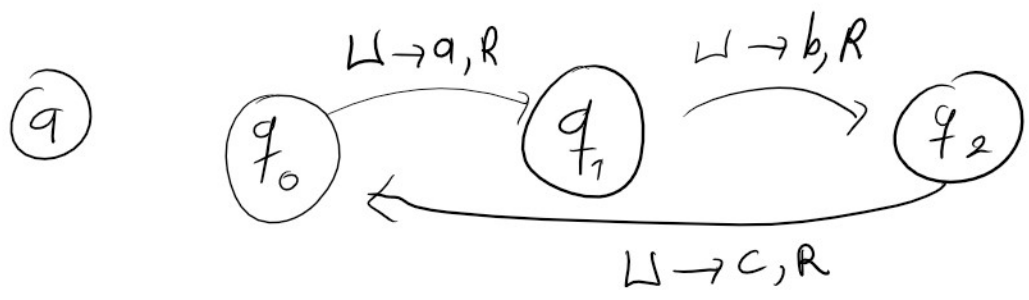
Taller 10

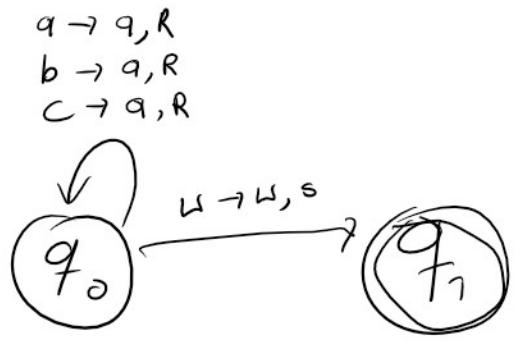
Monday, October 11, 2021 9:47 AM

EJERCICIO 1: Escriba el correspondiente diagrama de transiciones para los siguientes conjuntos de instrucciones:

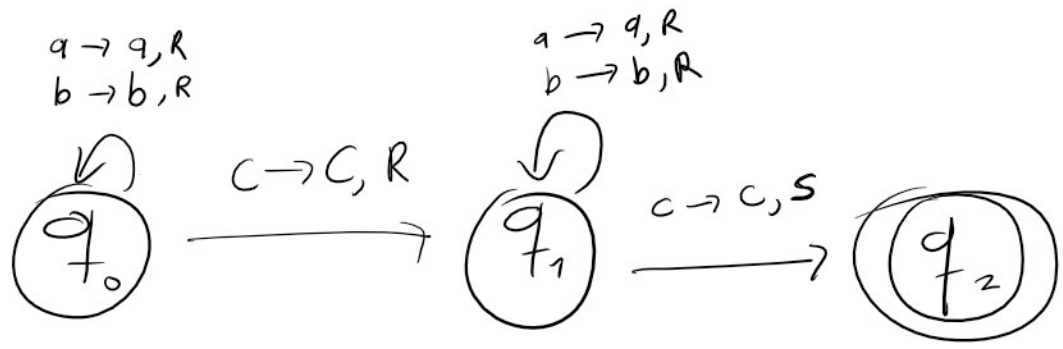
a) $\delta(q_0, \sqcup) = (q_1, a, R)$
$\delta(q_1, \sqcup) = (q_2, b, R)$
$\delta(q_2, \sqcup) = (q_0, c, R)$



b) $\delta(q_0, a) = (q_0, a, R)$
$\delta(q_0, b) = (q_0, a, R)$
$\delta(q_0, c) = (q_0, a, R)$
$\delta(q_0, \sqcup) = (q_1, \sqcup, S)$



c) $\delta(q_0, a) = (q_0, a, R)$
$\delta(q_0, b) = (q_0, b, R)$
$\delta(q_0, c) = (q_1, c, R)$
$\delta(q_1, a) = (q_1, a, R)$
$\delta(q_1, b) = (q_1, b, R)$
$\delta(q_1, c) = (q_2, c, S)$



