

Multi-stack Machine

1. Formal Description

$M = (Q, \Sigma, \Gamma, \partial, q_0, Z_0, F, k)$

where:

- Q - Set of states,
- Σ - Input Alphabet,
- Γ - Stack Alphabet,
- ∂ - Transition function,
- q_0 - Start state,
- Z_0 - End of stack symbol,
- F - Set of final states,
- k - Number of stacks,

The transition function is:

$\partial(q, a, x_1, x_2, \dots, x_k) = (p, y_1, y_2, \dots, y_k)$

where:

- q - Current state,
- a - Current input,
- x_i - Symbol on top of stack i ,
- p - New state,
- y_i - New symbol on top of stack i ,

2. Chomsky Hierarchy

Depending on the number of stacks, a multi-stack machine can recognize different languages.

- 0-stack machine \Leftrightarrow FSA \Leftrightarrow Accepts Regular Languages.
- 1-stack machine \Leftrightarrow PDA \Leftrightarrow Accepts Context-Free Languages.
- 2 or more stack machine \Leftrightarrow TM \Leftrightarrow Accepts Recursive Enumerable Languages.

Theorem 1 (Theorem 8.13—page 349):

If a language L is accepted by a Turing Machine, then L is accepted by a 2-stack machine.

Theorem 2 (Theorem 8.9—page 338):

Every language accepted by a multi-tape Turing Machine is Recursively Enumerable.

Theorem 3:

If language L is accepted by a k -stack machine, then L is also accepted by a k -tape Turing Machine.

Proof: Take a k -stack machine, and simulate each stack as a separate tape on a Turing Machine. The alphabet of each tape is the stack alphabet of the multi-stack machine. The idea is that the cursor on each tape will point to the rightmost element on the tape, thus simulating the fact that a k -stack machine can look at only the topmost element on each stack. If the stack machine:

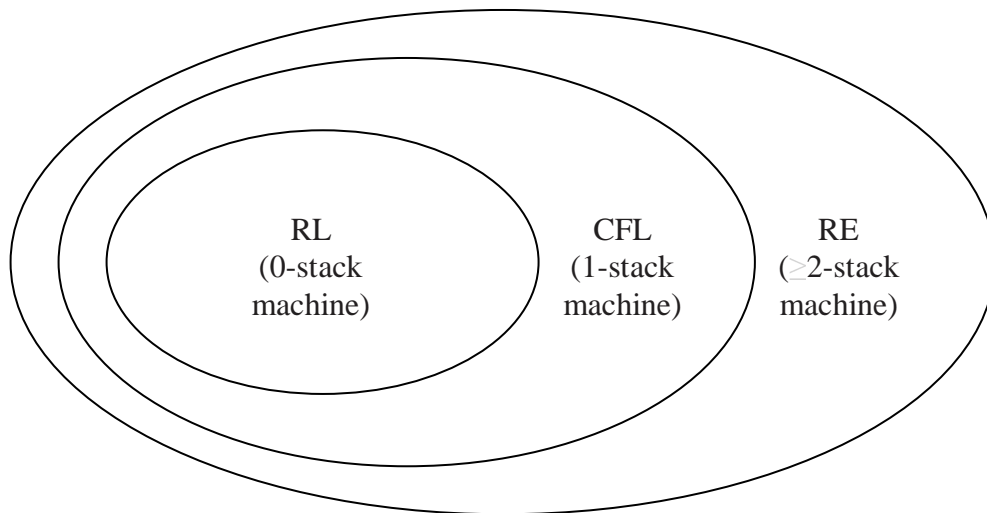
- pushes element a on stack i , then the TM will move cursor i to the right (R), write a , move R, then move L. Thus the cursor is on the new element a .
- pops element a from stack i , then the TM will write a blank (B) and move cursor i to the L. However, if $a = Z_0$ (i.e. the element popped is the end of stack marker), then the TM does not do anything.

By using this conversion, any language accepted by the k -stack machine, will be accepted by the k -tape TM.

Theorem 4:

If language L is accepted by a k -stack machine, then L is also accepted by a 2-stack machine.

Proof: The language L accepted by k -stack machine, will be accepted by a k -tape Turing Machine, by Theorem 3. So, L is accepted by a single tape Turing Machine, by Theorem 2. And also L is accepted by a 2-stack machine, by Theorem 1.



