

# Tema-2.pdf



PruebaAlien



Modelos de Computación



3º Grado en Ingeniería Informática

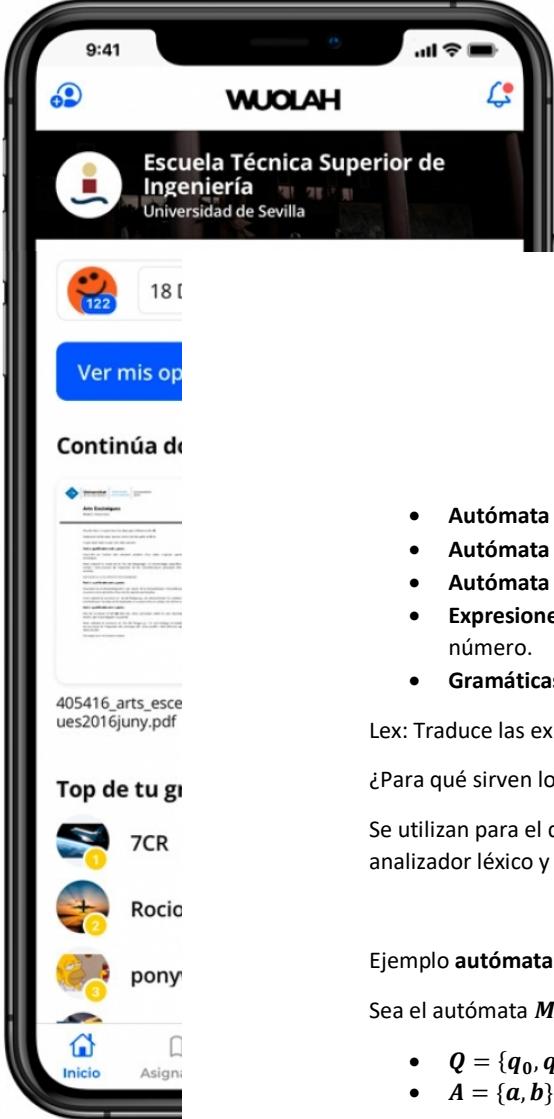


Escuela Técnica Superior de Ingenierías Informática y de  
Telecomunicación  
Universidad de Granada



**Descarga la APP de Wuolah.**  
Ya disponible para el móvil y la tablet.





# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play

- Autómata Finito Determinista
- Autómata Finito No-Determinista
- Autómata Finito con Transiciones Nulas: no son determinísticos y son más simples.
- Expresiones Regulares: Son un nombre que le damos, ejemplo:  $(0^*1^*)$  cualquier número.
- Gramáticas Regulares

Lex: Traduce las expresiones regulares y lo transforma en un autómata.

¿Para qué sirven los autómatas finitos?

Se utilizan para el diseño de circuitos digitales, para construir compiladores (tienen el analizador léxico y analizador sintáctico), para el Big Data, ...

Ejemplo **autómata finito determinista**:

Sea el autómata  $M = (Q, A, q_0, \delta, F)$ , donde

- $Q = \{q_0, q_1, q_2\}$
- $A = \{a, b\}$

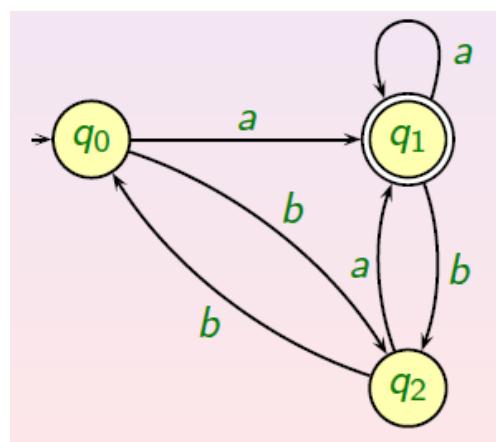
La función de transición viene dada por:

$$\begin{aligned}\delta(q_0, a) &= q_1, & \delta(q_0, b) &= q_2 \\ \delta(q_1, a) &= q_1, & \delta(q_1, b) &= q_2 \\ \delta(q_2, a) &= q_1, & \delta(q_2, b) &= q_0 \\ F &= \{q_1\}\end{aligned}$$

$q_0$  es el estado inicial (se indica con una flecha)

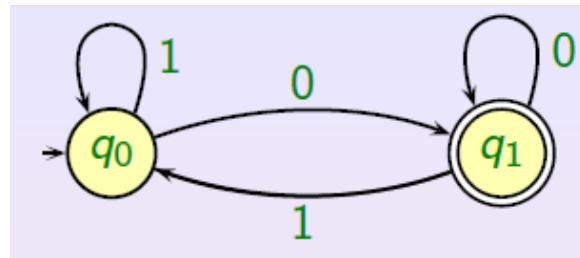
Es un grafo en el que:

- Hay un nodo por cada estado
- Por cada transición  $\delta(q, a) = p$  hay un arco de  $q$  a  $p$  con la etiqueta  $a$ .
- Los estados finales están indicados con una doble circunferencia.



### Calculo asociado. Traza

Este autómata genera la expresión:  $L = \{u0 \mid u \in \{0, 1\}^*\}$

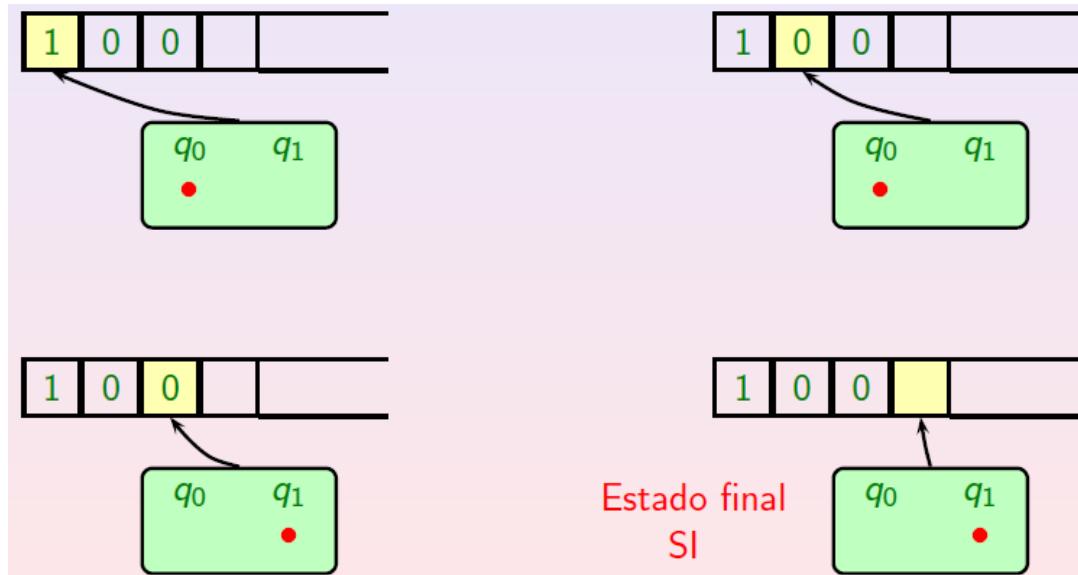


Ejemplo: 0010

$q_0 \equiv$  al ultimo simbolo leido es un 1

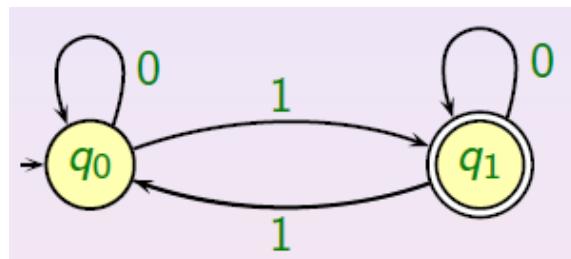
$q_1 \equiv$  al ultimo simbolo leido es un 0

Entonces  $q_0$  lo ponemos como inicial y  $q_1$  lo ponemos como final.



Otro ejemplo:

Este es un automata finito determinista



Este acepta las cadenas:

- 010
- 1



**KEEP  
CALM  
AND  
ESTUDIA  
UN POQUITO**

- 10
- 1110

Esto es porque el estado acaba en  $q_1$

Cadenas que rechaza:

- 011
- 11
- 110
- 0110
- 1111

Con esto llegamos a la conclusión de que para se acepte tiene que ser un numero de 1's impar.

$q_0 \equiv$  ha leido un numero par de 1's

$q_1 \equiv$  ha leido un numero impar de 1's

En las gramáticas regulares tenemos 2 tipos:

- Gramática lineal por la derecha (la que usamos)

Ejemplo: 01010, con la gramática:  $S \rightarrow 0A$ ,  $A \rightarrow 10A$ ,  $A \rightarrow \epsilon$

$$S \rightarrow 0A \rightarrow 010A \rightarrow 01010A \rightarrow 01010$$

La variable siempre queda a la derecha (S,A,B,...) en la gramática lineal por la derecha.

- Gramática lineal por la izquierda

Ejemplo: 01010, con la gramática:  $S \rightarrow S10$ ,  $S \rightarrow 0$

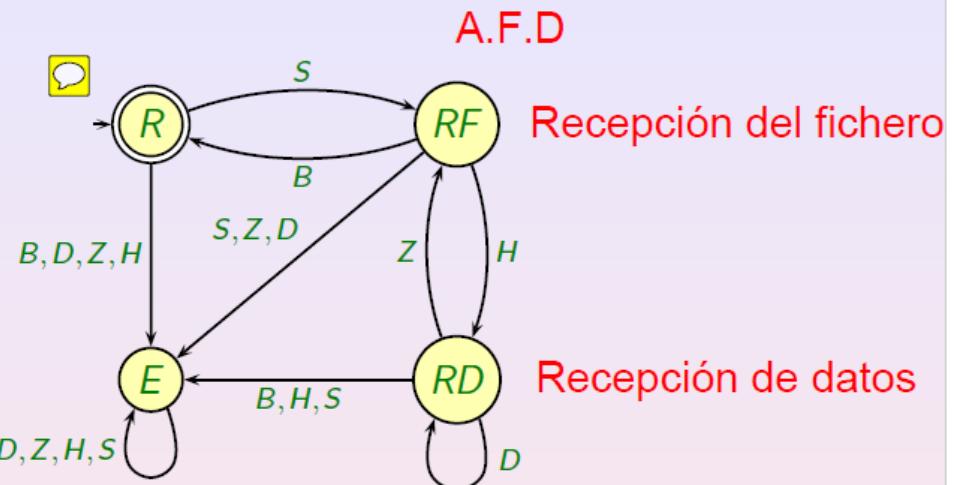
$$S \rightarrow S10 \rightarrow S1010 \rightarrow 01010$$

Expresiones regulares:

- $0(10)^*$  → Esto nos quiere decir que siempre empieza en 0 y hay 0, 1 o varas parejas de 10. Ejemplo (0, 010, 01010, ...) [el \* significa 0 o varios]
- $0(10)^+$  → lo mismo que el anterior, salvo que el + indica que al menos haya 1 pareja de 10. Ejemplo: 010, 01010, ...

(OJO NO CONFUNDIR CON  $1^i 0^i$  (111000) SON DISTINTOS)

# Comunicaciones Correctas



**R:** Estado de espera

**RD:** Recepción de datos

**S:** Comienza recepción

**H:** Cabecera de fichero

**D:** Datos

**RF:** Estado de recepción de ficheros

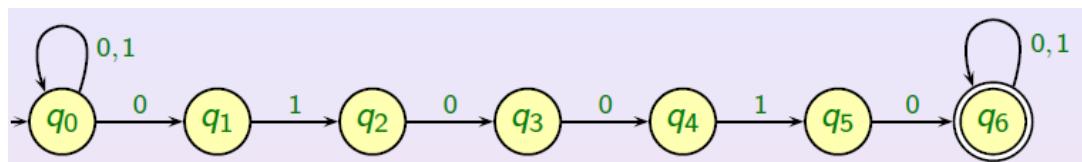
**E:** Error

**B:** Fin de recepción

**Z:** Fin de fichero

**R →** Es el estado inicial y final

Este es un autómata finito determinista, porque está perfectamente especificado, es decir que todos los posibles errores acaban en el **nodo E**.



