Autómatas con Pila

Un Autómata con Pila (AP) es Autómata Finito al que se le ha incorporado una memoria tipo LIFO (Last In, First Out).

Los componentes de un AP son:

$$\mathbf{AP = (\Sigma_E, \Gamma, Q, q_0, \#, A, \mathit{f})}$$

Posee los mismos componentes de un AF reconocedor más dos nuevos componentes:

Γ: alfabeto de símbolos de pila

#: Símbolo de referencia de pila, # ∈ Γ y # ∉ Σ_E

La función de transición queda definida:

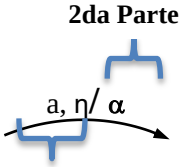
$$\mathbf{\mathit{f}: Q \times (\Sigma_E \cup \{\lambda\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)}$$

El comportamiento del autómata en un intervalo de tiempo dado queda determinado por tres argumentos: *i*) el estado actual, *ii*) el símbolo de entrada y *iii*) el símbolo leído de tope de la pila.

A partir de estos tres argumentos queda definido: *i*) el próximo estado del autómata y *ii*) la acción sobre la pila. Esta acción consiste en la grabación de varios símbolos, la de uno solo o de ninguno (**λ**), y esto último corresponde al caso en que la pila es descargada, es decir leída y no grabada.

Representación de las transiciones

Las transiciones en un AP en un grafo se definen de la siguiente manera:

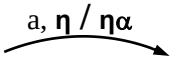


Con **a** ∈ Σ_E, η∈ **Γ** y **α**∈$\mathbf{\Gamma^*}$

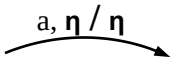
1era parte: se corresponde a la lectura del símbolo de la cadena de entrada (Σ_E) o λ y de un símbolo de la memoria de la pila (**Γ**), son dos lecturas no obligadas por ejemplo, en el caso de Analizadores Sintácticos que puede ignorar la lectura de una cadena o de la memoria de pila, en el desarrollo de nuestros ejercicios prácticos de construcción de AP realizamos las dos lecturas, cada vez que se realiza la lectura de la memoria de Pila, funciona de la siguiente manera: *por leer la pila, el símbolo leído automáticamente sale de la memoria.*

2da parte: se corresponde con la acción que se realiza en la memoria que puede ser que se grabe un símbolo, varios o ninguno, los mismos se representan a continuación:

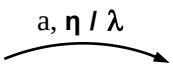
- *grabar más de un símbolo:* esto puede suceder cuando el símbolo que leímos de la memoria salió de la misma y no queremos perderlo, entonces, volvemos a guardar en la memoria el símbolo leído de la pila y un nuevo símbolo que queremos guardar en la memoria. Esta operación lo llamamos *apilar.*



- *Grabar un símbolo:* solo guardamos en la memoria un solo símbolo, podemos decir que *apilamos* un solo símbolo y ese puede ser el mismo que leímos de la memoria u otro distinto



- *No grabar símbolos:* si queremos ir sacando símbolos de la pila esta es la función correspondiente y para ello en esta 2da parte simplemente indicamos con el símbolo lambda (λ) y significa que no guardamos símbolos en la pila, por lo tanto el símbolo que se leyó de la pila en la 1era parte, salió de la memoria. A esta función la denominamos *desapilar*.



***Nota**: Podría ocurrir que no se lea el símbolo de la cadena de entrada **(λ)** pero en estos ejercicios siempre leeremos un símbolo en cada transición.*

Aceptación de Lenguajes de un AP

La aceptación en un AP puede darse de tres maneras:

- ✓ por vaciado de pila
- ✓ por estado de aceptación
- ✓ por vaciado de pila y estado de aceptación

***Nota:** En los ejercicios prácticos aplicamos este último.*

Ejercicio 1

Construir un *AP* por cada ítem, que verifique si los paréntesis y/o corchetes se encuentran balanceados, es decir que por cada símbolo de apertura debe haber uno de cierre.

a) (( ))
b) (( )) ((( )))
c) [( ) (( ))]