The 0-stack machine is equivalent (i.e. accepts the same languages and not more) to the Finite State Automata, the 1-stack machine is equivalent to the Push Down Automata, and the 2 or more stack machine is equivalent to the Turing Machine, as proved above.

3. Multi-Stack Algorithm

The multi-stack machine can be considered an algorithm, because it can simulate a sequence of steps that always finishes and produces an answer. For example a multi-stack machine can compute subtractions and give a definite answer. Consider

the function: $f = \begin{cases} m - n, & \text{if } m \geq n \\ 0, & \text{otherwise} \end{cases}$ using unary notation.

Answer:

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}, \{0, 1\}, \{0, 1, Z_0\}, \partial, q_0, Z_0, \{q_f\}, 2)$

Where ∂ is:

- Pushing string on stack 1
 - $\partial (q_0, 0, X, X) \rightarrow \{ (q_0, 0X, X) \}$
 - $\partial (q_0, 1, X, X) \rightarrow \{ (q_0, 1X, X) \}$
 - $\partial (q_0, \Lambda, X, X) \rightarrow \{ (q_1, X, X) \}$

- Flip Everything
 - $\partial (q_1, \Lambda, 0, X) \rightarrow \{ (q_1, X, 0X) \}$
 - $\partial (q_1, \Lambda, 1, X) \rightarrow \{ (q_1, X, 1X) \}$
 - $\partial (q_1, \Lambda, Z_0, X) \rightarrow \{ (q_2, Z_0, X) \}$

- See 1st "0"
 - $\partial (q_2, \Lambda, X, 0) \rightarrow \{ (q_3, X, \epsilon) \}$
 - $\partial (q_3, \Lambda, X, 0) \rightarrow \{ (q_3, 0X, \epsilon) \}$
 - $\partial (q_3, \Lambda, X, 1) \rightarrow \{ (q_3, 1X, \epsilon) \}$
 - $\partial (q_3, \Lambda, X, Z_0) \rightarrow \{ (q_4, X, Z_0) \}$

- Pop 2nd "0"
 - $\partial (q_4, \Lambda, 0, X) \rightarrow \{ (q_1, \epsilon, X) \}$
 - $\partial (q_4, \Lambda, 1, X) \rightarrow \{ (q_f, \epsilon, X) \}$ ***FINAL**

- If $n \leq m$
 - $\partial (q_2, \Lambda, X, 1) \rightarrow \{ (q_5, X, \epsilon) \}$
 - $\partial (q_5, \Lambda, X, 0) \rightarrow \{ (q_5, X, \epsilon) \}$
 - $\partial (q_5, \Lambda, X, Z_0) \rightarrow \{ (q_f, X, Z_0) \}$ ***FINAL**

4. Examples

KEY: Λ - means no input
 ϵ - means pop the stack
 X - means any stack element
 q_f - means final state

a) $L = \{ ww^R \mid w \text{ is in } \{a, b\} \}$

This language is an example of a Context Free Language that our multi-stack machine can solve deterministically.

$$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_f\}, \{a, b\}, \{a, b, Z_0\}, \partial, q_0, Z_0, \{q_f\}, 2)$$

