ldots m - 1$, we have $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, where $s_i = a_i$ and $s_{i+1} = b_t$ for some a, $b \in \Gamma_\epsilon$ and $t \in \Gamma^*$.

- ✓This condition states that machine M moves properly according to **the current state, stack and next input symbol**.

## How does the PDA Compute?

**Condition (iii)**. $r_m \in F$.

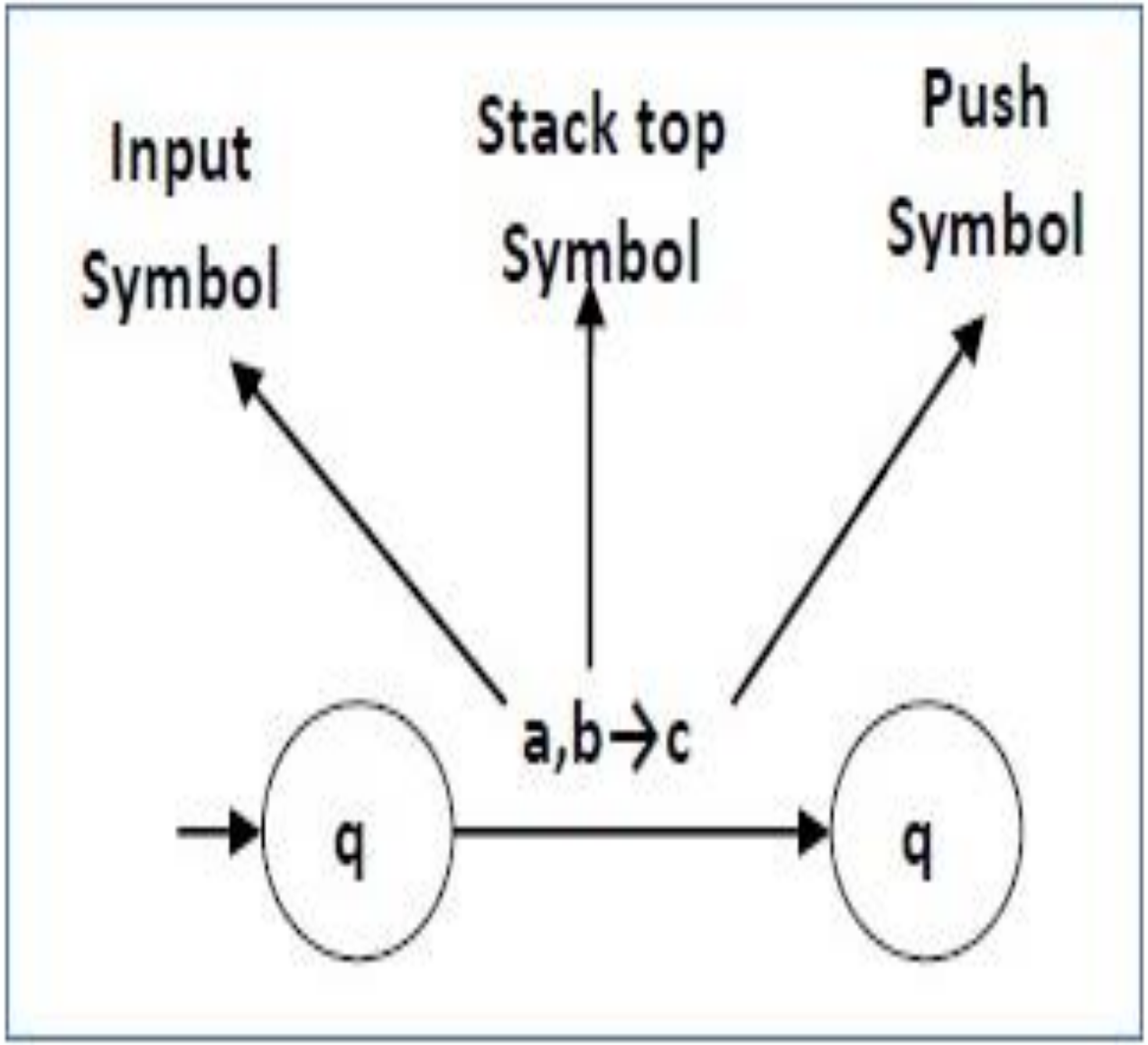
- ✓This condition states that an **accept state occurs at the input end.**
- ✓*The strings* $s_0 s_1 \ldots s_m \in \Gamma^*$ *represent the sequence of stack contents that machine M has on the accepting branch of the computation.*