Este es un **autómata finito de transición nula**, pero no es **determinístico**, de tal forma que podemos convertirlo en el mejor **autómata finito determinista** del mundo.

Esto nos permite crear 6 estados que aceptan la cadena **010010** llegando al estado q6.

Este autómata resuelve el problema de que haya cadenas delante o atrás, ejemplo 011001001010010 en q0 y q6.

Pero este autómata es ineficiente, puesto que explora muchos caminos para encontrar la subcadena.

Ejemplo:

**11001001010**

Lee el primer número **1** y genera un camino, pasa al segundo **1** y continua el camino, pero cuando lee el primer **0** se crean **2 caminos** (uno a **q0** y otro a **q1**) y avanza hasta encontrar otro



# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play



122

I

Ver mis op

Continúa d

405416\_arts\_esce  
ues2016juniy.pdf

Top de tu gu



7CR



Rocio



pony



Inicio



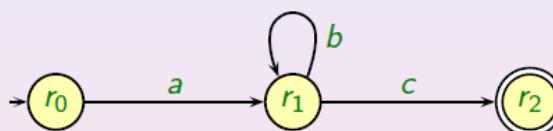
Asigni

0 y se generan otros 2 caminos y así sucesivamente, de tal forma que todos los caminos no serán validos menos 1 que será la correcta.

Para que no se creen tantos caminos, lo que hacemos es convertirlo en un **autómata finito determinístico minimal**, para ello se hace en JFLAP:

1. Convert → convert to FDA (con esto generamos el autómata finito determinístico)
2. Convert → minimize FDA (generamos el autómata finito determinístico minimal)
3. Seleccionamos el nodo raíz y pulsamos **complete subtree** y luego **finish** (con esto exportamos el autómata finito determinístico minimal y podremos hacer pruebas)

También es **no-determinista**:



Acepta cadenas formadas por una **a**, una sucesión de **b** y una **c**.

Este autómata acepta las cadenas **(a[0 o mas]b)c**

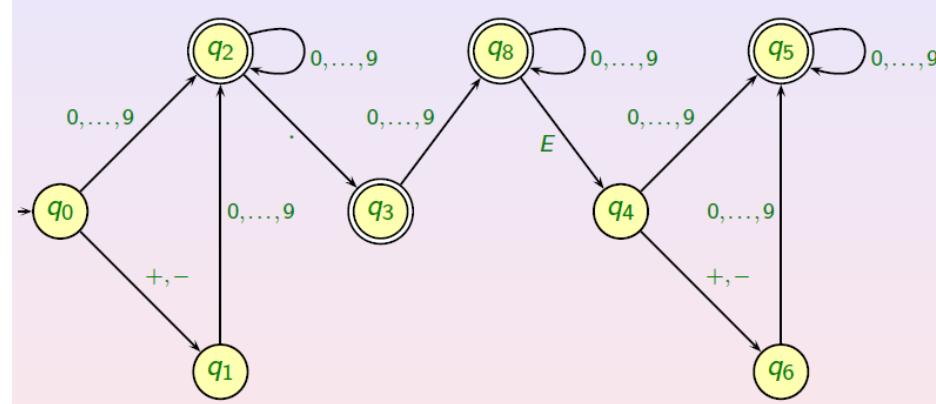
Ejemplos:

- Bbbc → no valida
- Abbbc → valida

Y para convertirlo en un **autómata finito determinístico minimal** los mismos pasos que antes e mencionado.

## Ejemplo: Constantes reales

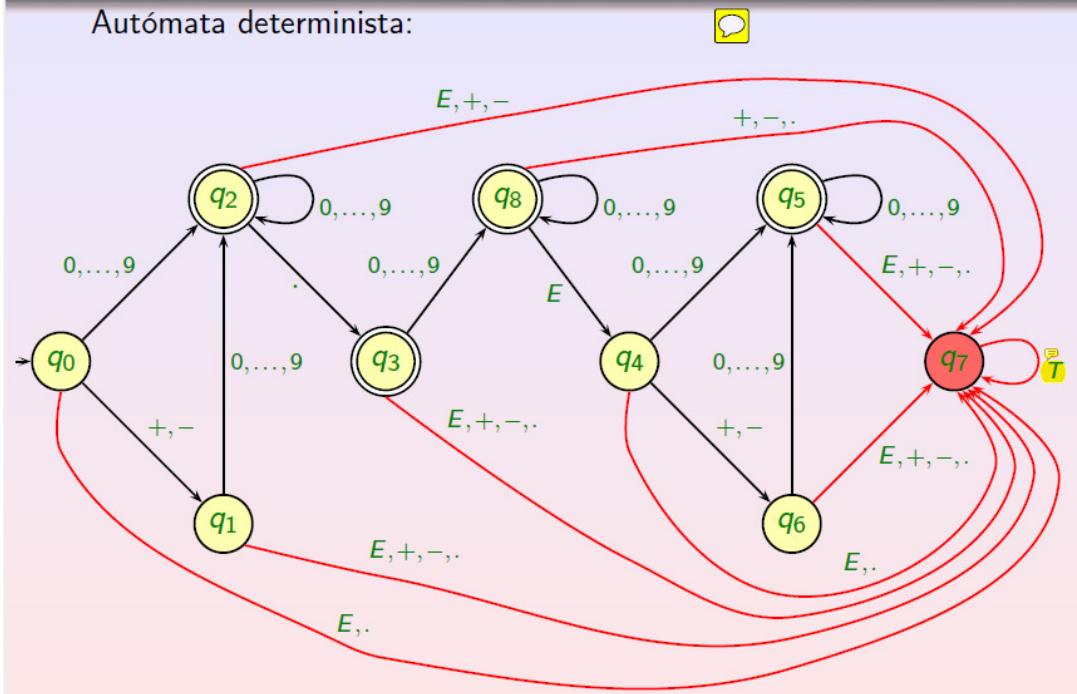
Autómata no determinista:



Para que sea determinista tiene que ir a un estado Error todos los posibles errores.

## Ejemplo: Constantes reales

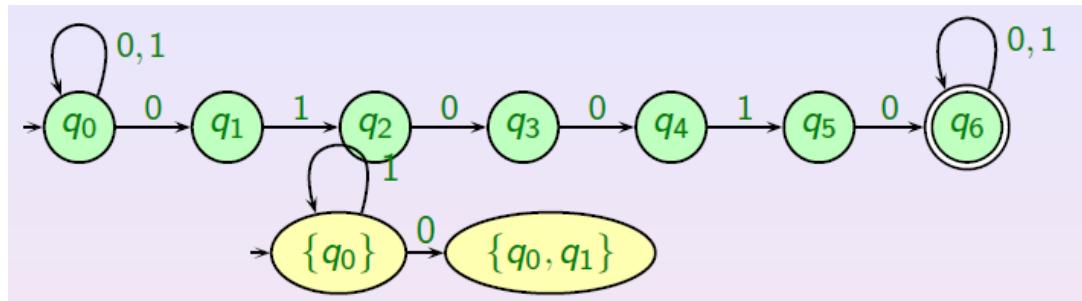
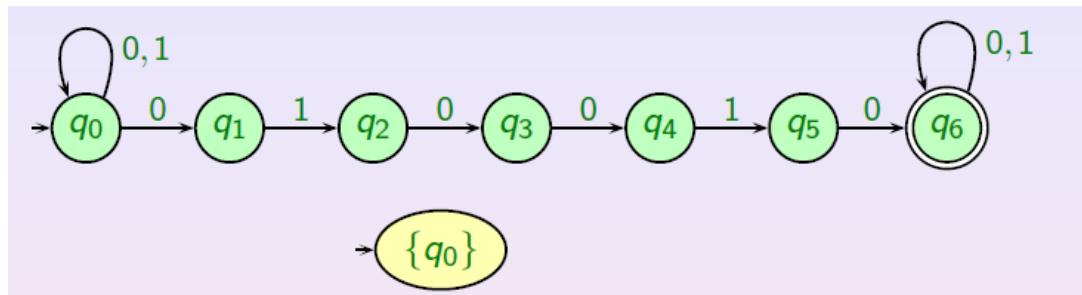
Autómata determinista:



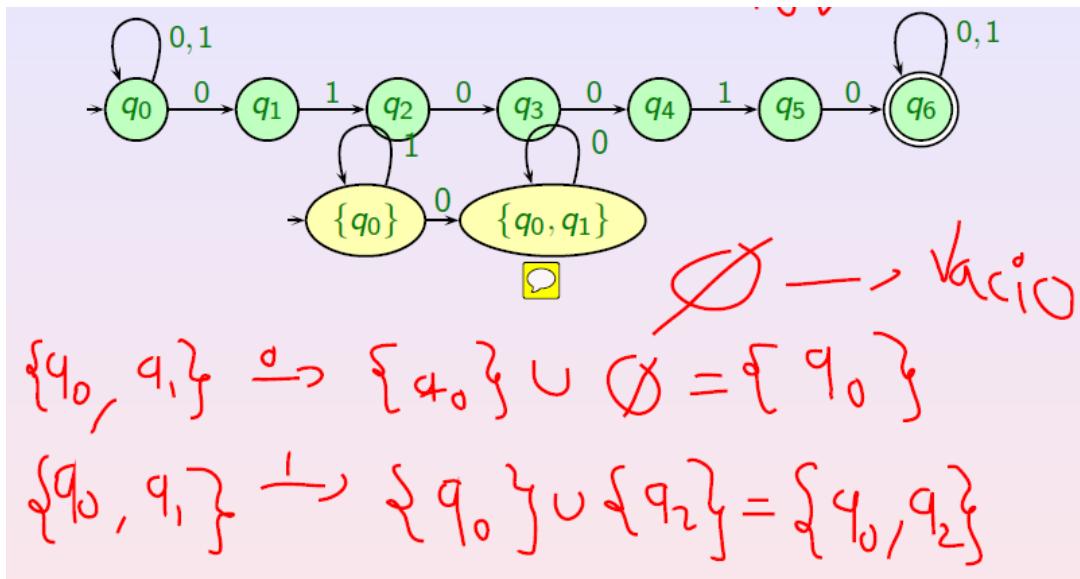
En rojo son las transiciones de error (los posibles errores)

(Nota: T es cualquier símbolo)

Ejemplo de convertir un autómata no determinista en uno determinista:

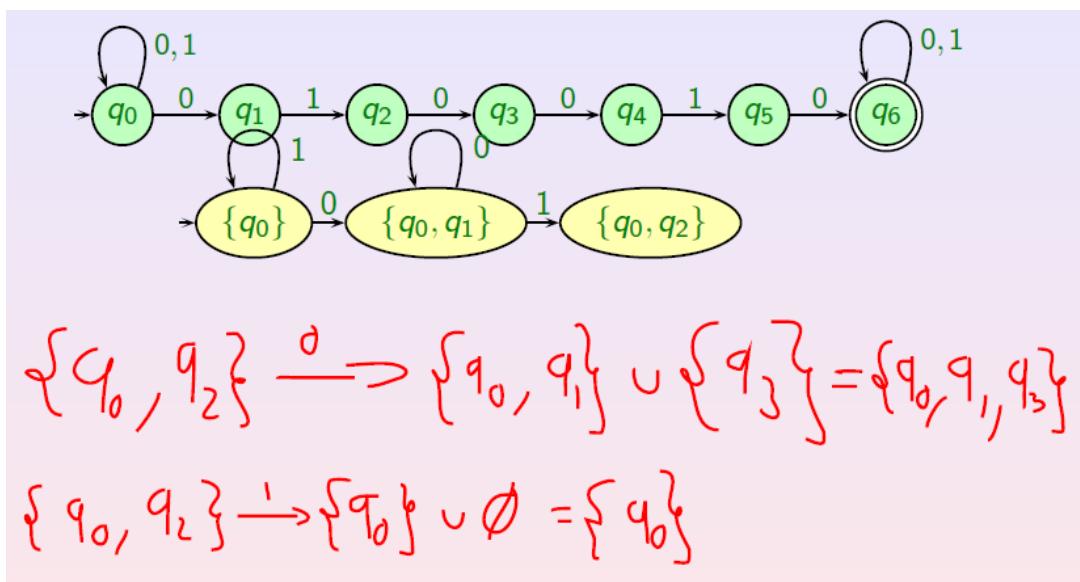


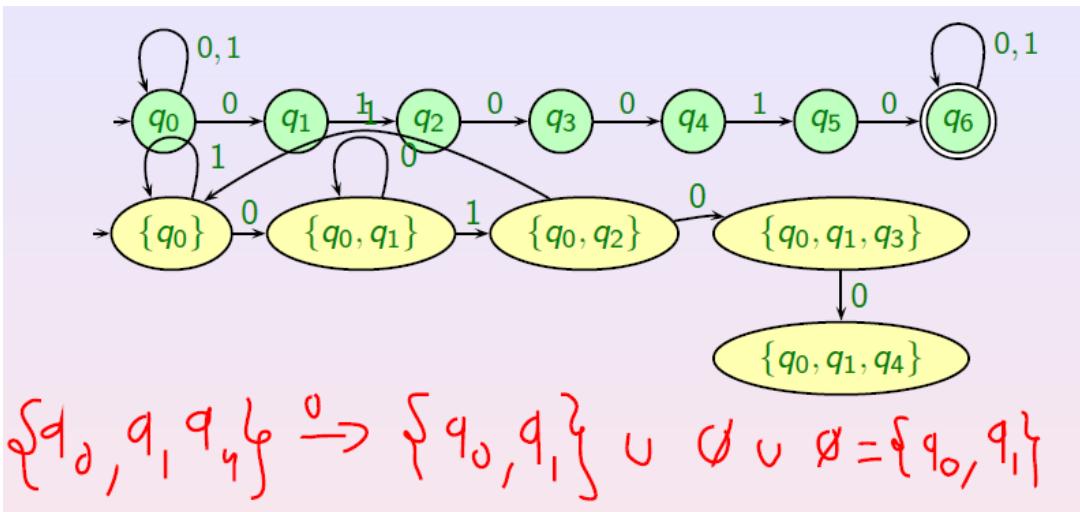
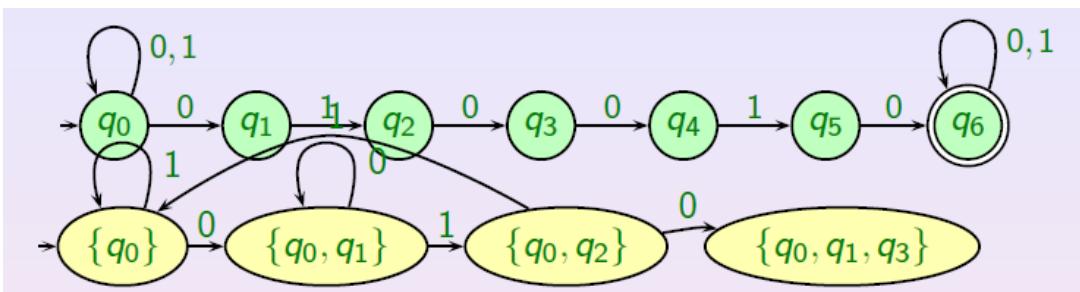
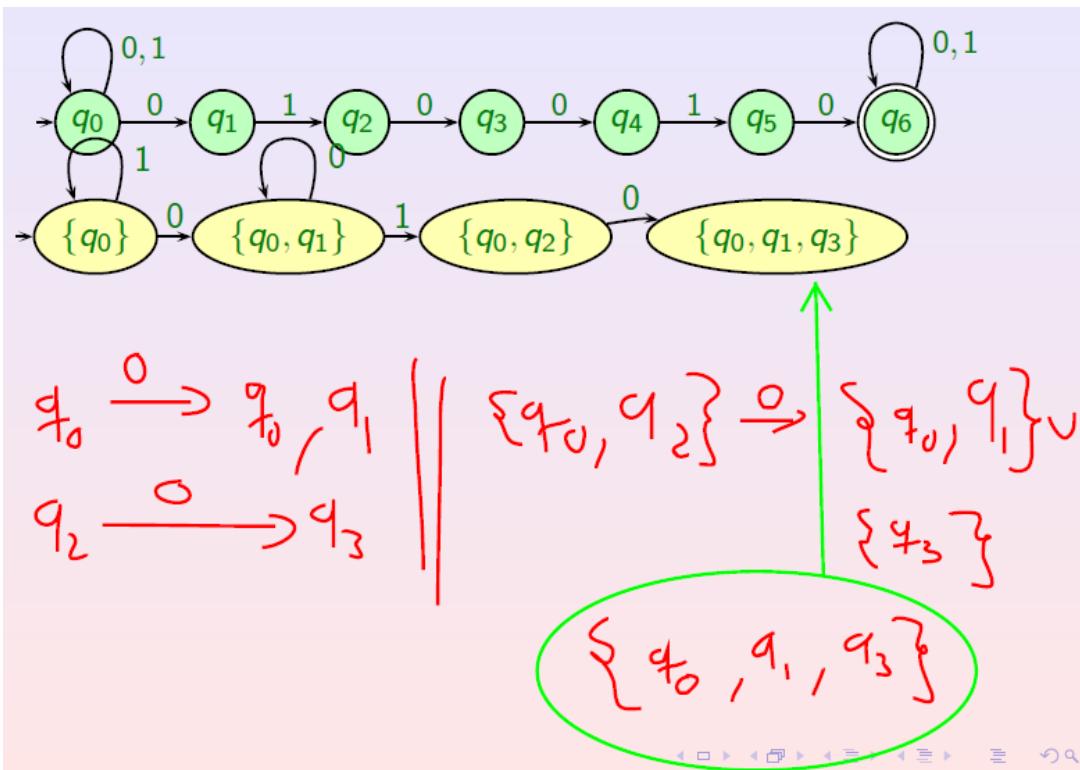
¿Cómo se hace estos pasos?



**Explicación 0:** Cuando lee un **0** en el estado  $\{q_0, q_1\}$  pasa a estado  $q_0$ , esto se debe a que se generan 2 caminos uno en el que está en el estado  $q_0$  y otro en el estado  $q_1$ . Como  $q_1$  no puede avanzar el único que puede avanzar es  $q_0$  que genera otros 2 caminos a  $q_0$  y  $q_1$ , por eso hay una flecha 0 al estado  $\{q_0, q_1\}$ .

**Explicación 1:** Cuando lee un **1** en el estado  $\{q_0, q_1\}$  se genera 2 caminos uno a  $q_0$  y otro a  $q_2$  creando el estado nuevo  $\{q_0, q_2\}$ .







# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play



122

I

Ver mis op

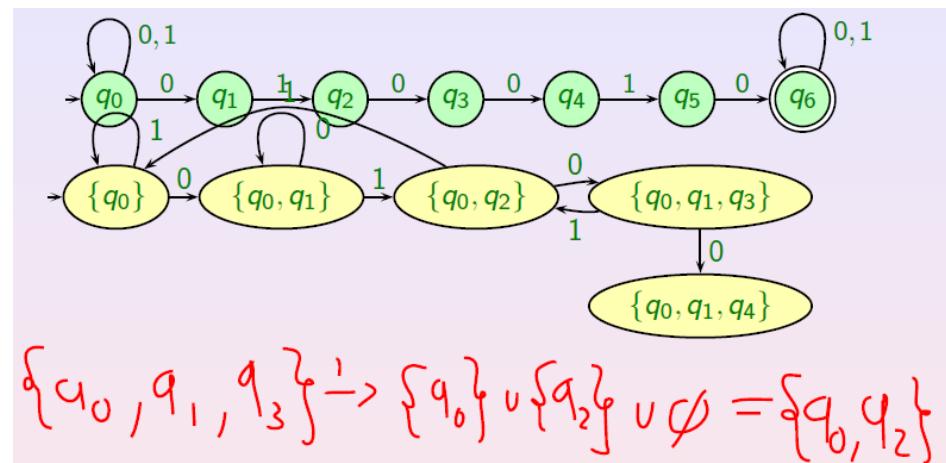
Continúa d

405416\_arts\_esce  
ues2016juniy.pdf

Top de tu gu

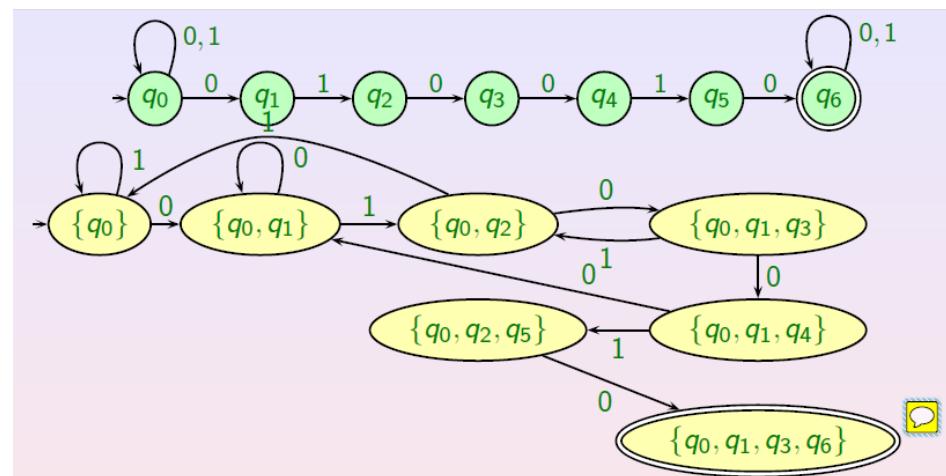
- 7CR
- Rocio
- pony

Inicio    Asigni

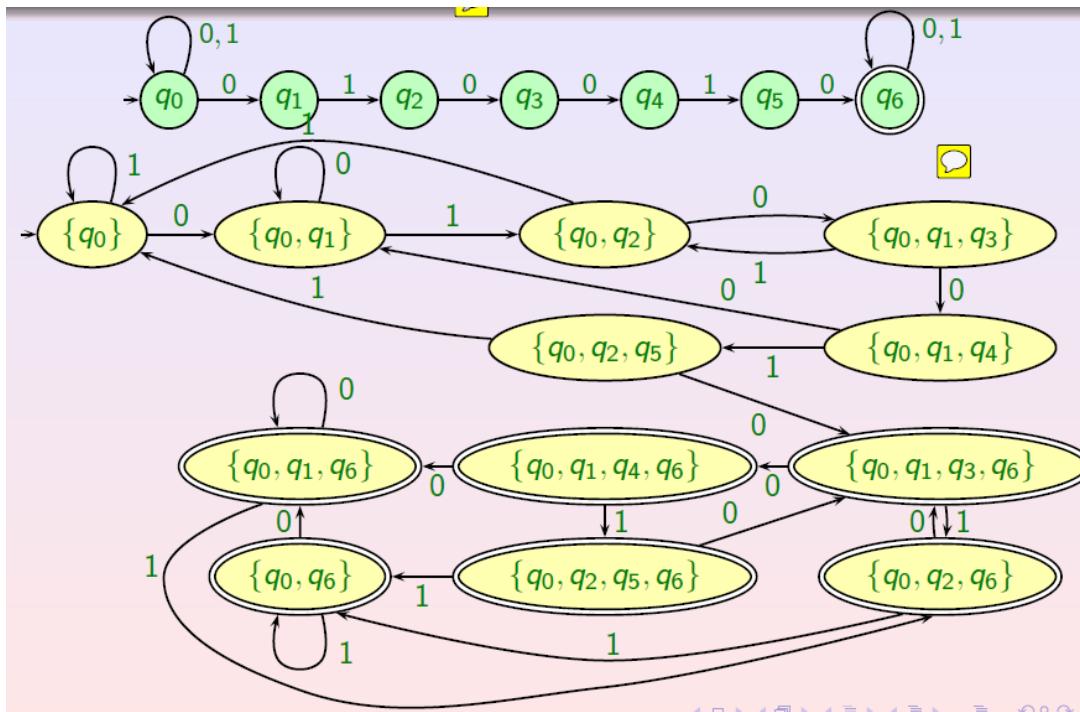


Y asi hasta completar el automata.