EJERCICIO 2: Para cada una de las siguientes descripciones de una subrutina de una TM, escribir en el simulador el respectivo código que la implementa:

a) Asuma $\Sigma = \{a, b\}$ y $\Gamma = \{a, b, \sqcup\}$ y $w \in \Sigma^*$. La subrutina requerida es que, comenzando en $q_0 w$, la unidad de control busca la primera a en w.

```
//LOAD AN EXAMPLE TO TRY
//then load an input and click play

//Syntax:

//-------CONFIGURATION
name:  taller10_ejercicio2
init: qInit
accept: qAccept

//-------DELTA FUNCTION:
//[current_state],[read_symbol]
//[new_state],[write_symbol],[>|<|-]

// < = left
// > = right
// - = hold
// use underscore for blank cells

//States and symbols are case-sensitive

//Load your code and click COMPILE.
//or load an example (top-right).
```

```
//se busca la primera a

qInit,b
buscar_a,b,>

qInit,a
qAccept,a,-

buscar_a,b
buscar_a,b,>

buscar_a,a
qAccept,a,-

buscar_a,_
qReject,_,-
```

//Load your code and click COMPILE.
//or load an example (top-right).

b) Asuma $\Sigma = \{a, b\}$ y $\Gamma = \{a, b, X, \sqcup\}$ y $w \in \Sigma^*$. La subrutina requerida es que, comenzando en $q_0 w$, la unidad de control busca la primera a en w, la cambia por una X y busca hacia la derecha la primera b que encuentre.

```
//LOAD AN EXAMPLE TO TRY
//then load an input and click play

//Syntax:

//-------CONFIGURATION
name:  taller10_ejercicio2
init: qInit
accept: qAccept

//-------DELTA FUNCTION:
//[current_state],[read_symbol]
//[new_state],[write_symbol],[>|<|-]

// < = left
// > = right
// - = hold
// use underscore for blank cells

//States and symbols are case-sensitive

//Load your code and click COMPILE.
//or load an example (top-right).
```

```
//se busca la primera a

qInit,b
buscar_a,b,>

qInit,a
buscar_b,X,>

buscar_a,b
buscar_a,b,>

buscar_a,a
buscar_b,X,>

//caso en que no se encuentra ninguna a
buscar_a,_
qReject,_,-

//rutina para buscar b
buscar_b, a
buscar_b, a, >

buscar_b, b
qAccept, b, -

//caso en que no se encuentra ninguna a
buscar_b,_
qReject,_,-
```

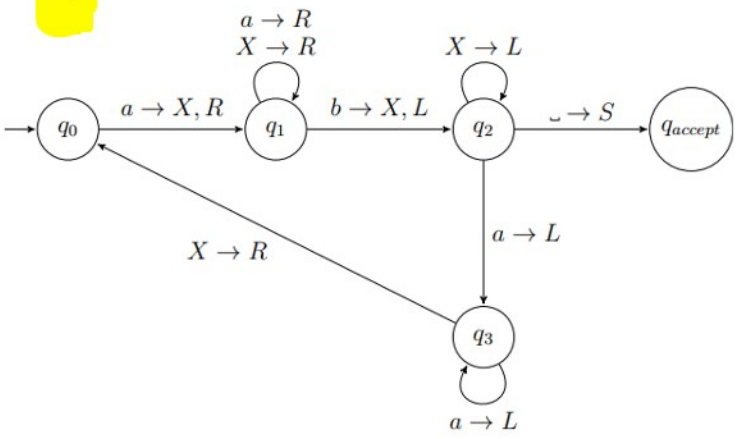
c) Asuma $\Sigma = \{a, b\}$ y $\Gamma = \{a, b, \sqcup\}$ y $w \in \Sigma^*$. La subrutina requerida es que, comenzando en $q_0 w$, la unidad de control busca la última b (de izquierda a derecha) en w.

```
//LOAD AN EXAMPLE TO TRY
//then load an input and click play

//Syntax:

//-------CONFIGURATION
name:  taller10_ejercicio2
init: qInit
accept: qAccept

//-------DELTA FUNCTION:
//[current_state],[read_symbol]
//[new_state],[write_symbol],[>|<|-]

// < = left
// > = right
// - = hold
// use underscore for blank cells

//States and symbols are case-sensitive

//Load your code and click COMPILE.
//or load an example (top-right).
```

```
//ir al final de la banda
qInit,b
qGoRight, b, >

qInit,a
qGoRight, a, >

// llega hasta el final de la banda y se regresa para
// buscar la b
qGoRight, a
qGoRight, a, >

qGoRight, b
qGoRight, b, >

qGoRight, _
qFind_b, _, <

// buscar la b

qFind_b, a
qFind_b, a, <

qFind_b, b
qAccept, b, -

qFind_b, _
qReject, _, -
```

