Autómatas finitos

Dr. José Lázaro Martínez Rodríguez

Por qué necesitamos modelos abstractos?

- Al describir un problema debemos de usar un lenguaje que pueda ser comprensible de forma única
  - Sin dar espacio a malas interpretaciones
  - Sin ambigüedad
  - Ser específicos
  - Llevar al mismo resultado

Caracterizando un ascensor

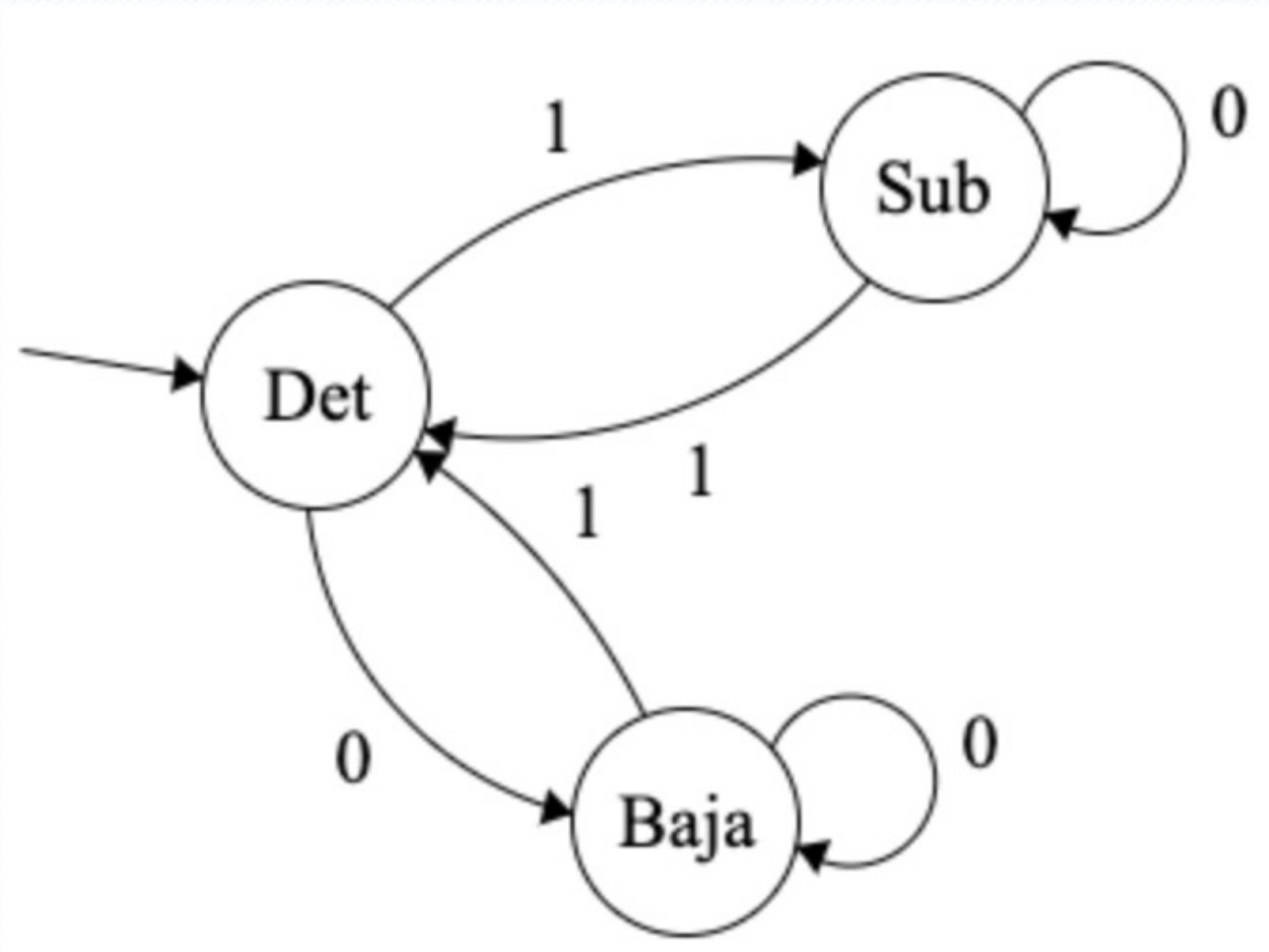
- Un ascensor tiene tres estados
  - Detenido
  - Subiendo
  - Bajando
- Cuando esta detenido y es 'activado'
  - Si *nivel seleccionado* > *nivel actual*, pasa al estado 'subiendo'
  - Si *nivel seleccionado* < *nivel actual*, pasa al estado 'bajando'