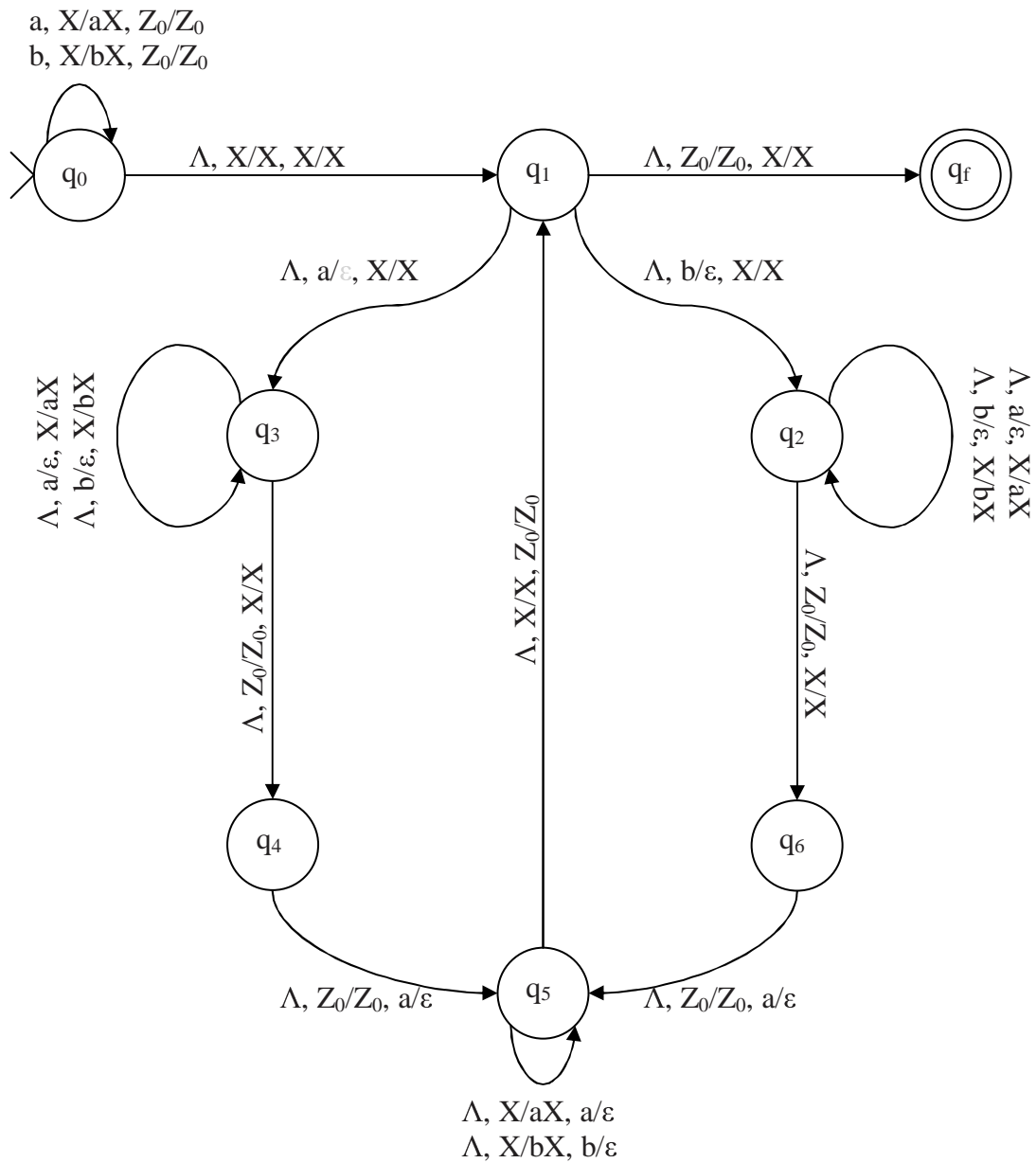
- Pushing string on stack 1
 $\partial (q_0, a, X, Z_0) \rightarrow \{ (q_0, aX, Z_0) \}$
 $\partial (q_0, b, X, Z_0) \rightarrow \{ (q_0, bX, Z_0) \}$
- Done pushing everything on stack 1
 $\partial (q_0, \Lambda, X, X) \rightarrow \{ (q_1, X, X) \}$
- See “a” on Stack 1, then POP it
 $\partial (q_1, \Lambda, a, X) \rightarrow \{ (q_3, \epsilon, X) \}$
- Push everything on stack 2
 $\partial (q_3, \Lambda, a, X) \rightarrow \{ (q_3, \epsilon, aX) \}$
 $\partial (q_3, \Lambda, b, X) \rightarrow \{ (q_3, \epsilon, bX) \}$
- Done pushing on stack 2
 $\partial (q_3, \Lambda, Z_0, X) \rightarrow \{ (q_4, Z_0, X) \}$
- See “a” on Stack 2, then POP it
 $\partial (q_4, \Lambda, Z_0, a) \rightarrow \{ (q_5, Z_0, \epsilon) \}$
- Push everything back to Stack 1
 $\partial (q_5, \Lambda, X, a) \rightarrow \{ (q_5, aX, \epsilon) \}$
 $\partial (q_5, \Lambda, X, b) \rightarrow \{ (q_5, bX, \epsilon) \}$
- Done pushing. Go back to q_1
 $\partial (q_5, \Lambda, X, Z_0) \rightarrow \{ (q_1, X, Z_0) \}$
- See “b” on Stack 1, then POP it
 $\partial (q_1, \Lambda, b, X) \rightarrow \{ (q_2, \epsilon, X) \}$
- Push everything on stack 2
 $\partial (q_2, \Lambda, a, X) \rightarrow \{ (q_2, \epsilon, aX) \}$
 $\partial (q_2, \Lambda, b, X) \rightarrow \{ (q_2, \epsilon, bX) \}$
- Done pushing on stack 2
 $\partial (q_2, \Lambda, Z_0, X) \rightarrow \{ (q_6, Z_0, X) \}$
- See “b” on Stack 2, then POP it

