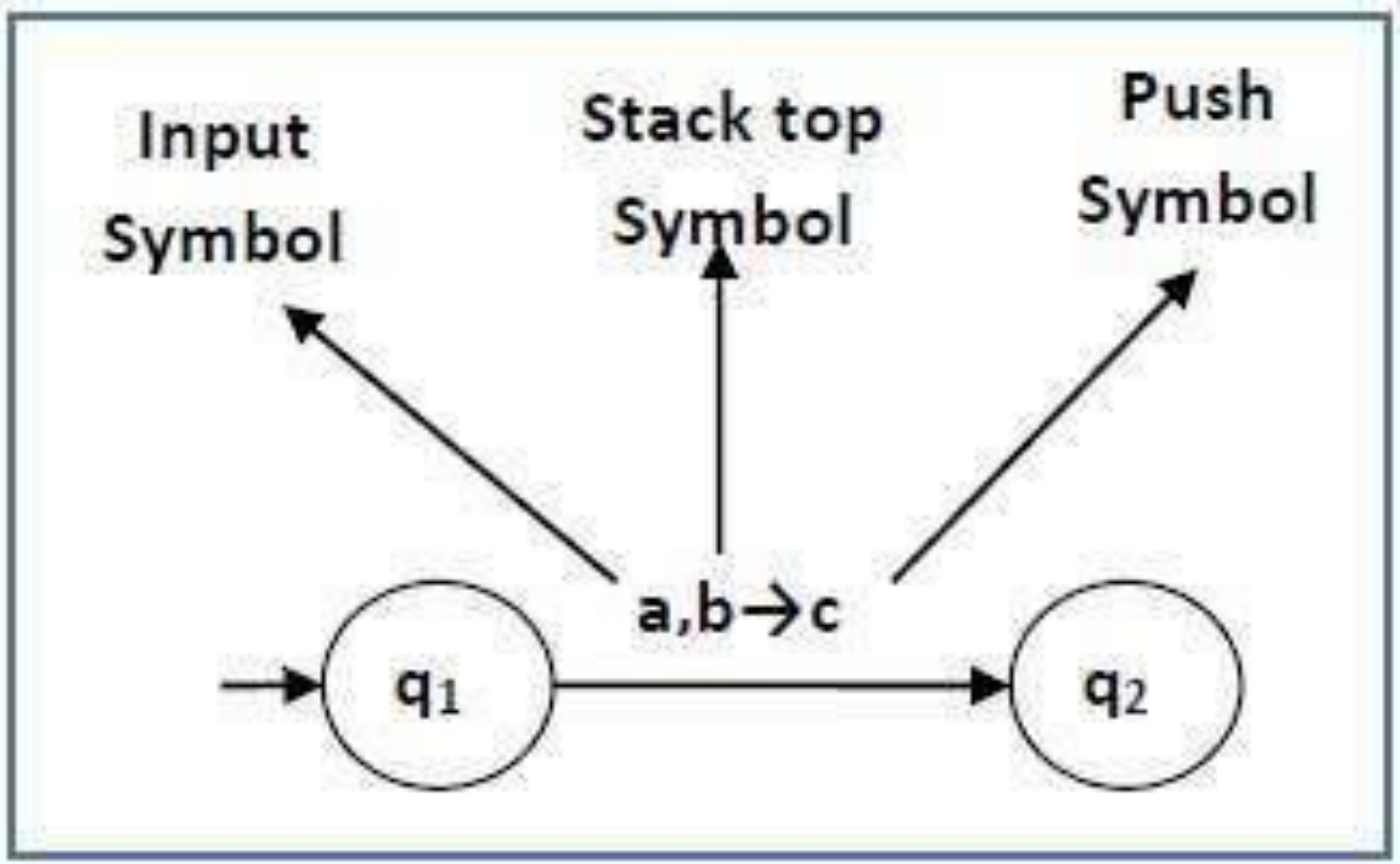
DEFINITION 2.13

A *pushdown automaton* is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \in Q$ is the set of accept states.



This means at state $\mathbf{q_1}$, if we encounter an input string **'a'** and top symbol of the stack is **'b'**, then we pop **'b'**, push **'c'** on top of the stack and move to state $\mathbf{q_2}$.