Caracterizando un ascensor

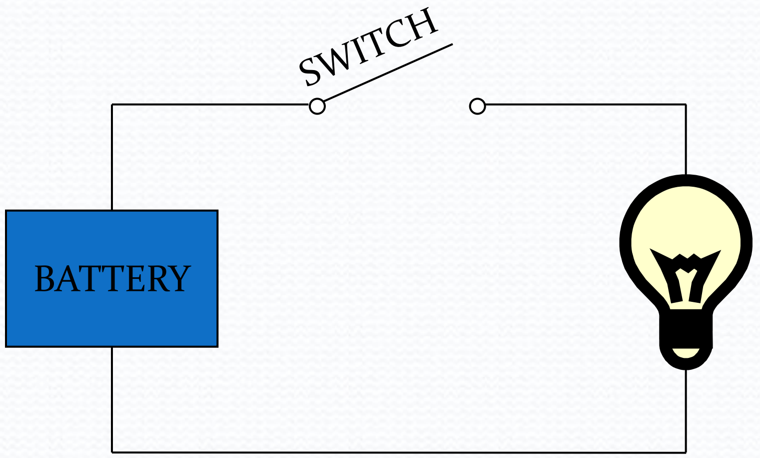
- Si el ascensor se encuentra en el estado ‘subiendo’
  - Si *nivel seleccionado* > *nivel actual*, permanece en estado ‘subiendo’
- Si el ascensor se encuentra en el estado ‘bajando’
  - Si *nivel seleccionado* < *nivel actual*, permanece en estado ‘bajando’

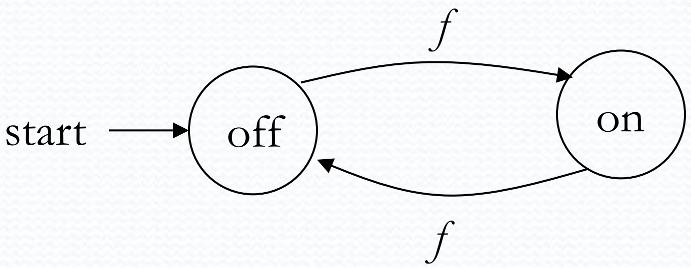
  - Si *nivel seleccionado* = *nivel actual*, el ascensor pasa al estado ‘detenido’