$$\partial (q_6, \Lambda, Z_0, b) \rightarrow \{ (q_5, Z_0, \epsilon) \}$$

- Push everything back to Stack 1
 $\partial (q_5, \Lambda, X, a) \rightarrow \{ (q_5, aX, \epsilon) \}$
 $\partial (q_5, \Lambda, X, b) \rightarrow \{ (q_5, bX, \epsilon) \}$

- Done pushing. Go back to q_1
 $\partial (q_5, \Lambda, X, Z_0) \rightarrow \{ (q_1, X, Z_0) \}$

- Going to Final state
 $\partial (q_1, \Lambda, Z_0, X) \rightarrow \{ (q_f, Z_0, X) \}$ ***FINAL**

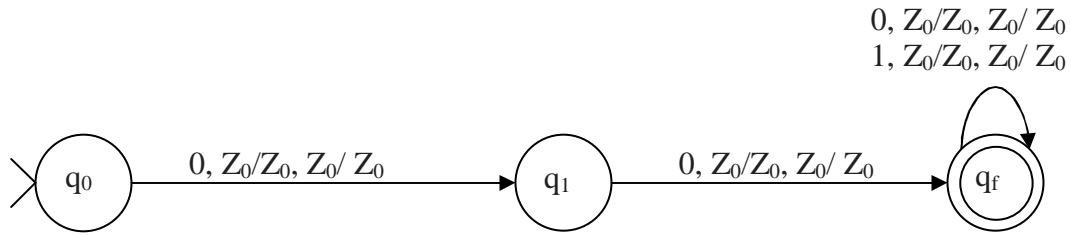


b) $L = \{ 00(0+1)^* \}$

This is an example of a Regular Language.

$$M = (\{q_0, q_1, q_f\}, \{0, 1\}, \{Z_0\}, \partial, q_0, Z_0, \{q_f\}, 2)$$

- $\partial (q_0, 0, Z_0, Z_0) \rightarrow \{ (q_1, Z_0, Z_0) \}$
- $\partial (q_1, 0, Z_0, Z_0) \rightarrow \{ (q_f, Z_0, Z_0) \}$ ***FINAL**
- $\partial (q_f, 0, Z_0, Z_0) \rightarrow \{ (q_f, Z_0, Z_0) \}$ ***FINAL**
- $\partial (q_f, 1, Z_0, Z_0) \rightarrow \{ (q_f, Z_0, Z_0) \}$ ***FINAL**