EXAMPLES OF PUSHDOWN AUTOMATA

EXAMPLE 2.14

The following is the formal description of the PDA that recognizes the language $\{0^n1^n \mid n \geq 0\}$. Let M1 be $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where

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

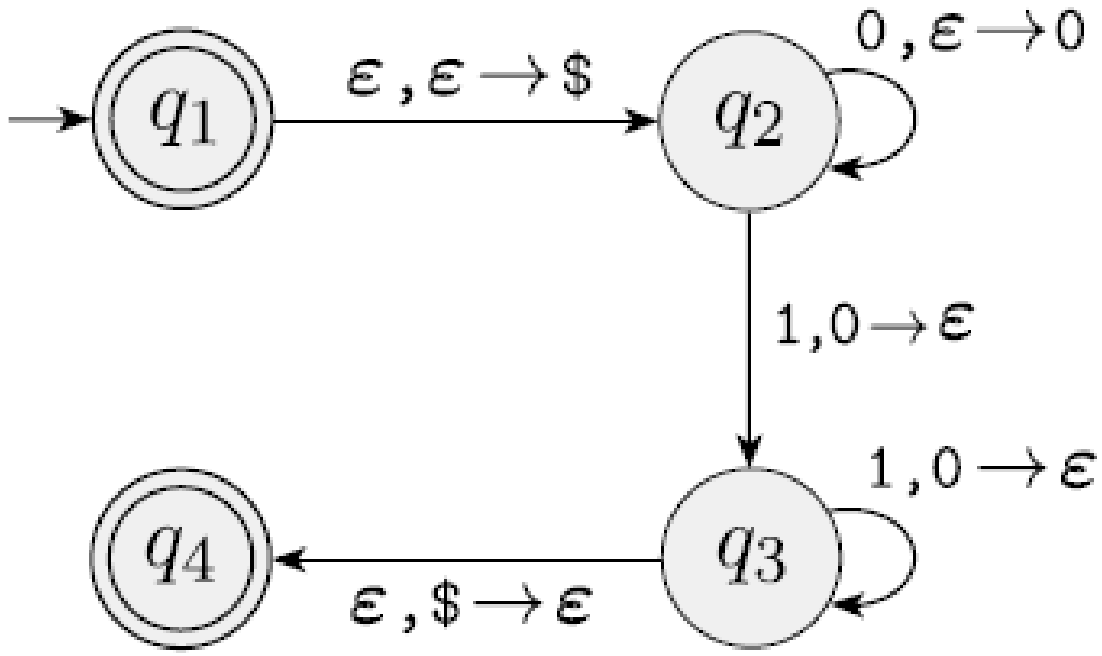
$\Sigma = \{0,1\}$

$\Gamma = \{o, \$\}$

$F = \{q_1, q_4\}$, and

δ **is given by the following table, wherein blank entries signify** φ.

Input:	0			1			ε		
Stack:	0	$	ε	0	$	ε	0	$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									