Cabe aclarar que cada punto del ejercicio es un AP distinto

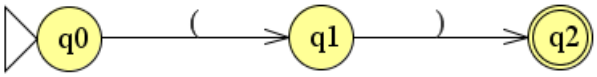
Solución Ejercicio 1 – a) (( ))

Como 1er Paso debemos analizar las cadenas que debe reconocer el AP es decir el Lenguaje como lo hemos explicado en AF, siempre que construimos una máquina abstracta para reconocer uno. En este caso el AP debe reconocer cadenas formadas por paréntesis pero que a su vez estén balanceados, qué significa? Que cada cadena debe tener la misma cantidad de paréntesis de abertura que paréntesis de cierre, además, que la sintaxis sea la correcta en el sentido de que siempre las cadenas comienzan con paréntesis de abertura y luego le siguen paréntesis de cierre, no admite otra estructura, entonces podemos decir que algunas cadenas del lenguaje son:

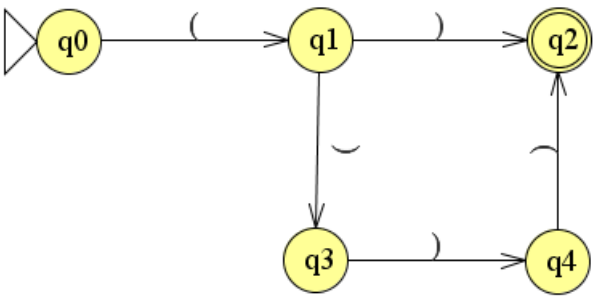
L = { ( ), (( )), ((( ))), (((( )))), …}

Recuerden que siempre deben comenzar por plantear la cadena más corta ya que nos enfrentamos a cadenas que no tienen una longitud fija.

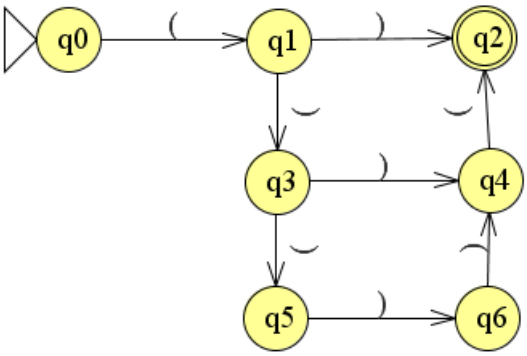
Hasta el momento solo conocen la construcción de AF, tratemos de construir este ejercicio como uno. Sabemos que las transiciones entre estados en un AF es la única manera de saber que se está leyendo de la cadena de entrada ya que la máquina no tiene memoria, vamos a construir un AF que reconozca la cadena más corta para empezar:



si la cadena aumenta de longitud por ejemplo (( )) recordando que debemos tener en cuenta el balanceo:



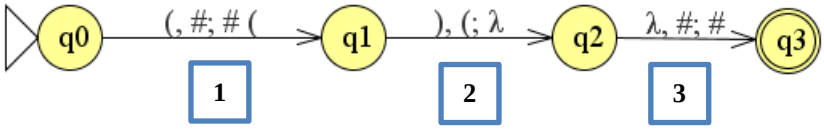
Aumentamos en un par más de paréntesis:



Cada vez que la cadena aumenta en longitud debemos aumentar la cantidad de estados, en conclusión un AF nunca va a reconocer este tipo de lenguaje.

La solución está en la memoria de la pila. ¿De qué manera? ¿Cómo me sirve? Es muy simple: vamos a guardar en memoria todos los paréntesis de apertura que presente cada cadena, ¿hasta cuándo? Hasta que se lea el primer paréntesis de cierre, ¿para qué? Para comparar cada paréntesis de cierre con cada paréntesis de apertura almacenado en la pila. Si al terminar la lectura de la cadena la pila está vacía, entonces hemos leído la misma cantidad de paréntesis de cierre que paréntesis de apertura.