Caracterizando un ascensor



Una "computadora" simple





input: switch

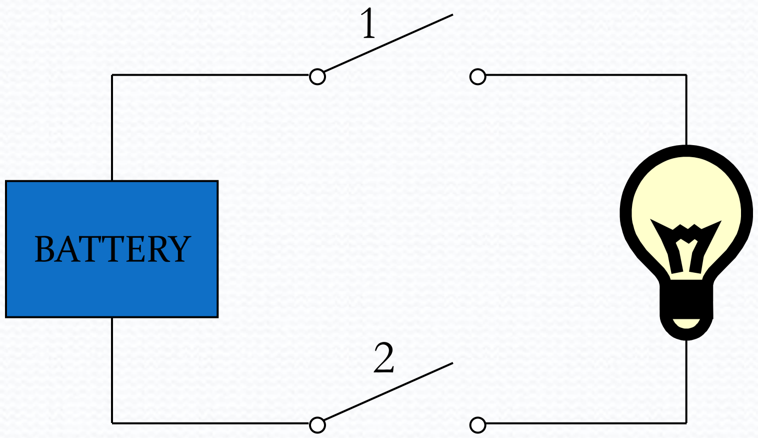
output: foco encendido

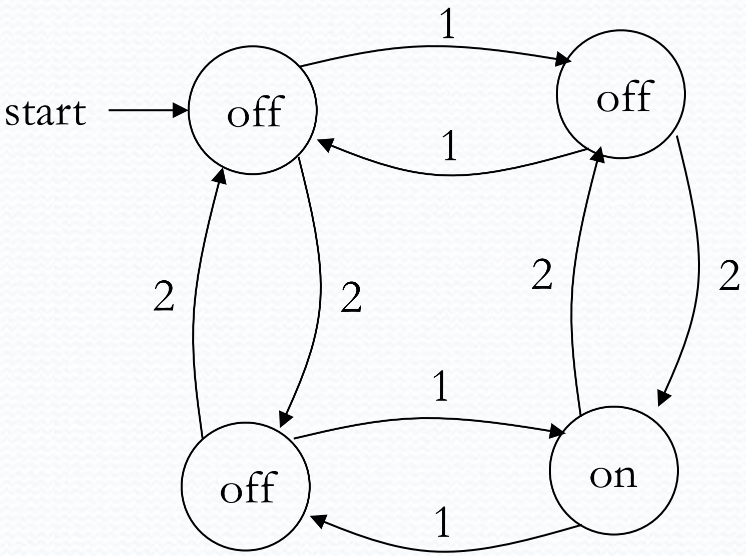
acciones: f para "flip switch"

estados: on, off

El foco esta encendido si y solo si hubo un número impar de vueltas

Otra "computadora"





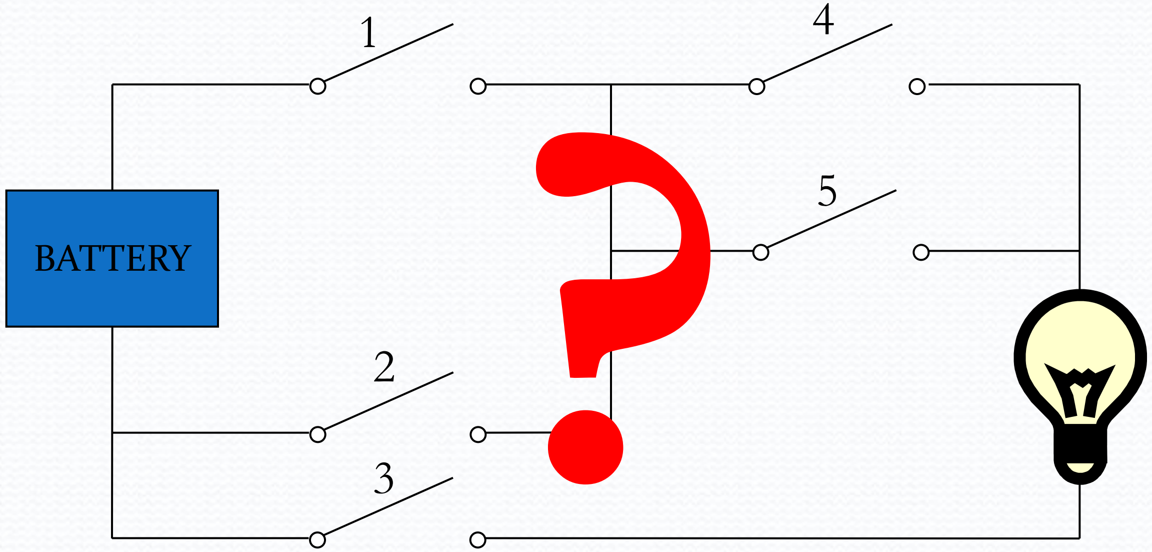
inputs: switches 1 y 2

acciones: 1 for "flip switch 1"
2 for "flip switch 2"

estados: on, off

la bombilla está encendida si y sólo si los dos interruptores han sido accionados un número impar de veces

Un problema de diseño



¿Puedes diseñar un circuito en el que la luz se encienda si y sólo si todos los interruptores se accionan exactamente el mismo número de veces?