State diagram for the PDA M_1 that recognizes $\{0^n 1^n \mid n \geq 0\}$

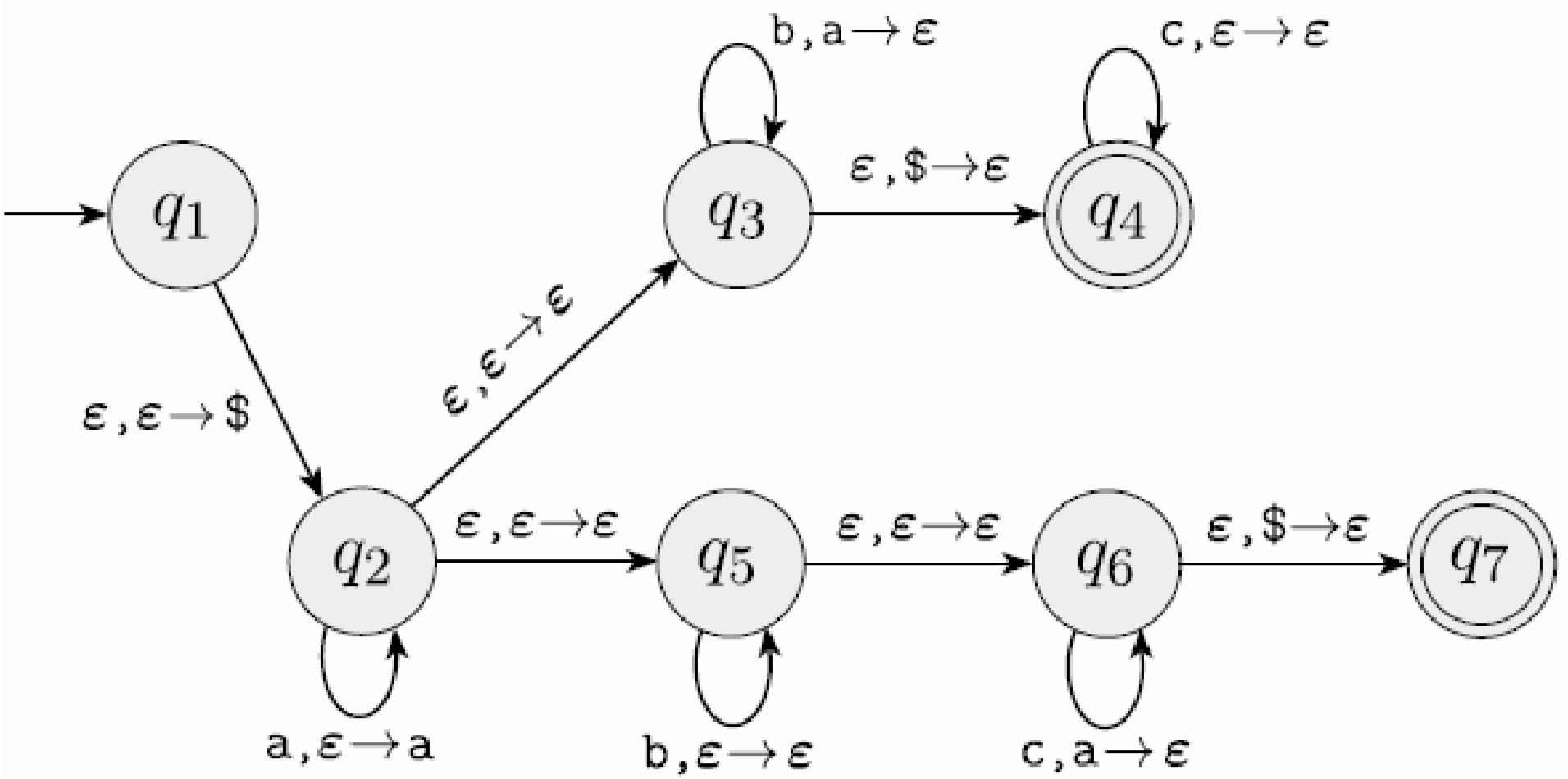
The formal definition of a PDA contains no explicit mechanism to allow the PDA to test for an empty stack. This PDA is able to get the same effect by initially placing a special symbol $ on the stack. Then if it ever sees the $ again, it knows that the stack effectively is empty.

EXAMPLE

This example illustrates a pushdown automaton that recognizes the language $\{a^i b^j c^k | \ i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$.

Informally, the PDA for this language works by first reading and pushing the a's. When the a's are done, the machine has all of them on the stack so that it can match, them with either the b's or the c's.

Think of the machine as having two branches of its nondeterminism, one for each possible guess. If either of themmatches, that branch accepts and the entire machine accepts.