Comencemos a construir un AP que reconozca la cadena más corta: ( )

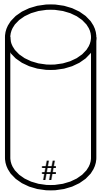


1 f (q0, (, #) = (q1, #( ) Primera transición donde se apila o almacena por primera vez en la memoria indicado en el grafo con el rótulo:

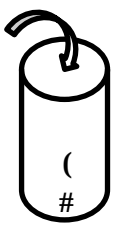
(, # / # (

En la 1era parte del rótulo se lee en la cadena de entrada un **(** y la pila tiene el tope de pila # (que indica que la pila está vacía) y en la 2da parte que es la acción sobre la pila, se almacena nuevamente el tope de pila (recordar que al leer en la pila el símbolo leído se desapila) y se agrega el símbolo nuevo que en este caso es un **(**, simulando la memoria de la pila podemos graficar:

En el estado q0:

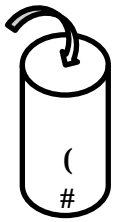


En el estado q1:



2 f (q1,), ( ) = (q2, λ) en esta 2da transición se ve realmente el objetivo del uso de la memoria: al leer un símbolo de paréntesis de cierre y la pila tener almacenado un paréntesis de apertura, se desapila, de esta manera se verifica que por cada paréntesis de cierre, anteriormente se ha leído un paréntesis de apertura

En el estado q1:

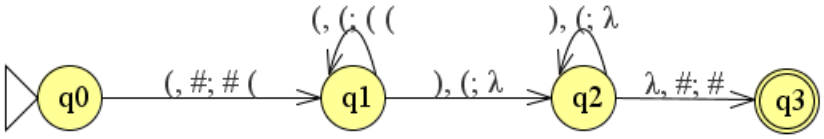


En el estado q2:

3 f (q2, λ, #) = (q3, #) finaliza el AP por vaciado de pila y por estado de aceptación.

Este AP construido reconoce el balanceo de la cadena de menor longitud del lenguaje que debe reconocer, mejoremos el mismo para que reconozca un lenguaje de cadenas de paréntesis de abertura y de cierre de cualquier longitud donde se verifique el balanceo de cada cadena.

A continuación se presenta una solución del ejercicio 1 a, se pueden construir con más o menos estados otras soluciones correctas para este ejercicio, además, en un mismo estado puedo almacenar símbolos en la pila como también sacar de la misma, nuestro ejemplo tiene las funcionalidades separadas para que se puedan visualizar de mejor manera:



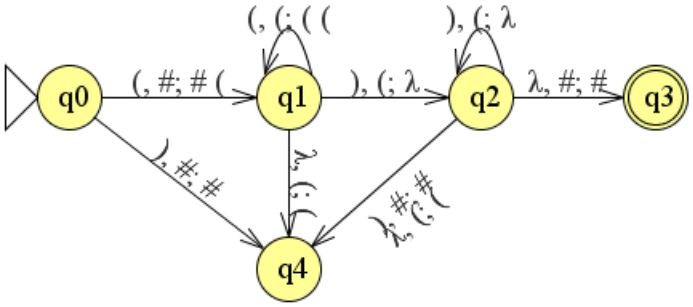
AP = ({ (,) }, { (, # }, { q0, q1, q2, q3 }, q0, #, {q3}, ***f***)

Como las cadenas no tienen longitud fija en el estado q1 se almacenan todos los paréntesis de apertura que se lean de cada cadena, cuando se lee el primer paréntesis de cierre transita al estado q2 que es donde se compara cada paréntesis de cierre con cada paréntesis de apertura almacenado y se saca dicho símbolo de la pila, finaliza cuando la pila está vacía transitando al estado de aceptación q3.