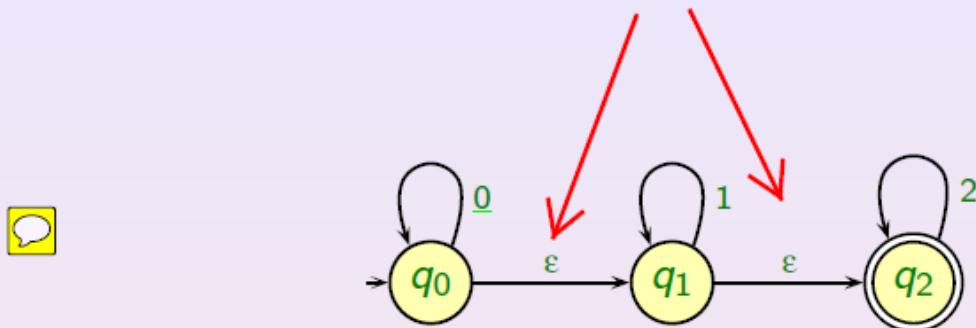
NOTA: ¿Cuándo llega a un estado final? Cuando aquel que tiene un automata finito es final que contenga  $q_6$  (010010)



Llegando a tener al final este autómata finito determinista, pero no minimal:

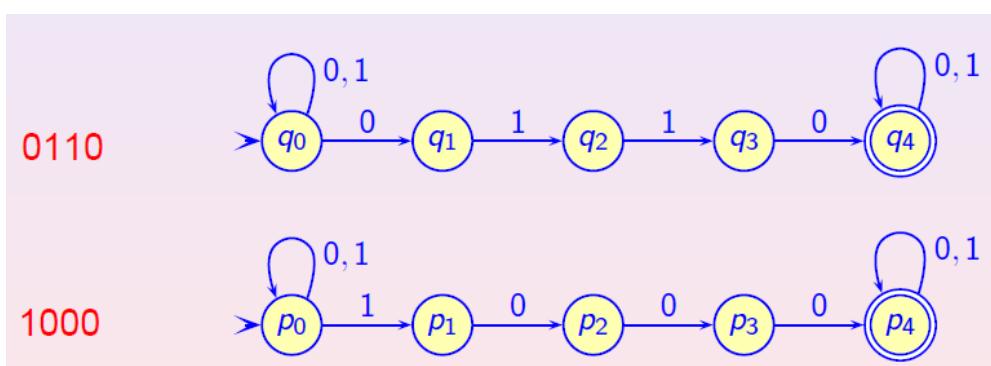


Permite transiciones nulas en JFLAP



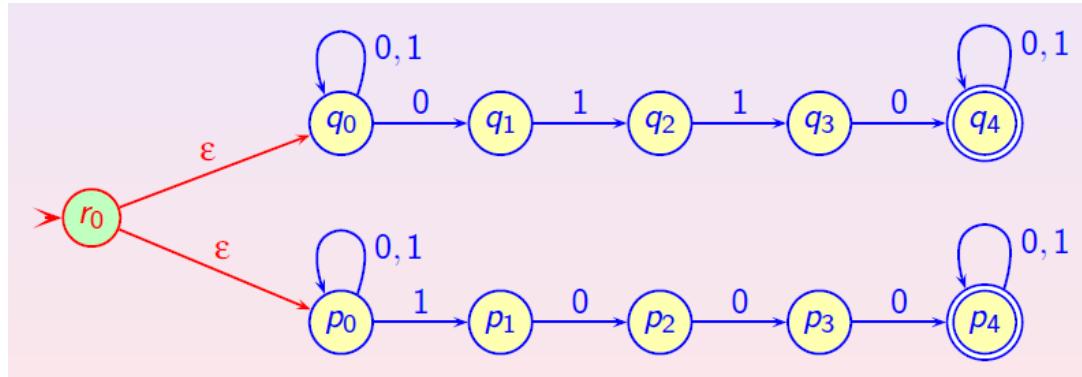
El lenguaje aceptado es  $L = \{0^i 1^j 2^k : i, j, k \geq 0\}$ .

Ejemplo en el que tiene que tener la subcadena **0110** o **1000**:



¿Cómo saber si no se considera determinístico? **Cuando tenga 2 o más caminos.**

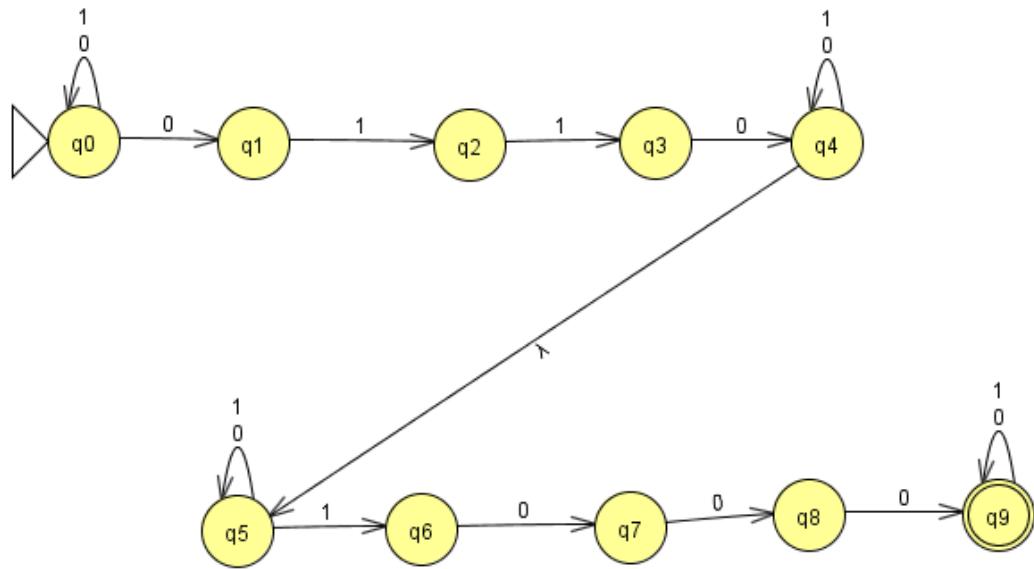
En este ejemplo nunca va a ser valida, pero esto se arregla con un Automata Finito No Determinista con Transiciones Nulas:



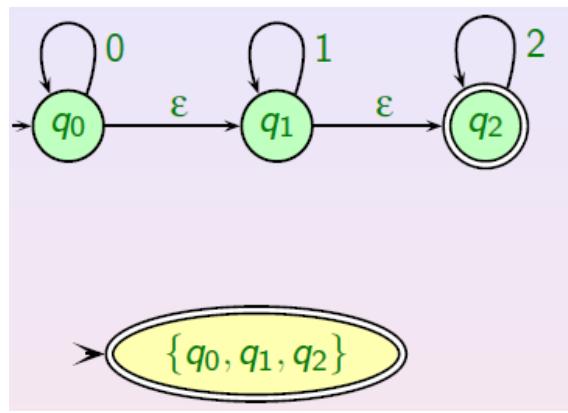
¿Y si quiero que contenga primero la subcadena **0110** y después la subcadena **1000**?

Ejemplo: 01010**01101011000111**

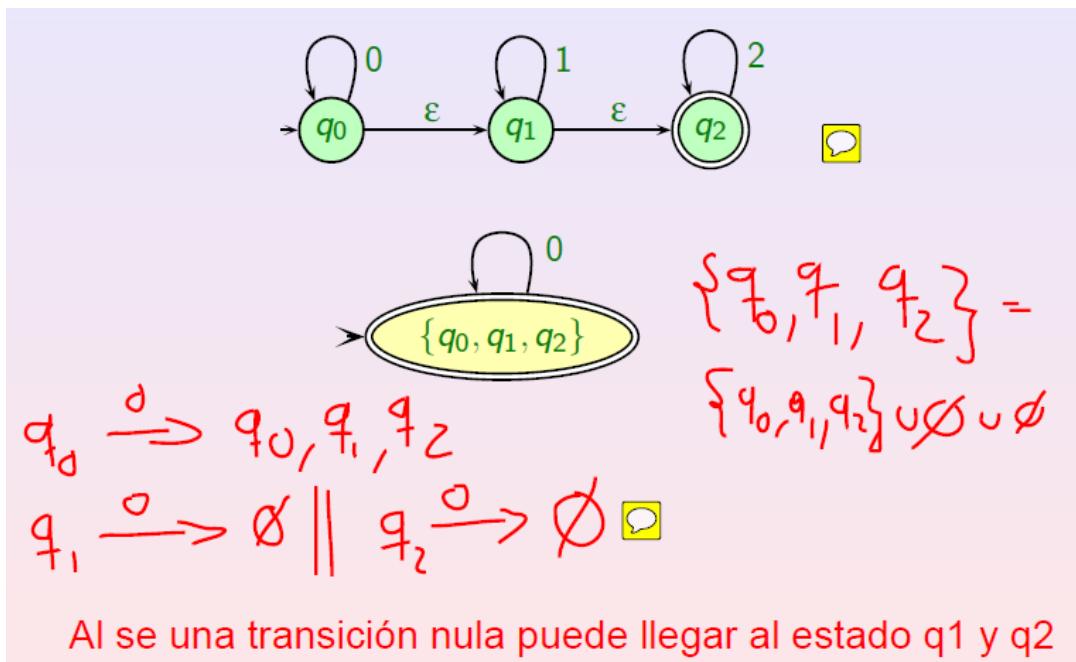
El autómata finito no determinista con transiciones nulas sería:



Este **Autómata Finito No Determinista con Transiciones Nulas** para transformarlo en un AFD (Autómata Finito Determinístico) en este caso el estado inicial es  $(q_0, q_1, q_2)$ , esto se debe a que al tener transiciones nulas podemos llegar al estado  $q_2$  sin haber leído nada de la cadena:



Cuando lee un 0 ocurre lo mismo pasa a  $q_0$ , luego a  $q_1$  por la transición nula y por último por  $q_2$  por la transición nula.



$q_1$  y  $q_2$  son vacíos porque no saben que hacer con el **valor 0**.

Despues en el caso de leer el **valor 1**, pasara al estado  $(q_1, q_2)$ , esto se debe a que  $q_0$  no sabe como interpretar el **valor 1** así que pasa por la transición nula al  $q_1$  lee el **valor 1** y pasa por la transición nula a  $q_2$ .

# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play



18

Ver mis op

Continúa d



405416\_arts\_esce\_ues2016juniy.pdf

Top de tu g



7CR



Rocio



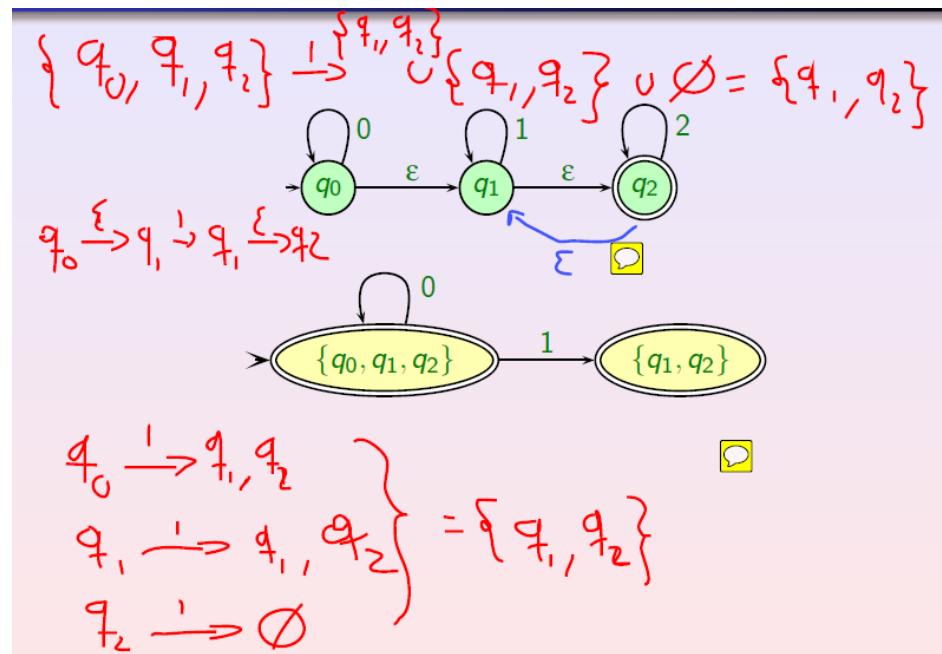
pony



Inicio

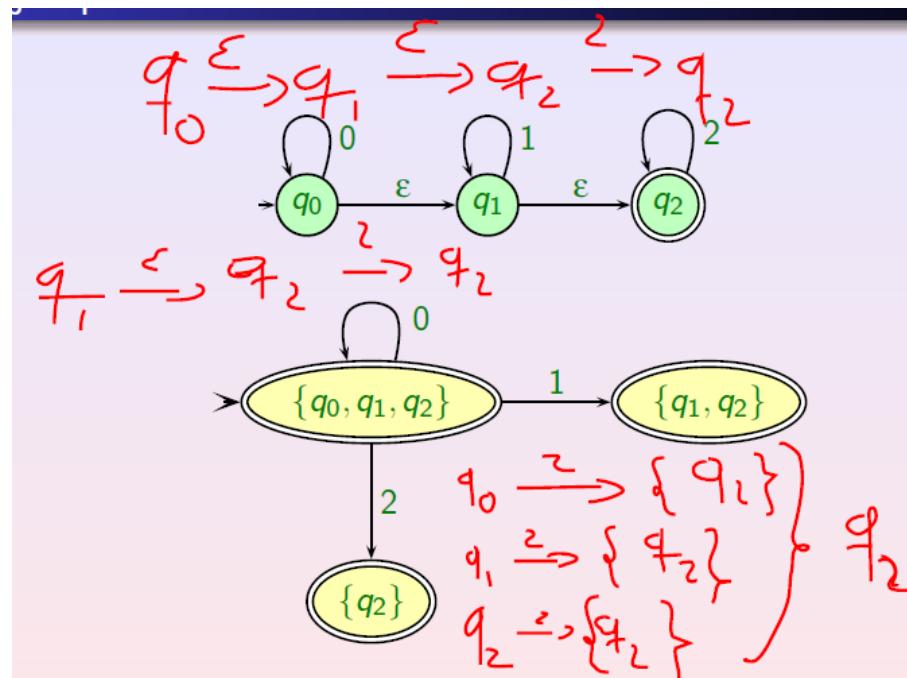


Asigni

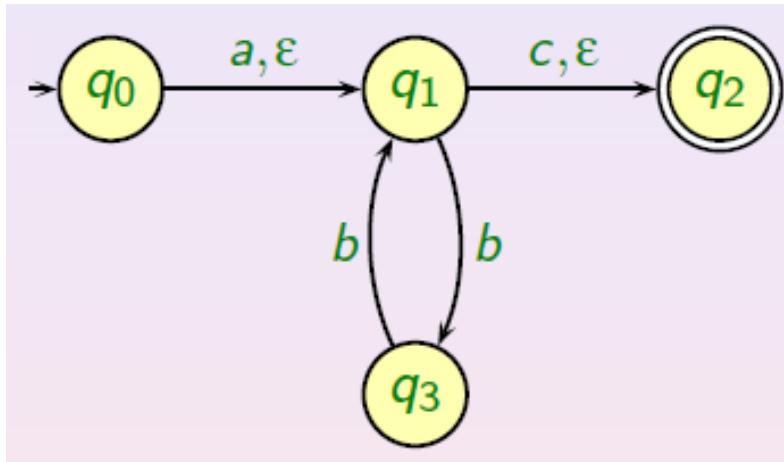
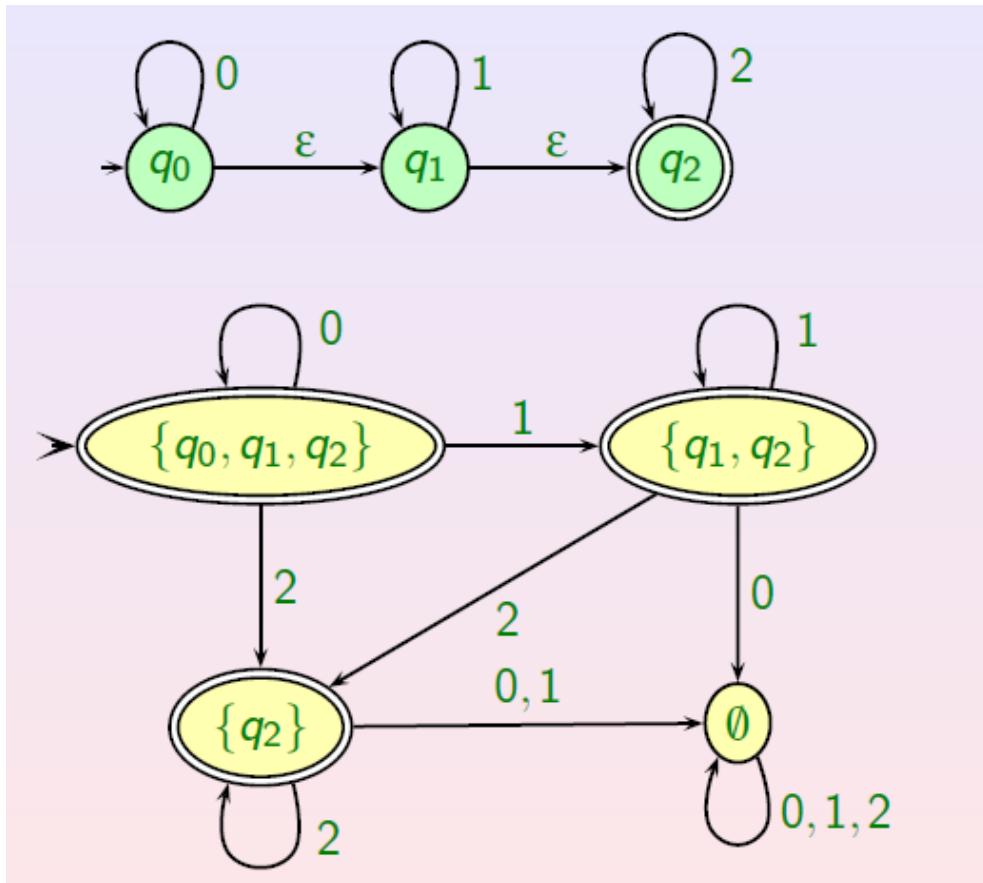


Para que  $q_2$  no sea un conjunto vacío tendría que haber una transición nula de  $q_2$  a  $q_1$ , pero como no la hay pasa a ser un conjunto vacío.

Y en el caso en el que lee el valor 2,  $q_0$  como no sabe qué hacer con el valor 2, pasa por la transición nula a  $q_1$ , con  $q_1$  ocurre lo mismo y pasa por la transición nula a  $q_2$ , como  $q_2$  si sabe qué hacer con el valor 2 lee y termina en el estado  $q_2$ .



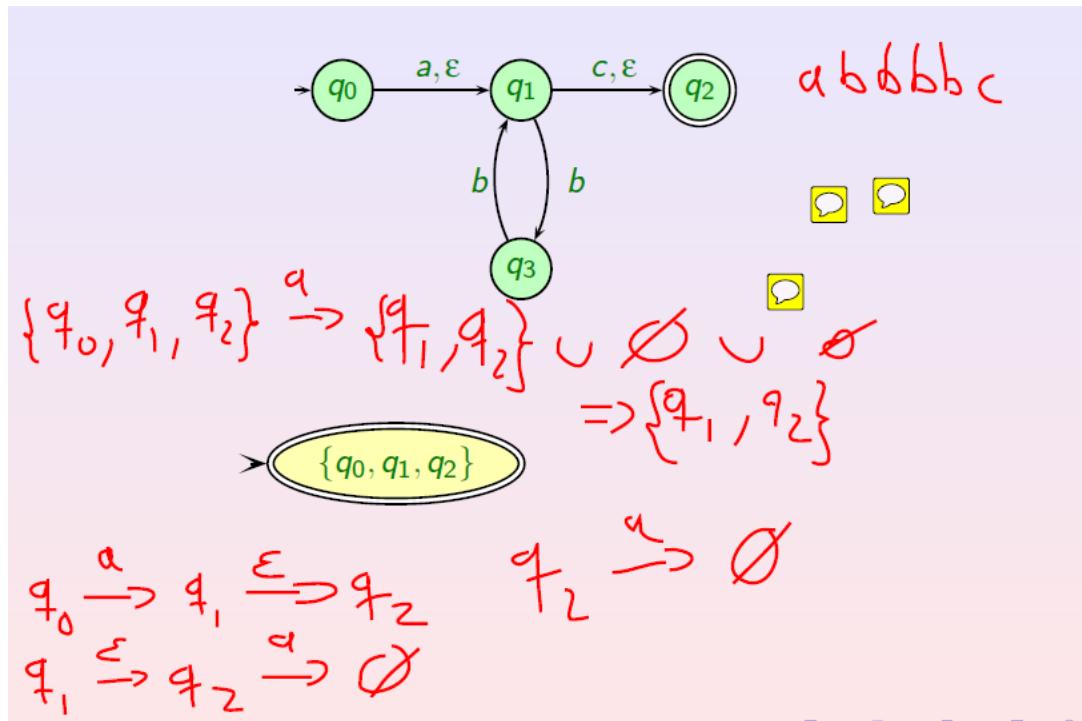
Y ya lo que nos queda son los valores erróneos:



Este **Autómata Finito No Determinista con Transiciones Nulas** genera el lenguaje:

$$L = \{a^i b^{2j} c^k : i, k = 0, 1 \text{ y } j \geq 0\}$$

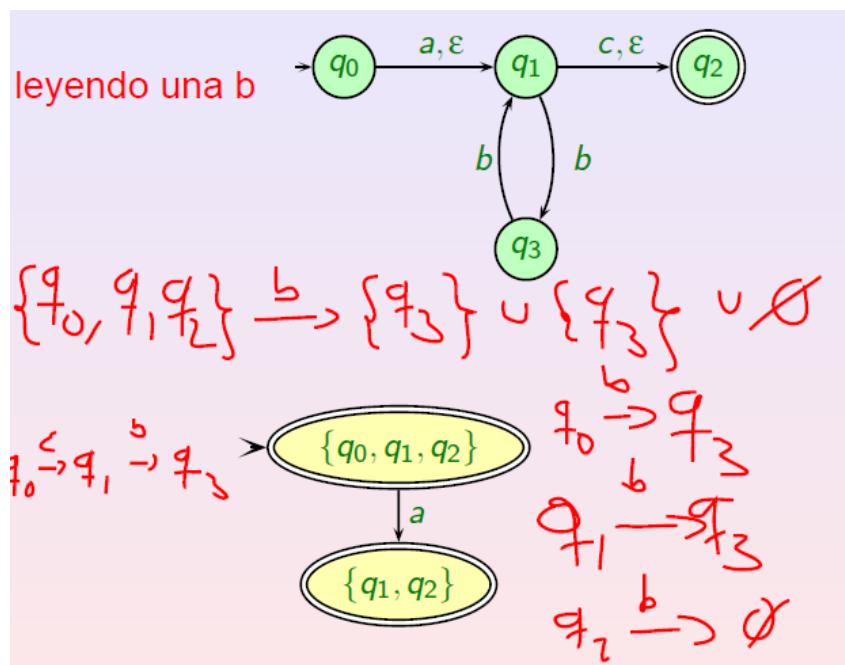
Para empezar el estado inicial de este automata finito no determinista con transiciones nulas es ( $q_0, q_1, q_2$ ), esto se debe a que antes de empezar a leer un valor, pasa por los estados  $q_0, q_1$  y  $q_2$ , por las transiciones nulas.