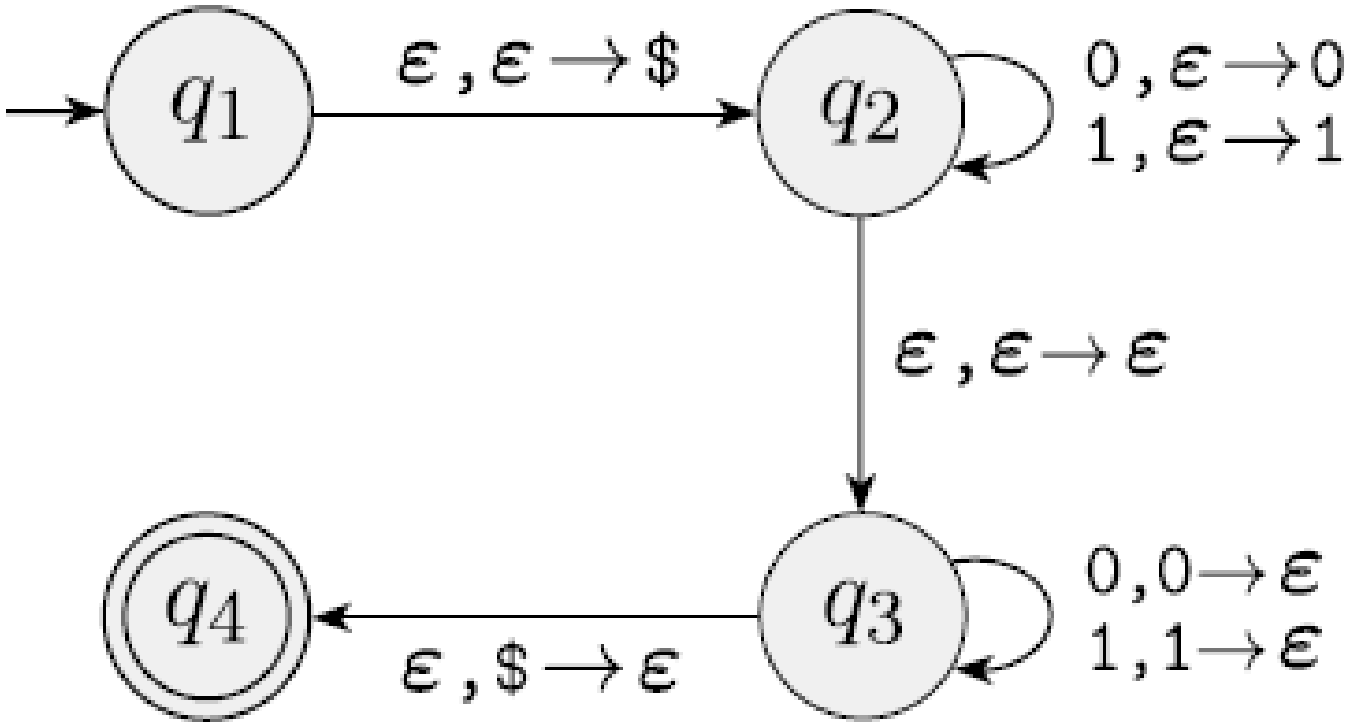
State diagram for PDA M_2 that recognizes $\{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$

EXAMPLE

give a PDA M3 recognizing the language $\{ww^R | w \in \{0,1\}\}$.

Recall that w^R means w written backwards. The informal description and state diagram of the PDA follow.

Begin by pushing the symbols that are read onto the stack. At each point, nondeterministically guess that the middle of the string has been reached and then change into popping off the stack for each symbol read, checking to see that they are the same. If they were always the same symbol and the stack empties at the same time as the input is finished, accept; otherwise reject.

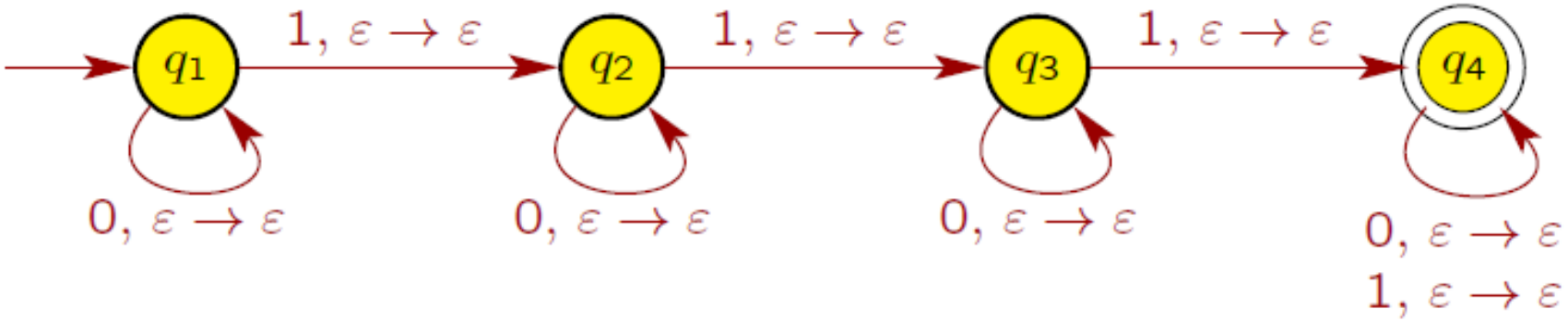


State diagram for the PDA M_3 that recognizes $\{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$

Example:

Give pushdown automata that recognize the following language; A = { w ∈ {0, 1} * | w contains at least three 1s }

Ans.:

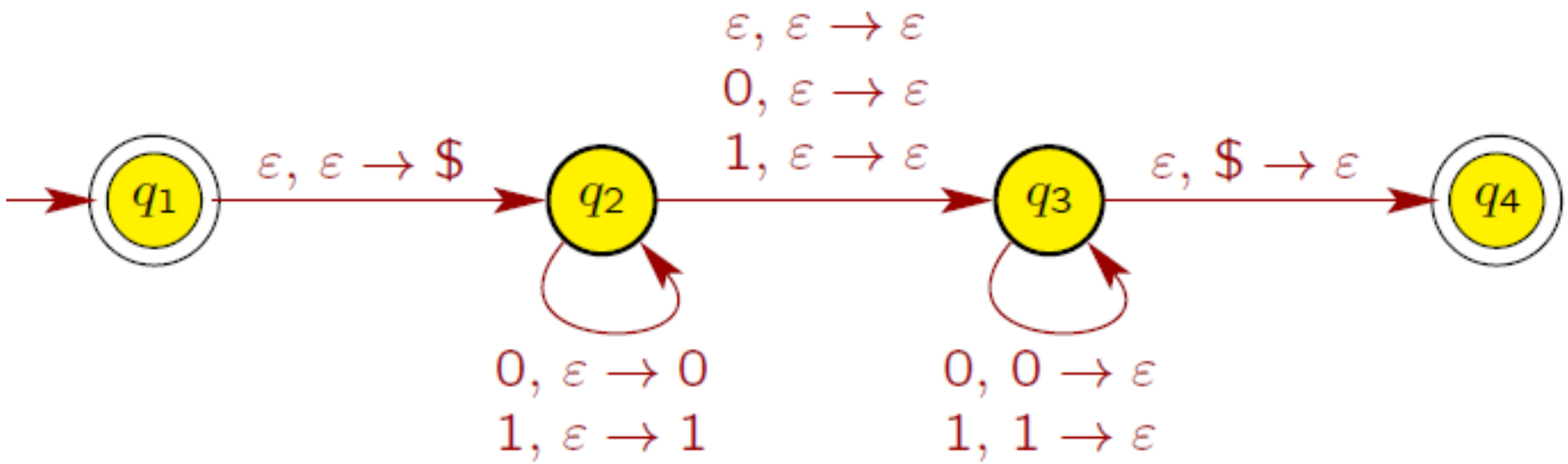


Note that A is a regular language, so the language has a DFA. We can easily convert the DFA into a PDA by using the same states and transitions and never push nor pop anything from the stack.

Example:

Give pushdown automata that recognize the following language; C = {w ∈ {0, 1}* | w = w^R }

Ans.:

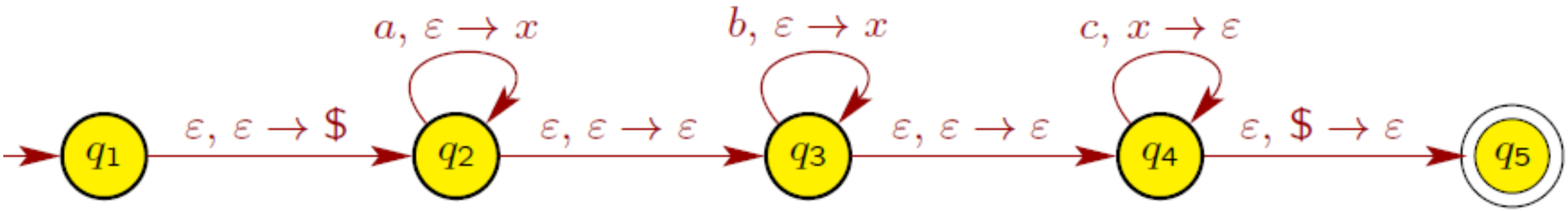


The length of a string w ∈ C can be either even or odd. If it's even, then there is no middle symbol in w, so the first half of w is pushed on the stack, we move from q2 to q3 without reading, pushing, or popping anything, and then match the second half of w to the first half in reverse order by popping the stack. If the length of w is odd, then there is a middle symbol in w, and the description of the PDA in part (b) applies.

Example:

Give pushdown automata that recognize the following language; E = { $a^i b^j c^k$ | i, j, k ≥ 0 and i+j = k }

Ans.:



For every a and b read in the first part of the string, the PDA pushes an x onto the stack. Then it must read a c for each x popped off the stack.