NOTA: Recomendamos construir los AP de los ejercicios en JFLAP recordando que la 2da parte de la transición que corresponde a la acción sobre la pila se debe expresar de manera inversa por ejemplo, en el 1er rótulo: ***(, # / # (*** *en JFLAP lo realiza:* ***(, # / (# . Además el símbolo de pila vacía es Z en lugar de #.*** *El simulador utiliza como símbolo de fondo de pila a* ***z.***

A continuación se presentan posibles soluciones (no necesariamente únicas) a los ejercicios propuestos:

a) (( ))



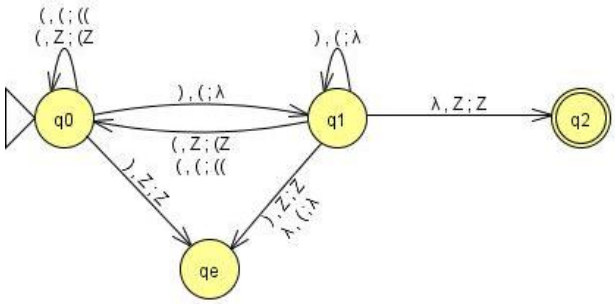
N°	Q	Cadena	Pila
1	q0	((( )))	#
2	q1	(( )))	# (
3	q1	( )))	# ((
4	q1	)))	# (((
5	q2	))	# ((
6	q2	)	# (
7	q2	λ	#
8	q3	λ	#

Tabla Operativa

AP = ({ (,) }, { (, # }, { q0, q1, q2, qe }, q0, Z, { q2 }, ***f***)

NOTA: Las soluciones que se presentan a continuación han sido creadas con el simulador JFLAP. En estos ejercicios sólo se considera necesario destacar los errores de emparejamiento.

b) (( )) ((( )))

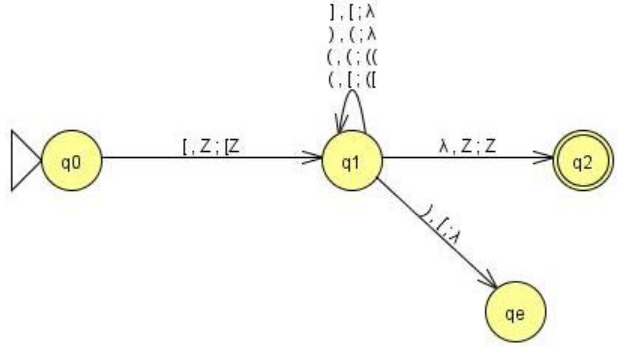


N°	Q	Cadena	Pila
1	q0	(( ) ( ))	z
2	q0	( ) ( ))	z (
3	q0	) ( ))	z ((
4	q1	( ))	z (
5	q0	))	z ((
6	q1	)	z (
7	q1	λ	z
8	q2	λ	z

Tabla Operativa

AP = ({ (,) }, { (, Z }, { q0, q1, q2, qe }, q0, Z { q2 }, ***f***)

c) [( ) (( ))]



N°	Q	Cadena	Pila
1	q0	[( ) ( )]	z
2	q1	( ) ( )]	z [
3	q1	) ( )]	z [(
4	q1	( )]	z [
5	q1	)]	z [(
6	q1	]	z [
7	q1	λ	z
8	q2	λ	z

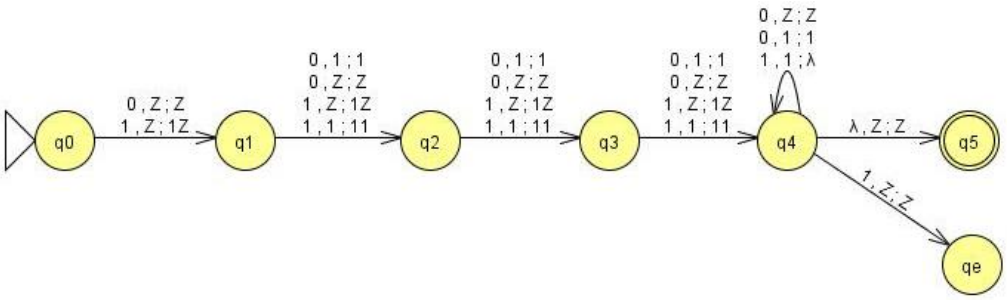
Tabla Operativa

AP = ({ [, (,),] }, { [, (, Z }, { q0, q1, q2, qe }, q0, Z, { q2 }, ***f***)