Cuando recibe el **valor a** al estado **q0** lee el valor y pasa de **q0** a **q1** por el valor a, después pasa de **q1** a **q2**, por la transición nula, ya que no hay más valores y finaliza.

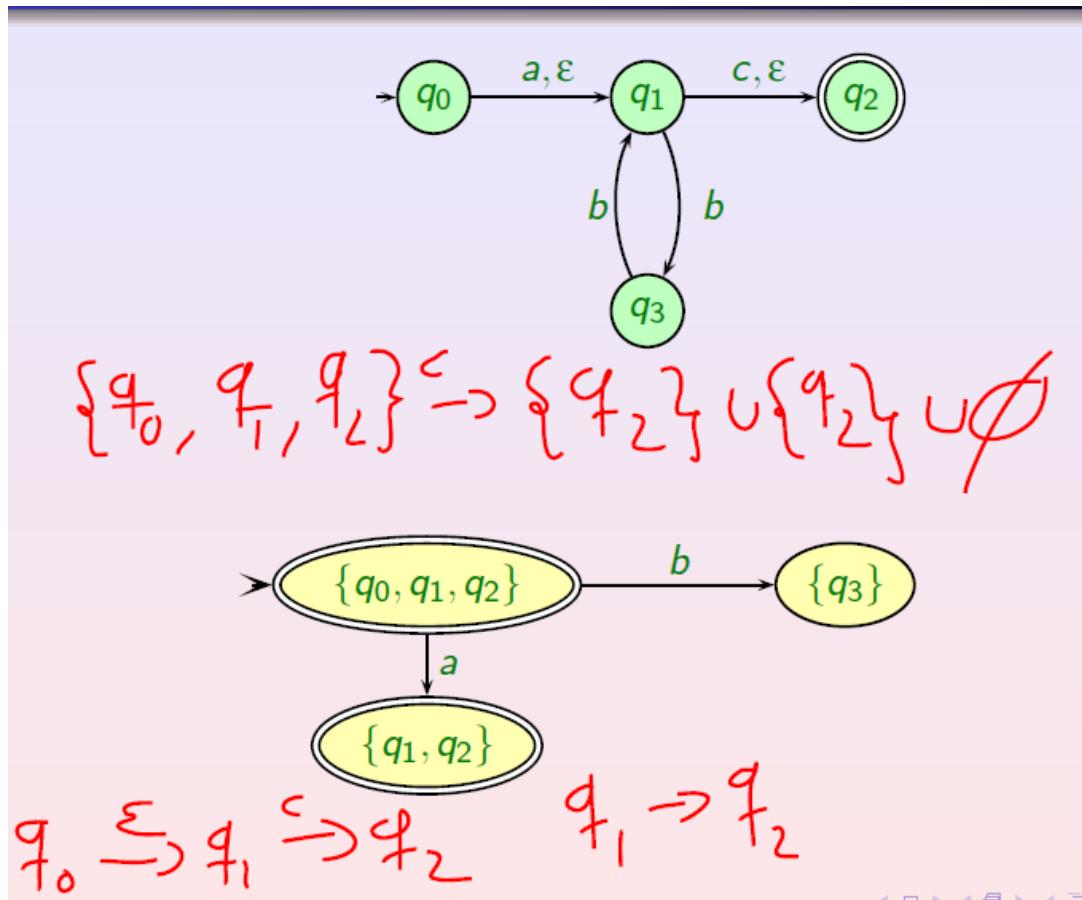
En los casos en el que el valor a esta en **q1** o en **q2** serían conjuntos vacíos, porque no saben que hacer con ese valor.

Con lo cual se crea el estado **(q1, q2)**.



Cuando en **q0** lee el **valor b** no sabe que hacer con ese valor, asi que pasa por la transición nula a **q1** y en **q1** lee el **valor b** y pasa al estado **q3**.

En **q1** lee el **valor b** y pasa al estado **q3**. Y en el **q2** cuando lee el **valor b**, como no sabe que hacer con ese valor es un conjunto vacío.



En el **q0** cuando lee el **valor c**, como no sabe que hacer con ese valor pasa por la transición nula a **q1**, en **q1** lee el **valor c** y pasa al estado **q2**.

En **q1** cuando lee el **valor c**, pasa a **q2** y finaliza. Y si esta en el estado **q2** y lee el valor **c**, como no sabe que hacer con ese valor es un conjunto vacío.



[Ver mis op](#)

Continúa d



405416\_arts\_esce  
ues2016juniy.pdf

Top de tu g

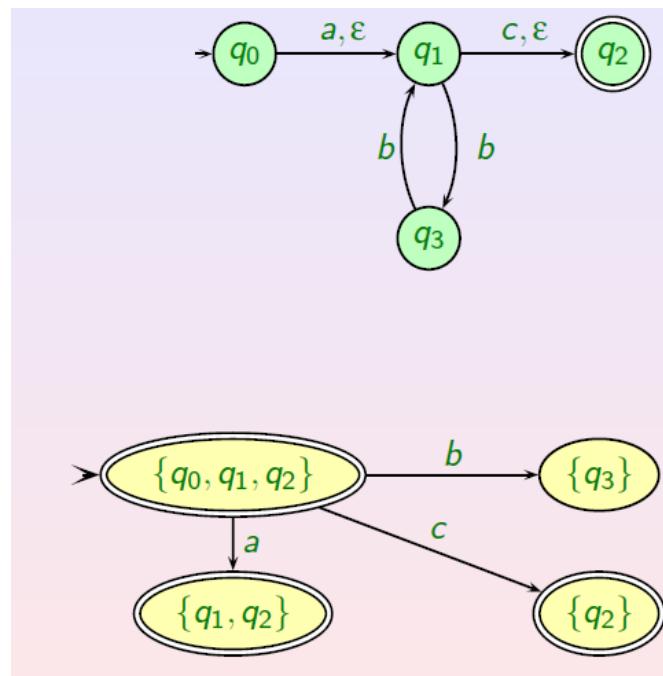
- 7CR
- Rocío
- pony
- Inicio
- Asigni

# Descarga la APP de Wuolah.

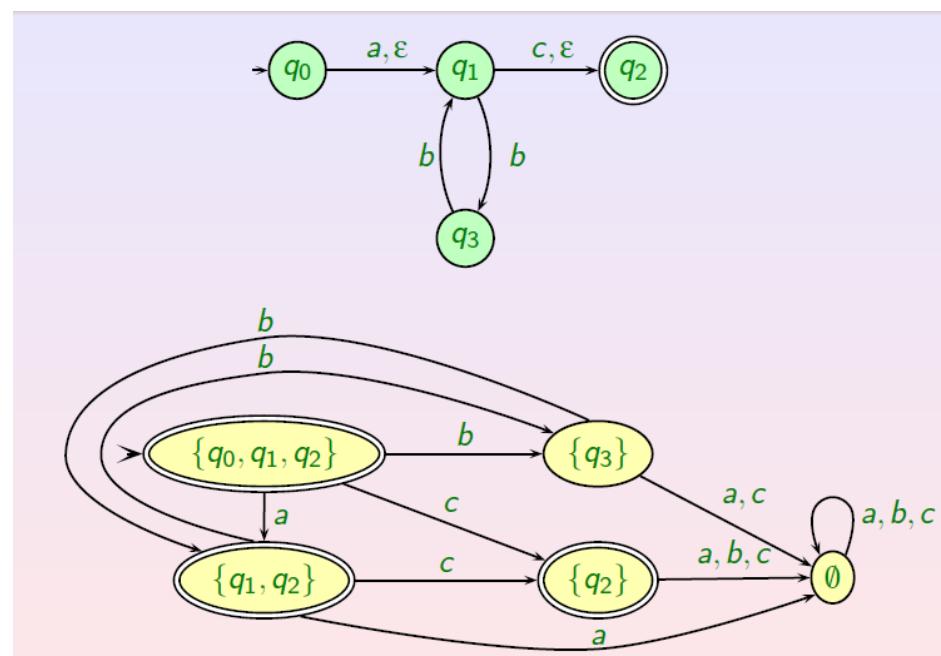
Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play



Y aquí está el autómata finito determinístico completo:



## Expresiones Regulares:

$\emptyset$  es una expresión regular que denota el lenguaje vacío.(expresión vacía)

$\epsilon$  es una expresión regular que denota el conjunto vacío.

Si  $r$  y  $s$  son expresiones regulares denotando los lenguajes  $R$  y  $S$  entonces definimos las siguientes operaciones:

- **Unión:**  $(r + s)$  es una expresión regular que denota el lenguaje  $(R \cup S)$ , es decir la unión de esos 2 lenguajes. Es decir que se cumpla la expresión  $r$  o la expresión  $s$ .
- **Concatenación:**  $(rs)$  es una expresión regular que denota el lenguaje  $RS$ . Es decir que se cumpla la expresión  $r$  y a continuación la expresión  $s$ .
- **Clausura:**  $r^*$  es una expresión regular que denota el lenguaje  $R^*$ . Es decir contiene 0 o muchas ocurrencias que cumpla la expresión regular  $r$ .

Ejemplos:

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

- **00** El conjunto  $\{00\}$
- **$01^* + 0$**  Conjunto de palabras que empiezan por 0 y después tienen una sucesión de unos.
- **$(1+10)^*$**  1010, 10, 110, 101110, 111

- **$(1+10)^*$**  Conjunto de palabras en las que los ceros están precedidos siempre por unos

- **$(0+1)^*011$**  Conjunto de palabras que terminan en 011  
011, 00011, 101011 

De tal forma que puede ser cualquier cadena de 1s y 0s , pero que terminen en 011.

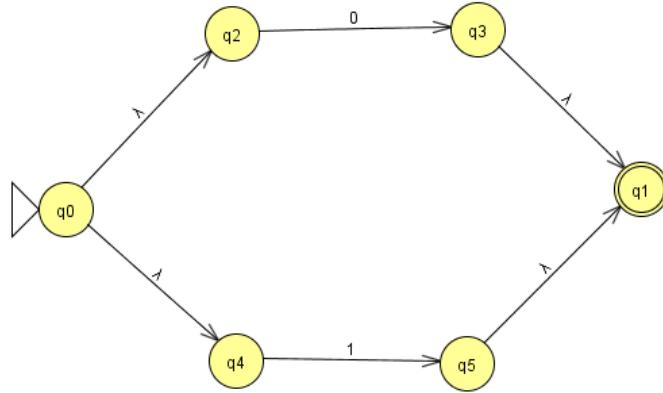
$0^*1^* \rightarrow$  cualquier cadena de 0s y 1s:  $0^* \rightarrow \{0^i \mid i \geq 0\}; 1^* \rightarrow \{1^j \mid j \geq 0\}$

- **$00^*11^*$**  Conjunto de palabras formadas por una sucesión de ceros seguida de una sucesión de unos. Ninguna de las sucesiones puede ser vacía

A  $r^*r$  se le denota como  $r^+$ . La última expresión regular quedaría  $0^+1^+$

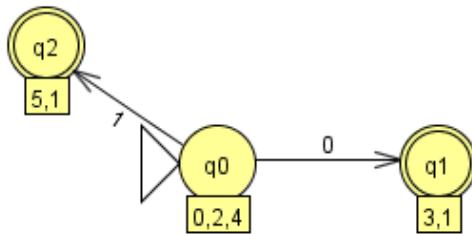
$00^*11^* == 0^+1^+$ ; en este caso jflap no sabe usar  $0^+1^+$ , pero si  $00^*11^*$ , ya que (+) interpreta la unión. Pero en lex la unión lo interpreta con la tubería ( $|$ ) ( $0|1$ ).