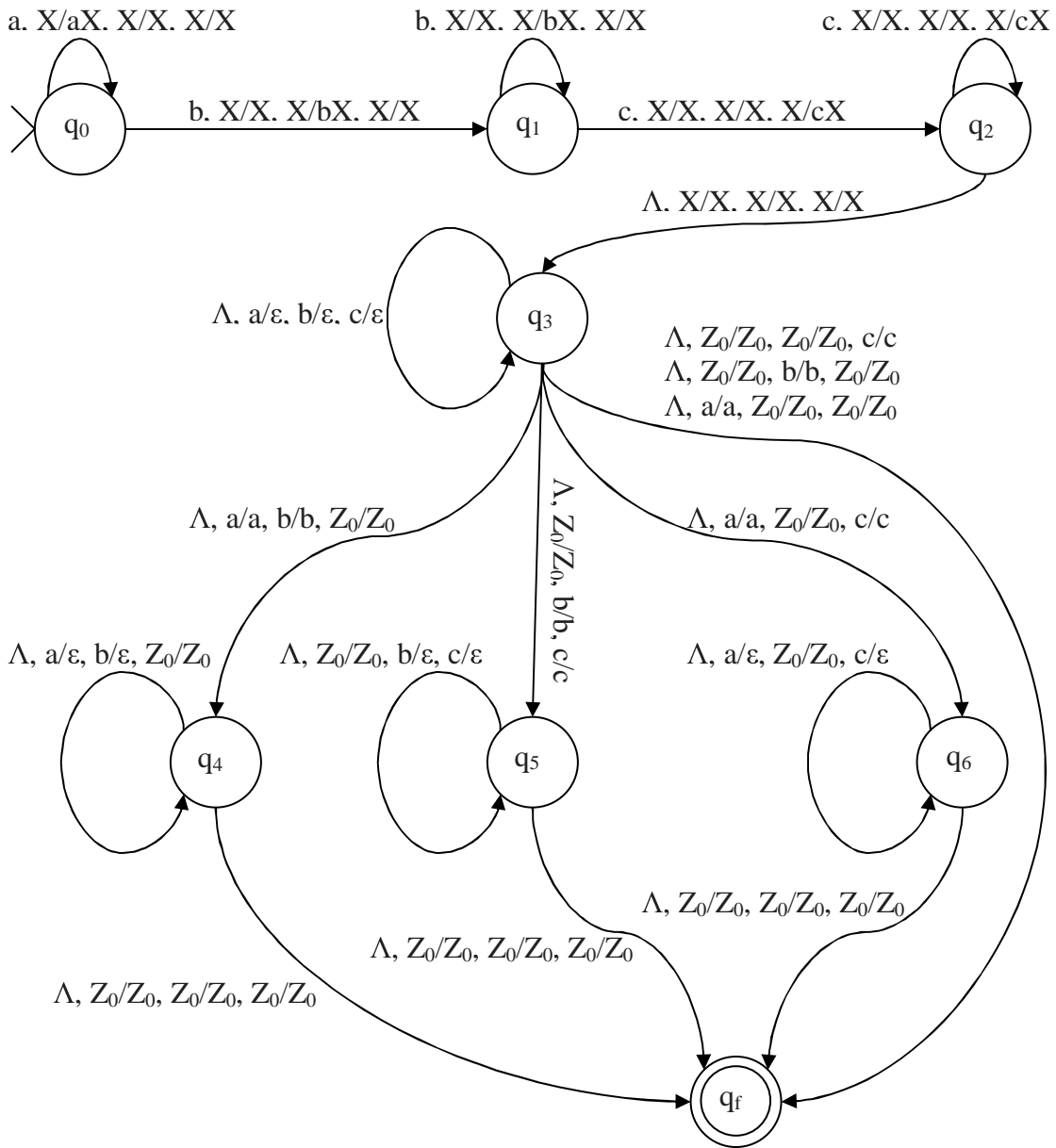


c) $L = \{ a^i b^j c^k \mid i = j \text{ or } j = k \text{ or } i = k \}$

This is an example of a Non-Context Free Language solved using a 3-stack Machine.

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_f\}, \{a, b, c\}, \{a, b, c, Z_0\}, \partial, q_0, Z_0, \{q_f\}, 3)$