Ejercicio 2

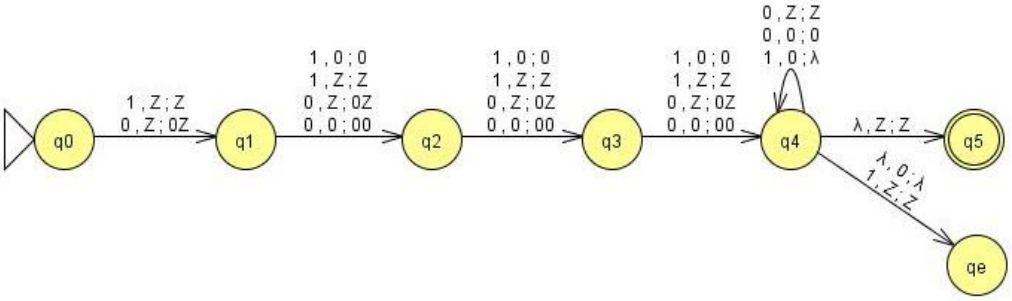
Para cada condición diseñar un AP que verifique si en un byte leído:

a) La cantidad de 1s en los cuatro primeros bits es la misma que en los cuatro últimos



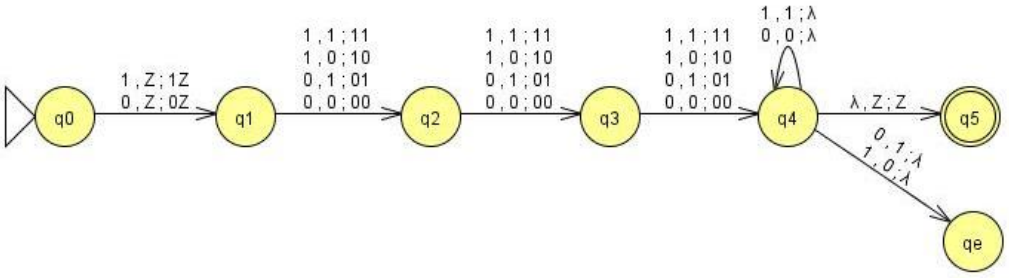
AP = ({ 0, 1 }, { 1, Z }, { q0, q1, q2, q3, q4, q5, qe }, q0, Z, { q5 }, ***f***)

b) La cantidad de 0s en los cuatro primeros bits es la misma que la cantidad de 1s en los cuatro últimos



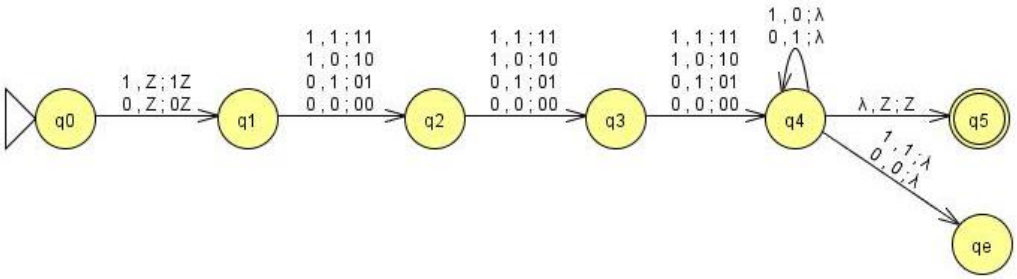
AP = ({ 0, 1 }, { 0, Z }, { q0, q1, q2, q3, q4, q5, qe }, q0, Z, { q5 }, ***f***)

c) Los cuatro primeros bits constituyen la imagen refleja de los cuatro últimos



AP = ({ 0, 1 }, { 0, 1, Z }, { q0, q1, q2, q3, q4, q5, qe }, q0, Z, { q5 }, ***f***)