# Characteristic of PDA

Given $(q_j, B) \in \delta(q_i, a, A)$: means that the PDA when **in state $q_i$, reads symbol a**, and **having symbol A on top of the stack**, is allowed to do the following: -

- Read a
- POP A
- PUSH B
- Change to state $q_j$

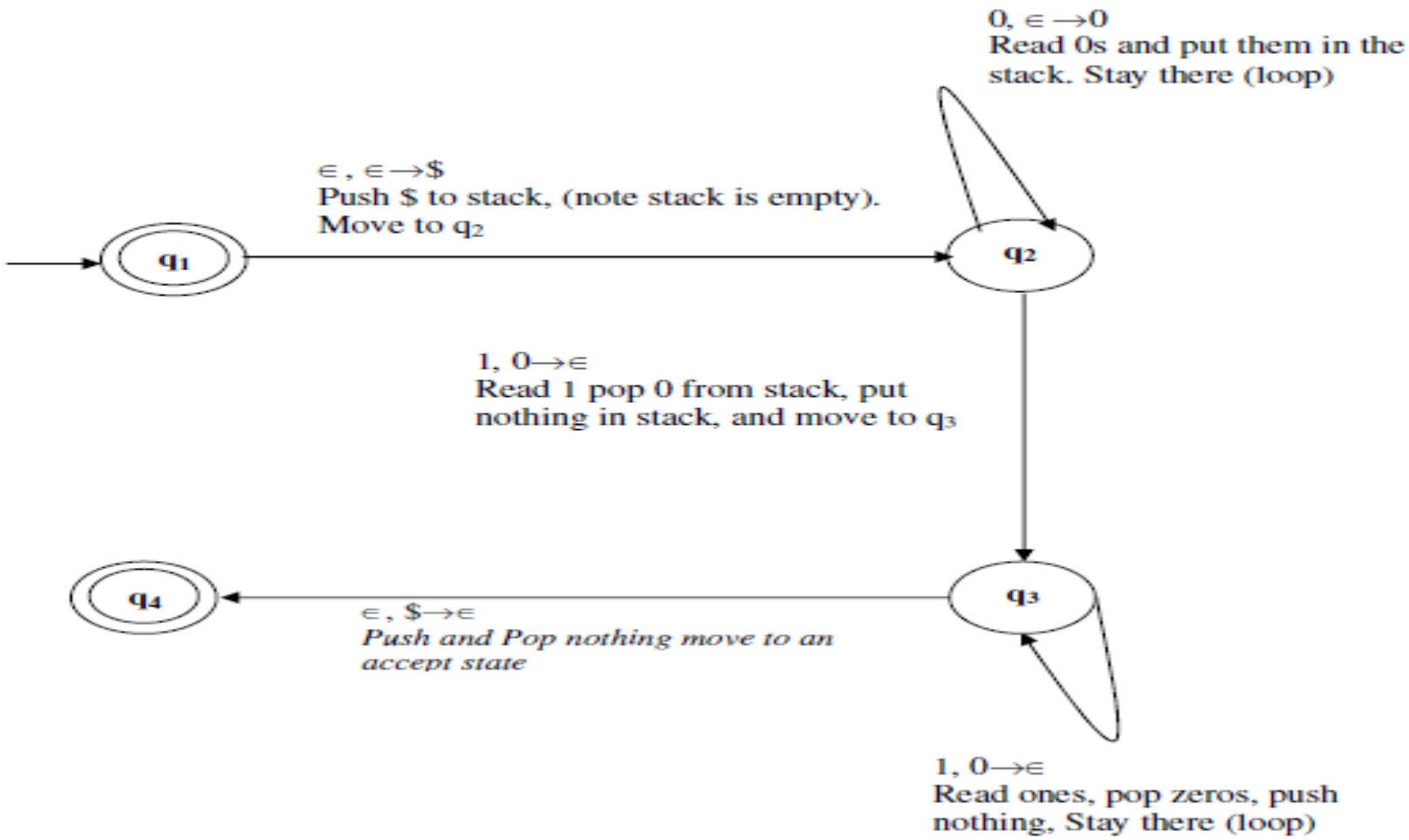
# Recall how the PDA computes...