- Push “a” on stack 1
 $\partial (q_0, a, X, X, X) \rightarrow \{ (q_0, aX, X, X) \}$
- Push “b” on stack 2
 $\partial (q_0, b, X, X, X) \rightarrow \{ (q_1, X, bX, X) \}$
 $\partial (q_1, b, X, X, X) \rightarrow \{ (q_1, X, bX, X) \}$
- Push “c” on stack 3
 $\partial (q_1, c, X, X, X) \rightarrow \{ (q_2, X, X, cX) \}$
 $\partial (q_2, c, X, X, X) \rightarrow \{ (q_2, X, X, cX) \}$
- Done Pushing
 $\partial (q_2, \Lambda, X, X, X) \rightarrow \{ (q_3, X, X, X) \}$
- Start Popping all three stacks simultaneously
 $\partial (q_3, \Lambda, a, b, c) \rightarrow \{ (q_3, \epsilon, \epsilon, \epsilon) \}$
 $\partial (q_3, \Lambda, a, b, Z_0) \rightarrow \{ (q_4, a, b, Z_0) \}$
 $\partial (q_3, \Lambda, Z_0, Z_0, c) \rightarrow \{ (q_f, Z_0, Z_0, c) \}$ *FINAL
 $\partial (q_3, \Lambda, Z_0, b, c) \rightarrow \{ (q_5, Z_0, b, c) \}$
 $\partial (q_3, \Lambda, a, Z_0, Z_0) \rightarrow \{ (q_f, a, Z_0, Z_0) \}$ *FINAL
 $\partial (q_3, \Lambda, a, Z_0, c) \rightarrow \{ (q_6, a, Z_0, c) \}$
 $\partial (q_3, \Lambda, Z_0, b, Z_0) \rightarrow \{ (q_f, Z_0, b, Z_0) \}$ *FINAL
 $\partial (q_4, \Lambda, a, b, Z_0) \rightarrow \{ (q_4, \epsilon, \epsilon, Z_0) \}$
 $\partial (q_4, \Lambda, Z_0, Z_0, Z_0) \rightarrow \{ (q_f, Z_0, Z_0, Z_0) \}$ *FINAL
 $\partial (q_5, \Lambda, Z_0, b, c) \rightarrow \{ (q_5, Z_0, \epsilon, \epsilon) \}$
 $\partial (q_5, \Lambda, Z_0, Z_0, Z_0) \rightarrow \{ (q_f, Z_0, Z_0, Z_0) \}$ *FINAL
 $\partial (q_6, \Lambda, a, Z_0, c) \rightarrow \{ (q_6, \epsilon, Z_0, \epsilon) \}$
 $\partial (q_6, \Lambda, Z_0, Z_0, Z_0) \rightarrow \{ (q_f, Z_0, Z_0, Z_0) \}$ *FINAL



5. Non-deterministic vs. deterministic machines

Every non-deterministic multi-stack machine M_N can be converted to a deterministic multi-stack machine M_D by using the following construction. M_D will have 2 pairs of stacks, where each pair simulates a tape. The first pair simulates a queue which stores the sequence of IDs of M_N that need to be processed, while the second pair is used for scratch work. At the beginning of execution, M_D will have the starting ID of M_N in the queue. Also, M_D will have a mark at the current ID that it is looking at.

To process the current ID, M_D does the following:

1. M_D examines the state and the scanned symbol of the current ID. Built into the finite control of M_D is the knowledge of what choices of move M_N has for each state and symbol. If the state in the current ID is accepting, then M_D accepts and simulates M_N no further.
2. However, if the state is not accepting and the state-symbol combination has k moves, then M_D uses its second pair of stacks to copy the ID and then make k copies of that ID at the end of the sequence of IDs on the first pair of stacks.
3. M_D modifies each of those k IDs according to a different one of k choices of move that M_N has from its current ID.
4. M_D returns to the mark, current ID, erases the mark, and moves the mark to the next ID to the right. The cycle then repeats with step 1.

Since a non-deterministic 0-stack machine is equivalent to a NFSA and NFSAs are equivalent to DFSAs, then non-deterministic 0-stack machines are as powerful as deterministic 0-stack machines.

Since a non-deterministic 1-stack machine is equivalent to a NPDA and NPDAs are more powerful than DPDA's, then non-deterministic 1-stack machines are more powerful than deterministic 1-stack machines.

Since a non-deterministic 2 or more-stack machine is equivalent to a deterministic 4-stack machine, then non-deterministic 2 or more-stack machines are as powerful as deterministic 4-stack machines.

6. Quiz questions

a) L_{wcw}

Create a multi-stack machine that recognizes the Non-Context Free

Language $L_{wcw} = \{wcw \mid w \in L((0+1)^+)\}$.

b) $L = \{a^n b^n c^n \mid n \geq 1\}$

Create a machine that recognizes this language. This is a Non-Context Free Language.