Ejemplo:  $(0+1)^*$  → busca conjuntos que tenga 0 o 1 únicamente, generando este autómata finito no determinista:

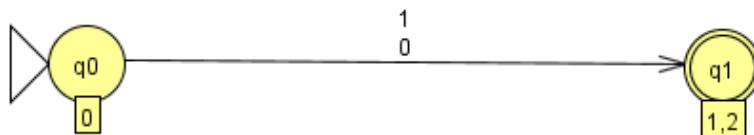


Es decir, solo acepta un 1 o un 0, 01 no lo aceptaría.

Autómata finito determinista:



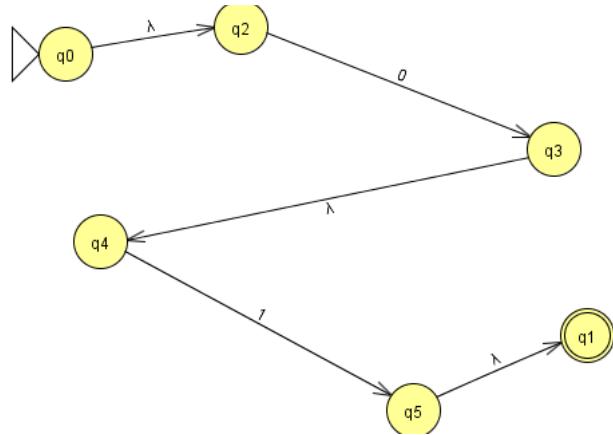
Autómata finito determinista minimal:



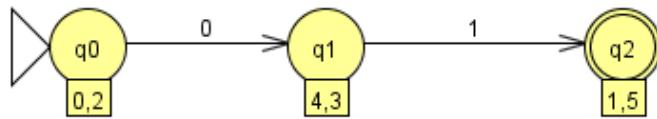
$(0^*+1^*)$  → acepta cadenas que tengan 0 o muchos 1s y 0s

La expresión regular 01:

Autómata finito con transiciones nulas

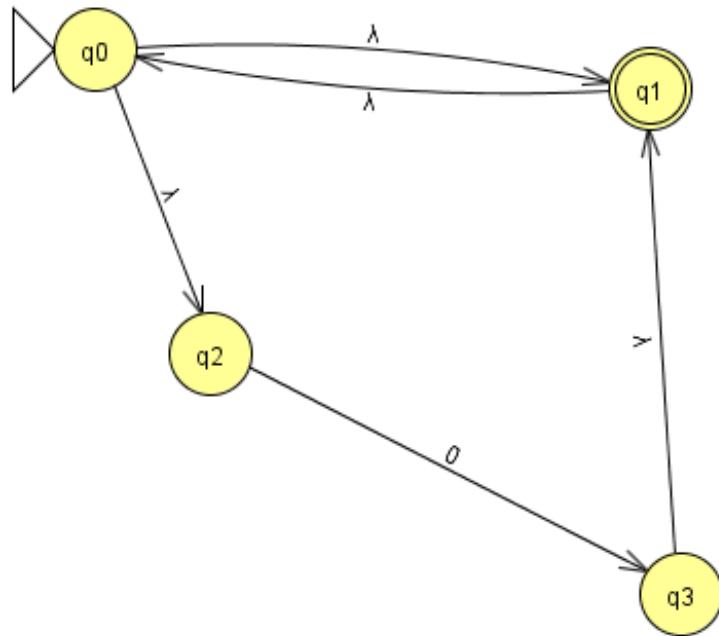


ADF minimal:



$0^*$ :

Autómata finito con transiciones nulas:





# Descarga la APP de Wuolah.

Ya disponible para el móvil y la tablet.

Available on the  
App Store

GET IT ON  
Google Play



122

18

Ver mis op

Continúa d



Top de tu g



7CR



Rocio



pony

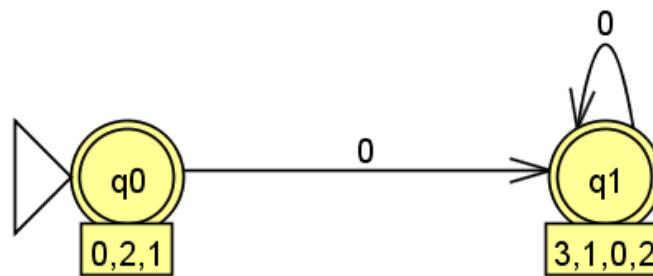


Inicio

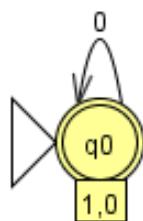


Asigni

ADF:



ADF minimal:



Construir una expresión regular para las palabras en las que el número de ceros es par.

$$1^* (01^*01^*)^*$$

Cualquier cadena que contenga un numero par de 0s

De tal forma que empieza en 1 o no y puede haber un numero par de 0s y puede contener 1s.  
Ejemplos: el conjunto vacío, 00, 1, 100, 10101, ...

Construir una expresión regular para las palabras que contengan a 0110 como subcadena.

$$(0+1)^*0110(0+1)^*$$

Construir una expresión regular para el conjunto de palabras que empiezan por 000 y tales que esta subcadena sólo se encuentra al principio de la palabra.

$$(000)(1+10+100)^*$$

Construir una expresión regular para el conjunto de palabras que tienen a 000 o a 101 como subcadena

$$(0+1)^*(000+101)(0+1)^*$$

¿Cómo pasar de una expresión regular a un autómata?

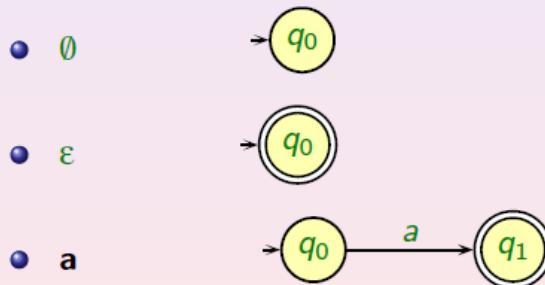
Estos autómatas que se hacen son con transiciones nulas.

**Dada una expresión regular existe un autómata finito que acepta el lenguaje asociado a esta expresión regular.**

Vamos a demostrar que existe un AFND con transiciones nulas. A partir de él se podría construir el autómata determinista asociado.

La construcción del autómata va a ser recursiva.

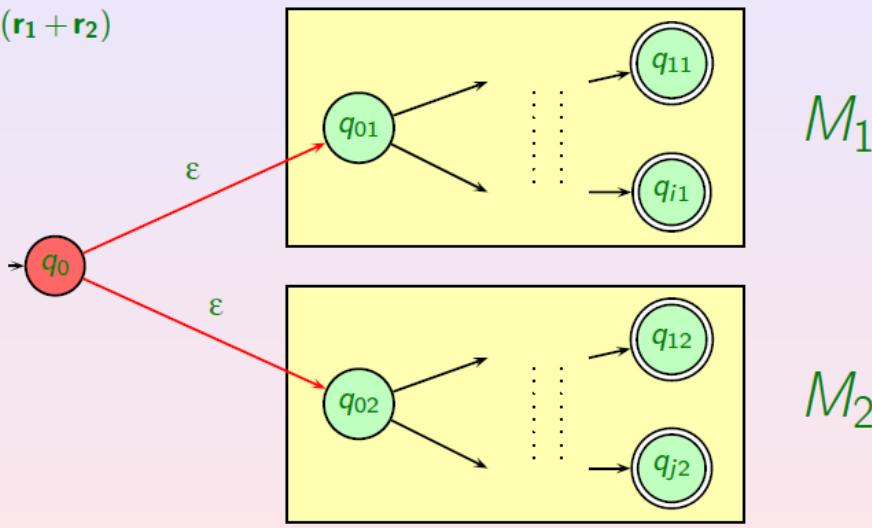
Para las expresiones regulares iniciales tenemos los siguientes autómatas:



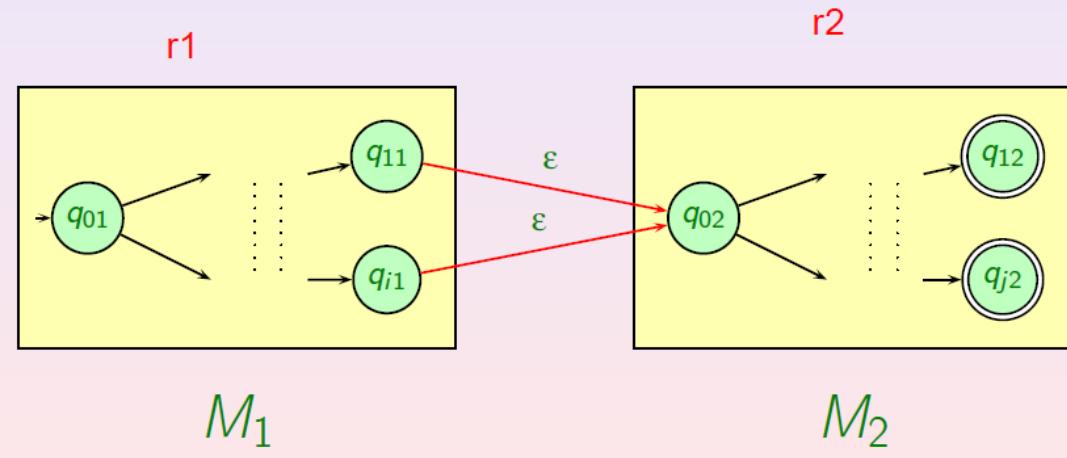
La Unión lo hace de esta forma:

Si  $M_1$  es el autómata que acepta el mismo lenguaje que el representado por  $r_1$  y  $M_2$  el que acepta el mismo lenguaje que el de  $r_2$ , entonces