# Consider the PDA machine that recognizes the language

$\{0^n,1^n \mid n\geq 0\}$,

# We design the following state diagram for the PDA.




Source: Introduction to the theory of computation (3rd ed.), Michael, S. Boston, Cengage Learning. ISBN-13: 978-1133187790, (2012). Page 115.

- ✓Recall the general function **a, b → c** ;
- ✓"a" is from the input alphabet
- ✓when the machine **reads** an "a" from the input, **it *may* replace the symbol "b" on top of the stack with a symbol "c".**
- ✓*Any of a, b, c may be ∈.*

- NB: **$ is a symbol / Mechanism to help us know that the stack is empty**.
- Now, from the above characteristic of PDA, we can conclude that: -
- $(q_2, \$) \in \delta(q_1, \in, \in)$
- $(q_2, 0) \in \delta(q_2, 0, \in)$

# Now, from the above characteristic of PDA, we can conclude that: -

- $(q_3, \in) \in \delta(q_2, 1, 0)$
- $(q_3, \in) \in \delta(q_3, 1, 0)$
- $(q_4, \in) \in \delta(q_3, \in, \$)$

etc.

The following is the formal description of this PDA that recognizes the language $\{0^n,1^n \mid n\geq 0\}$,

- Let $M_1$ be $(Q, \sum, \Gamma, \delta , q_0, F)$;

Where;

i. $Q = \{ q_1, q_2, q_3, q_4\}$,

ii. $\sum = \{0,1, \in\}$

iii. $\Gamma = \{0, \$, \in\}$

iv. $\delta: Q \times \sum_{\in} \times \Gamma_{\in} \rightarrow P(Q \times \Gamma_{\in})$ is the transition function given by the **table in the next slide**: -

v. $q_0 \in Q = q_1$ is the start state, and

vi. $F \subseteq Q = \{q_1, q_4\}$

$\delta$: Q X $\sum_{\in}$ X $\Gamma_{\in}$ $\rightarrow$P (Q X $\Gamma_{\in}$) is the transition function as shown in the table below

| Input Stack | 0 | | | 1 | | | ∈ | | |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | $ | ∈ | 0 | $ | ∈ | 0 | $ | ∈ |
| $q_1$ | | | | | | | | | $(\{q_2, \$\})$ |
| $q_2$ | | | $(\{q_2, 0\})$ | $(\{q_3, \in\})$ | | | | | |
| $q_3$ | | | | $(\{q_3, \in\})$ | | | | $(\{q_4, \in\})$ | |
| $q_4$ | | | | | | | | | |

recall

- ✓ $(q_2, 0) \in \delta(q_2, 0, \in)$
- ✓ $(q_2, \$) \in \delta(q_1, \in, \in)$
- ✓ etc.

**Question:** Indicate the set of steps followed by the PDA upon reading string 000111

$(q_2, \$) \in \delta(q_1, \in, \in)$
$(q_2, 0) \in \delta(q_2, 0, \in)$
$(q_2, 0) \in \delta(q_2, 0, \in)$
$(q_2, 0) \in \delta(q_2, 0, \in)$
$(q_3, \in) \in \delta(q_2, 1, 0)$
$(q_3, \in) \in \delta(q_3, 1, 0)$
$(q_3, \in) \in \delta(q_3, 1, 0)$
$(q_4, \in) \in \delta(q_3, \in, \$)$

# Does the PDA recognize the string 000111?

- **Yes** it does because the a symbol in the equation **a, b → c ; is ∈**, the machine while at state **$q_3$may make a transition landing at state $q_4$ without reading any symbol from the input**.
- State **$q_4$ being the accept state** means that the machine accepts string 000111

# References


- Rowan G. & John T., (2009), *Discrete Mathematics: Proofs, Structures and Applications*, CRC Press, ISBN: 9781439812808.
- W. D. Wallis (2003), *A Beginners Guide to Discrete Mathematics*, Springer Science & Business Media, ISBN: 978-0817642693.
- Introduction to the theory of computation (3rd ed.), Michael, S. Boston, Cengage Learning. ISBN-13: 978-1133187790, (2012).
- Introduction to languages and the theory of computation (3rd ed.), Martin, J., New York: McGraw-Hill. ISBN-13: 978-0072322002, (2002)