EJERCICIO 3: Implemente en el simulador el algoritmo implementado por cada uno de los siguientes diagramas de transiciones (asuma $\Sigma = \{a, b\}$ y $\Gamma = \{a, b, X, \sqcup\}$). En cada caso, describa el funcionamiento de la unidad de control sobre la cinta:

a)



//q0
// reemplazo la 'a' por una 'X' en la cinta
q0, a
q1, X, >

//q1
// Mientras lea 'a' o 'X' sigo caminando hacia la derecha
// hasta encontrarme con una 'b', reemplazo la 'b' por 'X'
// y me devuelvo hacia la izquierda en la cinta
q1, a
q1, a, >

q1, X
q1, X, >

q1, b
q2, X, <

// q2
// voy llendo hacia la izquierda hasta que vea una 'a',
// si encuentro un espacio vacio paso a estado de aceptación
// y me detengo

q2, X
q2, X, <

q2, _
qAccept, S, -

q2, a
q3, a, <

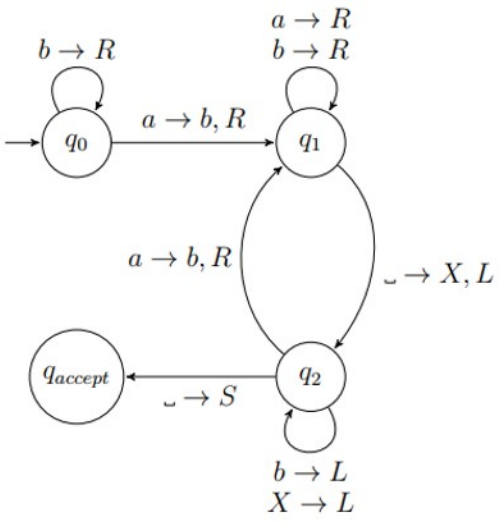
//q3
// si veo una 'a' sigo llendo hacia la izquierda en la cinta
// hasta encontrar una 'X', allí reempiezo toda la rutina

q3, a
q3, a, <

q3, X
q0, X, >

En general este autómata de cinta busca revisar si hay igual cantidad de 'a' que de 'b', si la palabra no empieza por b

b)



//q0

//siga leyendo 'b' hasta que encuentre una 'a'
//y reemplazela por una 'b'
q0, b
q0, b, >

q0, a
q1, b, >

//después siga leyendo hasta que encuentre un espacio, y
//reemplacelo por 'X' y devuelvase hacia la izquierda
q1, a
q1, a, >

q1, b
q1, b, >

q1, _
q2, X, <

//si lee 'b' o 'X' siga avanzando la cinta hacia la izquierda
//si encuentra un espacio vacío detengase y acepte,
//en caso de ver una 'a' reemplacela por 'b' y regrese a la rutina de q1