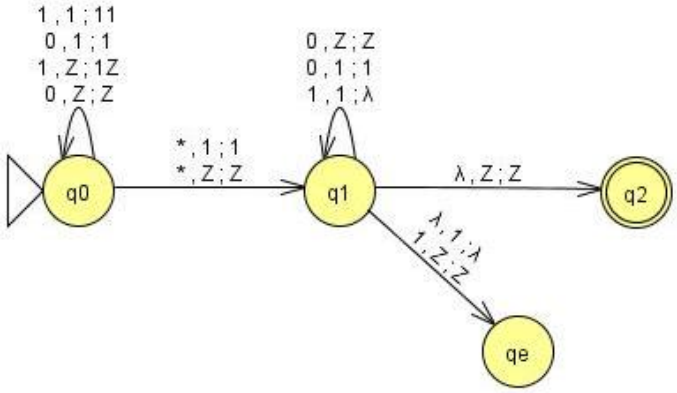
d) Los cuatro últimos bits constituyen la imagen refleja negada de los cuatro primeros



AP = ({ 0, 1 }, { 0, 1, Z }, { q0, q1, q2, q3, q4, q5, qe }, q0, Z, { q5 }, ***f***)

Ejercicio 3

Diseñar un AP que verifique si dos nibles leídos, separados por el símbolo ***** (asterisco) tienen ambos la misma cantidad de 1s.



N°	Q	Cadena	Pila
1	q0	0101*1001	z
2	q0	101*1001	z
3	q0	01*1001	z 1
4	q0	1*1001	z 1
5	q0	*1001	z 11
6	q1	1001	z 11
7	q1	001	z 1
8	q1	01	z 1
9	q1	1	z 1
10	q1	λ	z
11	q2	λ	z

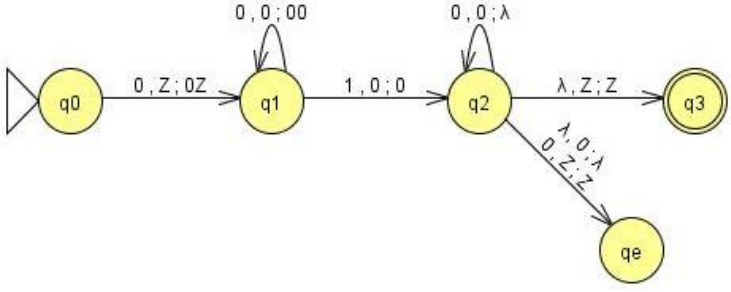
Tabla Operativa

AP = ({ 0, 1, ***** }, { 1, Z }, { q0, q1, q2, qe }, q0, Z, { q2 }, ***f***)

Ejercicio 4

Diseñar un AP para cada uno de los lenguajes definidos a continuación:

a) $L_1 = \{0^n 1 0^n / n \geq 1, \Sigma = \{0, 1\}\}$

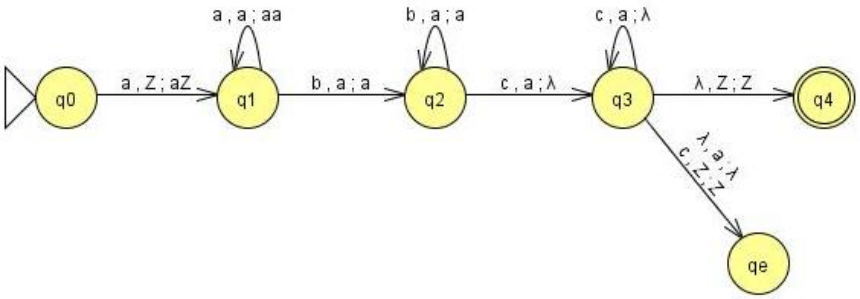


N°	Q	Cadena	Pila
1	q0	0001000	z
2	q1	001000	z0
3	q1	01000	z 00
4	q1	1000	z 000
5	q2	000	z 000
6	q2	00	z 00
7	q2	0	z 0
8	q2	λ	z
9	q3	λ	z