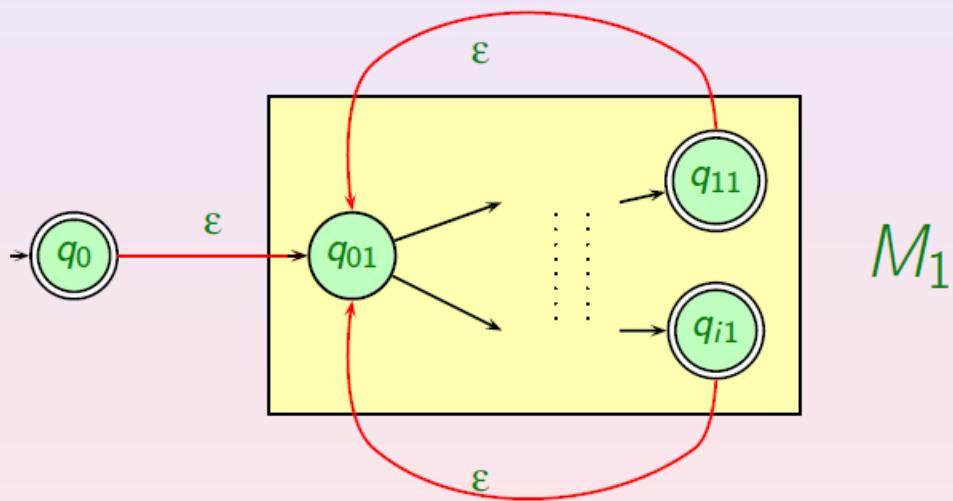
• **Unión ( $r_1 + r_2$ )**



- **Concatenación:** El autómata para la expresión  $(r_1 r_2)$  es

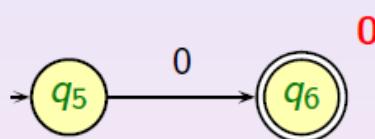


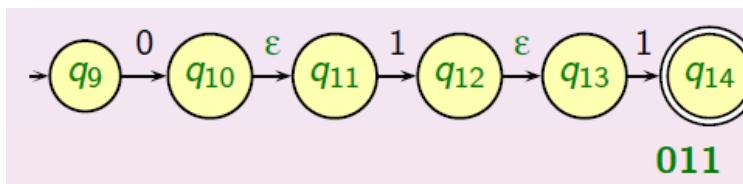
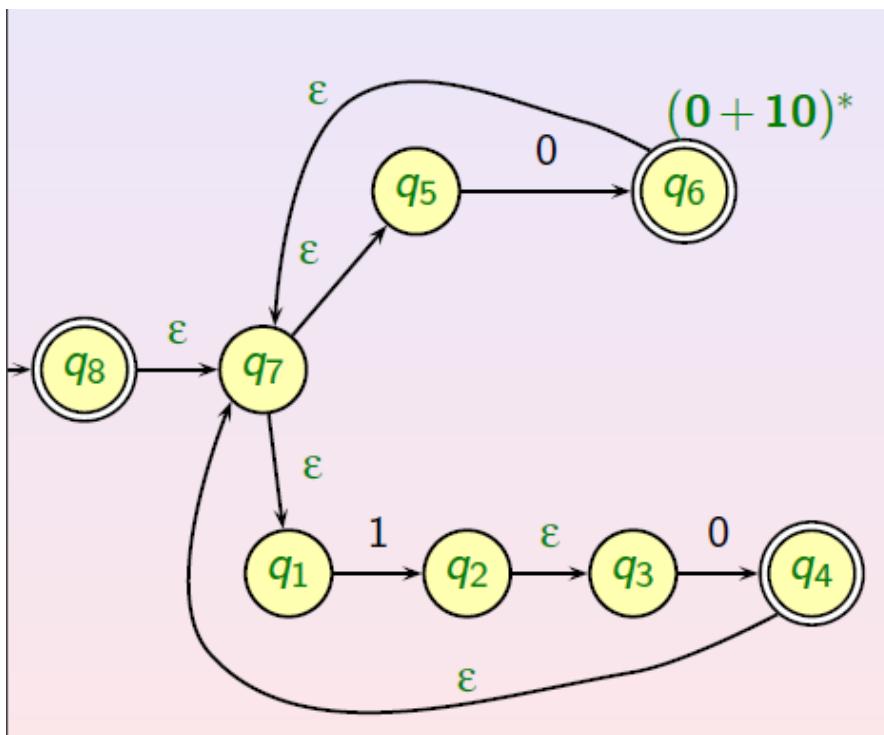
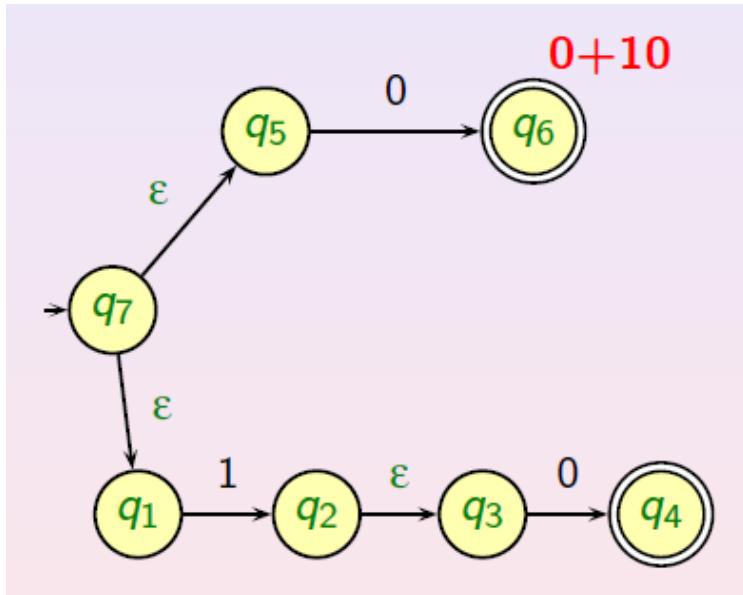
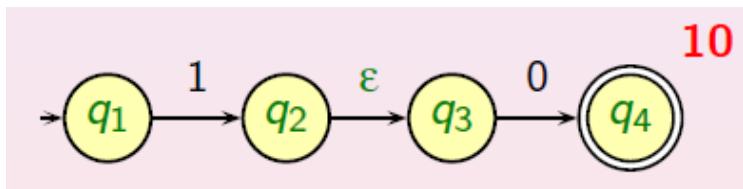
- **Clausura:** El autómata para  $r_1^*$  es



Ejemplo:

Encontrar un autómata que acepte el mismo lenguaje que el asociado a la expresión regular  $(0 + 10)^*011$







122

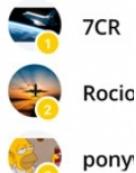
18

Ver mis op

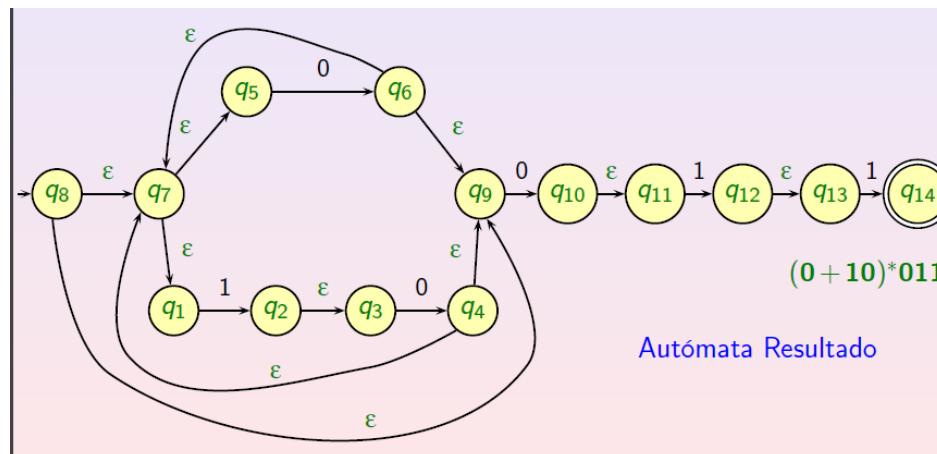
Continúa d



Top de tu gu



Inicio Asigni



Expresiones regulares usando LEX

| Expresión Regular   | Significado                                      |
|---------------------|--|
| Caracteres Normales | Ellos mismos                                     |
| $+, *$              | Superíndices $+, ^*$                             |
| $[ ]$               | La unión de los lenguajes                        |
| $[a - b]$           | Cualquier símbolo entre corchetes                |
| $[^...]$            | todos los caracteres entre $a$ y $b$             |
| $?$                 | El complementario de $[...]$                     |
| $\{ nombre \}$      | 0 ó 1 repetición de lo anterior                  |
| $\{ n \}$           | Se substituye la e.r. <i>nombre</i>              |
| $\{ n, m \}$        | $n$ repeticiones de la anterior e.r.             |
| $\backslash *$      | entre $n$ y $m$ repeticiones de la anterior e.r. |
| $"..."$             | El carácter $*$                                  |
| $^, $$              | Los caracteres entre comillas literalmente       |
|                     | Principio, fin de línea                          |

 $+$  --> 1 o mas ocurrencias $*$  --> 0 o muchas ocurrencias $|$  --> or  $(1 | 0)$   $(1 | 0)$  $[a-z]$  --> representa todos los caracteres del alfabeto en minuscula $[0-9]$  --> representa todos los digitos del 0 al 9. $[^a-z]$  --> cualquier cosa que no sea de la **a** a la **z**. Es decir excluye. $[0-9]?$  --> **0 o 1 digito**. $[0-9]\{3\}$  --> digitos de longitud 3 $[0-9]\{2,5\}$  --> digitos de longitud 2 a 5 (123, 12, 1234, 12345) $[0-9]\{2,\}$  --> digitos de longitud minima de 2 $.*$  --> cualquier cosa $\"+34"$  --> literalmente (+34) $^*[0-9]$  --> un digito al principio de la linea $[0-9]$$  --> un digito al final de la linea

ejemplo:

 $(0 | 1)^+$  --> cualquier conjunto de 0s o 1s, pero que al menos haya 1 ocurrencia.

# Descarga la APP de Wuolah.

## Ya disponible para el móvil y la tablet.

Available on the  
App StoreGET IT ON  
Google Play

### Estructura de un fichero lex:

The diagram illustrates the structure of a Lex file. It shows tokens (regular expressions) and actions (actions). A yellow speech bubble icon is at the top right.

|                                |              |    |                                   |
|--------------------------------|--------------|----|-----------------------------------|
| nombre1                        | er1          | {} | creamos las expresiones regulares |
| nombre2                        | er2          |    |                                   |
| nombrei                        | eri          |    |                                   |
|                                | declaglobal1 |    |                                   |
|                                | declaglobal2 |    |                                   |
| % %                            |              |    |                                   |
|                                | declalocal1  | {} | llama a las expresiones regulares |
|                                | declalocal2  |    |                                   |
| er1                            | accion1;     |    |                                   |
| er2                            | accion2;     |    |                                   |
| er3                            | accion3;     |    |                                   |
| % %                            |              |    |                                   |
| definiciones de funciones en C |              | {} | Main del fichero lex              |
|                                |              |    |                                   |

### Variables y procedimientos:

- **yylex()** → Programa que reconoce las expresiones regulares y ejecuta las acciones.
- **main()** → Programa principal. Por defecto solo llama a **yylex()**.
- **yywrap()** → función que se ejecuta cuando acaba **yylex()**. Si devuelve 1 termina **yylex()**, 0 no ha terminado **yylex()**.
- **Yyin** → Fichero de entrada (**stdin** por defecto)
- **yytext** → variable que contiene la cadena reconocida por **yylex()**
- **yylen** → longitud de la cadena reconocida

## Ejemplo

```
car          [a-zA-Z]
dígito      [0-9]
signo       (\-|\+)
suc          ({dígito}+)
enter        ({signo}?{suc})
real1        ({enter}\.{dígito}*)
real2        ({signo}?\.{suc})
int    ent=0, real=0, ident=0, sumaent=0;
%%
int    i;
{enter}           {ent++; sscanf(yytext,"%d",&i); sumaent += i;
                  printf("Número entero%s\n",yytext);}
({real1}|{real2}) {real++; printf("Num. real%s\n",yytext);}
{car}({car}|{dígito})* {ident++; printf("Var. ident.%s\n",yytext);}
.\n                {;}
%%
yywrap()
{printf("Número de Enteros%d, reales%d, ident%d,
         Suma de Enteros%d",ent,real,ident,sumaent); return 1;}
```

## Procedimiento

- ① Crear fichero `ejemplo` con el contenido anterior
- ② Ejecutar `lex` con el fichero creado:  
`lex ejemplo`
- ③ Compilar el programa que crea `lex`:  
`gcc lex.yy.c -o prog -llex`
- ④ ejecutar el programa `prog < Entrada > Salida`
- ⑤ NOTA: Si usais `flex` en lugar de `lex` quizás tendréis que cambiar las opciones de compilación:  
`gcc lex.yy.c -o prog -lfl`

Reservados todos los derechos.  
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

**WUOLAH**

Descarga la app de Wuolah desde tu store favorita