q2, b
q2, b, <

q2, X
q2, X, <

q2, a
q1, b, >

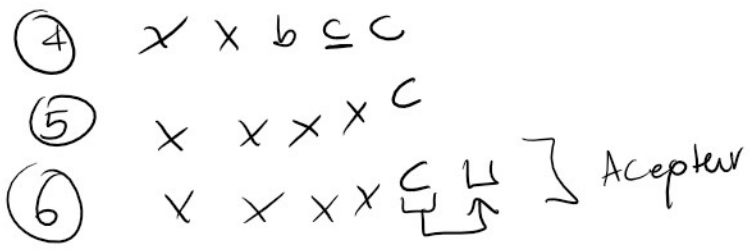
q2, _
qAccept, _, -

Reemplaza una 'a' por una 'b' una única vez

EJERCICIO 5: Implemente en el simulador las TMs que reconozcan los siguientes lenguajes (asuma $\Sigma = \{a, b, c\}$ y $\Gamma = \{a, b, c, X, \sqcup\}$):

a) $L = \{w \in \Sigma^*: |w| = 2n + 1,\ n \in \mathbb{N}\}$.

La idea del algoritmo será moverse un paso hacia adelante, si adelante hay una letra entonces poner una 'X' en el lugar actual, ir un paso atrás y poner otra 'X' a la anterior para después avanzar hasta encontrar la primera letra; repetir el paso anterior, en caso de que no haya letra delante se acepta.




```
//primer paso hacia adelante en caso de que lea algo

qFirstMove, a
qFindSuccesor, a, >

qFirstMove, b
qFindSuccesor, b, >

qFirstMove, c
qFindSuccesor, c, >

//busca el sucesor para la letra que encontró

qFindSuccesor, a
qSuccesorFound, X, <

qFindSuccesor, b
qSuccesorFound, X, <

qFindSuccesor, c
qSuccesorFound, X, <

qFindSuccesor, _
qAccept, _, -
```

```
//en caso de que se encuentre un sucesor

qSuccesorFound, a
qPredecessorDeleted, X, >

qSuccesorFound, b
qPredecessorDeleted, X, >

qSuccesorFound, c
qPredecessorDeleted, X, >


//después de borrar al predecesor, y llegar a donde estaba el sucesor
//de este, debe dar un paso más hacia adelante

qPredecessorDeleted, X
qFindSuccesor2, X, >

//busca denuevo hacia adelante para ver si hay sucesor, si no hay
entonces
//acepta, si hay repite el proceso 'qFindSuccesor'

qFindSuccesor2, a
qFindSuccesor, a, >

qFindSuccesor2, b
qFindSuccesor, b, >

qFindSuccesor2, c
qFindSuccesor, c, >
```

b) $L = \{w \in \Sigma^*: N_a(w) = 3\}$, donde $N_a(w)$ representa el número de as en w.

Este algoritmo va a ser muy simple, será un búsqueda lineal, donde el contador sea el estado de la TM como se explica:

```
//busca la primera 'a'
qFindFirst_a, c
qFindFirst_a, c, >

qFindFirst_a, b
qFindFirst_a, b, >

qFindFirst_a, _
qReject, _, -

qFindFirst_a, a
qFindSecond_a, X, >

//busca la segunda 'a'
qFindSecond_a, b
qFindSecond_a, b, >

qFindSecond_a, c
qFindSecond_a, c, >

qFindSecond_a, _
qReject, _, -

qFindSecond_a, a
qFindThird_a, X, >
```

```
//busca la segunda 'a'

qFindThird_a, b
qFindThird_a, b, >

qFindThird_a, c
qFindThird_a, c, >

qFindThird_a, _
qReject, _, -

qFindThird_a, a
qFind_a_Overflow, X, >

// mira si hay más 'a' de las esperadas

qFind_a_Overflow, b
qFind_a_Overflow, b, >

qFind_a_Overflow, c
qFind_a_Overflow, c, >

qFind_a_Overflow, a
qReject, X, -

qFind_a_Overflow, _
qAccept, _, -
```

c) $L = \{a^n b^n: n \geq 0\}$.