Tabla Operativa

AP = ({ 0, 1 }, { 0, Z }, { q0, q1, q2, q3, qe }, q0, Z, { q3 }, ***f***)

b) $L_2 = \{a^p b^n c^p / \; p \geq 1, n > 0, \Sigma = \{a, b, c\}\}$



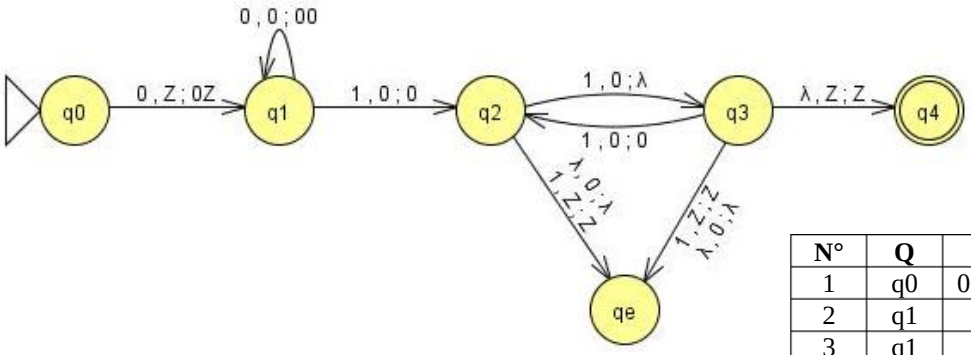
AP = ({ a, b, c }, { a, Z }, { q0, q1, q2, q3, q4, qe }, q0, Z, { q4 }, ***f***)

c) $L_3 = \{0^n 1^{2n} / \; n \geq 1, \Sigma = \{0, 1\}\}$

Podemos pensar en dos estrategias distintas para resolver este problema:

Estrategia 1: - Leo los 0s de la cadena de entrada y los apilo.

- Por cada par de 1s de la cadena de entrada: Cuando leo el primer 1 no modifico la pila y cuando leo el segundo 1 desapilo un 0.



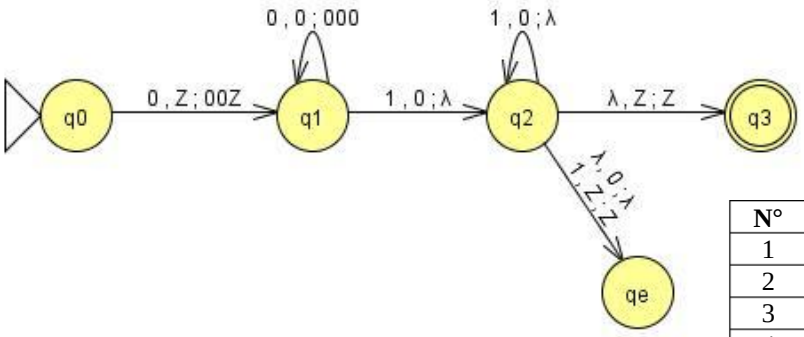
AP = ({ 0, 1}, { 0, Z }, { q0, q1, q2, q3, q4, qe }, q0, Z, { q4 }, ***f***)

N°	Q	Cadena	Pila
1	q0	001111	z
2	q1	01111	z0
3	q1	1111	z 00
4	q2	111	z 00
5	q3	11	z 0
6	q2	1	z 0
7	q3	λ	z
8	q4	λ	z

Tabla Operativa

Estrategia 2: - Por cada 0 que leo en la cadena de entrada, apilo 00.

- Por cada 1 que leo en la cadena de entrada, desapilo un 0.



AP = ({ 0, 1}, { 0, Z }, { q0, q1, q2, q3, qe },
q0, Z, { q3 }, ***f***)

N°	Q	Cadena	Pila
1	q0	001111	z
2	q1	01111	z00
3	q1	1111	z 0000
4	q2	111	z 000
5	q2	11	z 00
6	q2	1	z 0
7	q2	λ	z
8	q3	λ	z

Tabla Operativa