7. Answers to quiz questions

KEY: Λ - means no input
 ϵ - means pop the stack
 X - means any stack element
 q_f - means final state

a) L_{wcw}

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_f\}, \{0, 1, c\}, \{0, 1, Z_0\}, \partial, q_0, Z_0, \{q_f\}, 2)$

- Push everything on stack 1
 $\partial (q_0, 1, X, Z_0) \rightarrow \{ (q_0, 1X, Z_0) \}$
 $\partial (q_0, 0, X, Z_0) \rightarrow \{ (q_0, 0X, Z_0) \}$

- $\partial (q_0, \Lambda, X, X) \rightarrow \{ (q_1, X, X) \}$
- If see “0” on stack 1
 $\partial (q_1, \Lambda, 0, X) \rightarrow \{ (q_3, \epsilon, X) \}$
 - Push everything on stack 2 until we see a “c”
 $\partial (q_3, \Lambda, 0, X) \rightarrow \{ (q_3, \epsilon, 0X) \}$
 $\partial (q_3, \Lambda, 1, X) \rightarrow \{ (q_3, \epsilon, 1X) \}$
 $\partial (q_3, \Lambda, c, X) \rightarrow \{ (q_4, \epsilon, cX) \}$
 - Saw “c”, so check if next letter is a “0”
 $\partial (q_4, \Lambda, 0, X) \rightarrow \{ (q_5, \epsilon, X) \}$
 - Pushing everything back to stack 1
 $\partial (q_5, \Lambda, X, c) \rightarrow \{ (q_5, cX, \epsilon) \}$
 $\partial (q_5, \Lambda, X, 0) \rightarrow \{ (q_5, 0X, \epsilon) \}$
 $\partial (q_5, \Lambda, X, 1) \rightarrow \{ (q_5, 1X, \epsilon) \}$
 $\partial (q_5, \Lambda, X, Z_0) \rightarrow \{ (q_1, X, Z_0) \}$
 - If see “1” on stack 1
 $\partial (q_1, \Lambda, 1, X) \rightarrow \{ (q_2, \epsilon, X) \}$
 - Push everything on stack 2 until we see a “c”
 $\partial (q_2, \Lambda, 0, X) \rightarrow \{ (q_2, \epsilon, 0X) \}$
 $\partial (q_2, \Lambda, 1, X) \rightarrow \{ (q_2, \epsilon, 1X) \}$
 $\partial (q_2, \Lambda, c, X) \rightarrow \{ (q_6, \epsilon, cX) \}$
 - Saw “c”, so check if next letter is a “1”
 $\partial (q_6, \Lambda, 1, X) \rightarrow \{ (q_5, \epsilon, X) \}$
 - Pushing everything back to stack 1
 $\partial (q_5, \Lambda, X, c) \rightarrow \{ (q_5, cX, \epsilon) \}$
 $\partial (q_5, \Lambda, X, 0) \rightarrow \{ (q_5, 0X, \epsilon) \}$
 $\partial (q_5, \Lambda, X, 1) \rightarrow \{ (q_5, 1X, \epsilon) \}$
 $\partial (q_5, \Lambda, X, Z_0) \rightarrow \{ (q_1, X, Z_0) \}$
 - Going to the final state
 $\partial (q_1, \Lambda, c, X) \rightarrow \{ (q_f, \epsilon, X) \}$ ***FINAL**

b) $L = \{ a^n b^n c^n \mid n \geq 1 \}$

$M = (\{q_0, q_1, q_2, q_3, q_f\}, \{a, b, c\}, \{a, b, c, Z_0\}, \partial, q_0, Z_0, \{q_f\}, 3)$

- Push “a” on stack 1
 $\partial (q_0, a, X, X, X) \rightarrow \{ (q_0, aX, X, X) \}$
- Push “b” on stack 2
 $\partial (q_0, b, X, X, X) \rightarrow \{ (q_1, X, bX, X) \}$
 $\partial (q_1, b, X, X, X) \rightarrow \{ (q_1, X, bX, X) \}$
- Push “c” on stack 3
 $\partial (q_1, c, X, X, X) \rightarrow \{ (q_2, X, X, cX) \}$
 $\partial (q_2, c, X, X, X) \rightarrow \{ (q_2, X, X, cX) \}$
- Done Pushing
 $\partial (q_2, \Lambda, X, X, X) \rightarrow \{ (q_3, X, X, X) \}$
- Start Popping all three stacks simultaneously
 $\partial (q_3, \Lambda, a, b, c) \rightarrow \{ (q_3, \epsilon, \epsilon, \epsilon) \}$

 $\partial (q_3, \Lambda, Z_0, Z_0, Z_0) \rightarrow \{ (q_f, Z_0, Z_0, Z_0) \}$ ***FINAL**