aaaaaaaaaabbbbbbbbbb

```
//primer paso
qFirstStep, a
qFindLast_b, X, >


//busca la 'a'
qFind_a, b
qFind_a, b, >

qFind_a, a
qFindLast_b, X, >
```

```
//busca la 'a' hacia atras

qFind_a_BackWards, b
qFind_a_BackWards, b, <

qFind_a_BackWards, a
qFind_a_BackWards, a, <

qFind_a_BackWards, X
qEndOfWord_a, X, >
```

```
qFind_a, a
qFindLast_b, X, >

//busca la ultima 'b'
qFindLast_b, a
qFindLast_b, a, >

qFindLast_b, b
qFindLast_b, b, >

//si se encuentra una 'X' se devuelve y toma la 'b'
previa

qFindLast_b, X
qEndOfWord_b, X, <

qEndOfWord_b, b
qFind_a_BackWards, X, <

//fin de la cadena de texto por lo que se debe
regresar un paso y tomar la b que quedó atrás
qFindLast_b, _
qEndOfWord, _, <

qEndOfWord, b
qFind_a_BackWards, X, <
```

```
qFind_a_BackWards, X
qEndOfWord_a, X, >

//una vez acabó de leer todas las 'a' de atras hacia adelate
//toma la primera a disponible y repite el proceso

qEndOfWord_a, a
qFindLast_b, X, >

qEndOfWord_a, X
qAccept, X, -
```

d) $L = \{a^n b^n c^n : n \geq 0\}.$

Este ejercicio es prácticamente el mismo anterior pero con la siguiente idea clave, dado que en el anterior ibamos a quedar con 2n 'X' entonces por cada 'c' que se encuentre al final, se pueden borrar 2'X' del principio, el resto para que esto funcione bien es tema de manejo de estados.

```
//Load your code and click COMPILE.
//or load an example (top-right).

//primer paso
qFirstStep, a
qFindLast_b, X, >


//busca la 'a'
qFind_a, b
qFind_a, b, >

qFind_a, a
qFindLast_b, X, >

//busca la ultima 'b'
qFindLast_b, a
qFindLast_b, a, >

qFindLast_b, b
qFindLast_b, b, >

//si se encuentra una 'X' se devuelve
qFindLast_b, X
qEndOfWord_b, X, <

//si se encuentra una 'c' se devuelve
qFindLast_b, c
qEndOfWord_b, c, <

qEndOfWord_b, b
qFind_a_BackWards, X, <

qFindLast_b, _
qEndOfWord, _, <

qEndOfWord, b
qFind_a_BackWards, X, <

//busca la 'a' hacia atras

qFind_a_BackWards, b
qFind_a_BackWards, b, <

qFind_a_BackWards, a
qFind_a_BackWards, a, <

qFind_a_BackWards, X
qEndOfWord_a, X, >
```

```
//una vez acabó de leer todas las 'a' de atras hacia adelate
//toma la primera a disponible y repite el proceso

qEndOfWord_a, a
qFindLast_b, X, >

qEndOfWord_a, X
qGoToBeggining, X, <

//Ahora que se tiene un numero 2n de 'X' la idea va a ser que
//por cada 'c' encontrado se borren 2 'X' del principio,
//si no quedan 'X' atrás entonces todo correcto

qGoToBeggining, X
qGoToBeggining, X, <

qGoToBeggining, c
qGoToBeggining, c, <

qGoToBeggining, _
qDeleteFirst_X, _, >

//borra la primera X
qDeleteFirst_X, X
qDeleteSecond_X, _, >


//borra la segunda X
qDeleteSecond_X, X
qFindLast_C, _, >

//busca la última C para borrarla
qFindLast_C, X
qFindLast_C, X, >

qFindLast_C, c
qFindLast_C, c, >

qFindLast_C, _
qDeleteLast_C, _, <

qDeleteLast_C, c
qGoToBeggining, _, <

//en caso de que no haya nada que borrar acepte
qDeleteFirst_X, _
qAccept, _, -
```