Un problema de diseño

- Estos dispositivos son difíciles de razonar, porque pueden diseñarse de infinitas maneras.
- Al representarlos como dispositivos computacionales abstractos, o autómatas, aprenderemos a responder a estas preguntas

Autómatas finitos

- Los autómatas finitos son colecciones finitas de estados con reglas de transición que llevan de un estado a otro.
- Su aplicación original eran los circuitos de conmutación secuencial, en los que el "estado" era la configuración de los bits internos.
- Hoy en día, varios tipos de software pueden modelarse mediante autómatas finitos.

Representando Autómatas finitos

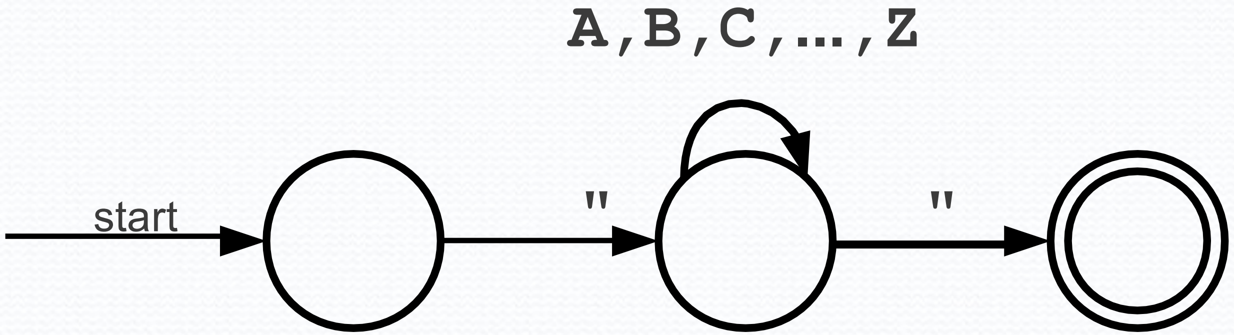
- La representación más sencilla suele ser un grafo.
- Nodos = estados.
- Las aristas (arcos) indican las transiciones de estado.
- Las etiquetas de las aristas indican la causa de la transición.

Implementando regex

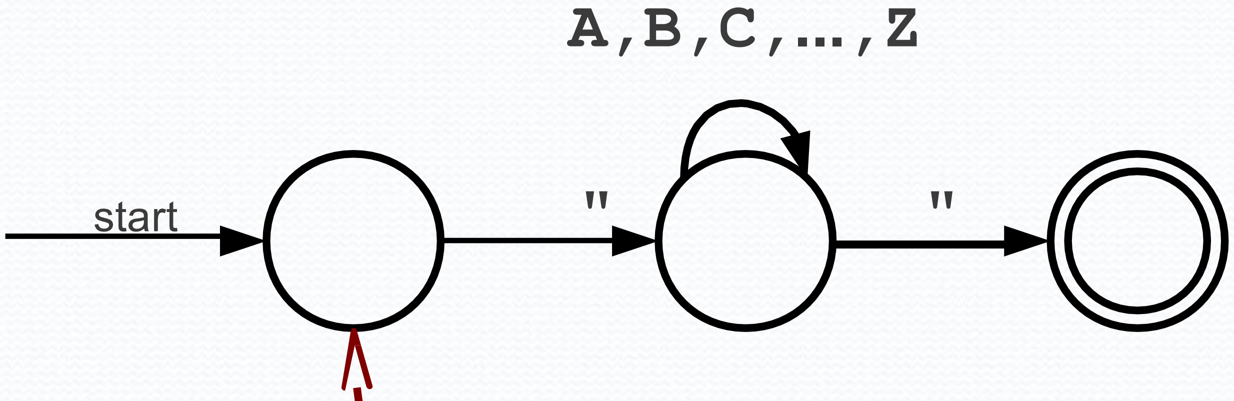
Las expresiones regulares pueden ser implementadas utilizando **autómatas finitos**.

- Existen dos tipos de autómatas finitos:
  - **NFA**s (**nondeterministic** finite automata), Autómatas finitos no-determinísticos
  - **DFA**s (**deterministic** finite automata), autómatas finitos determinísticos.
- Primero veremos algunos ejemplos

Un Autómata simple

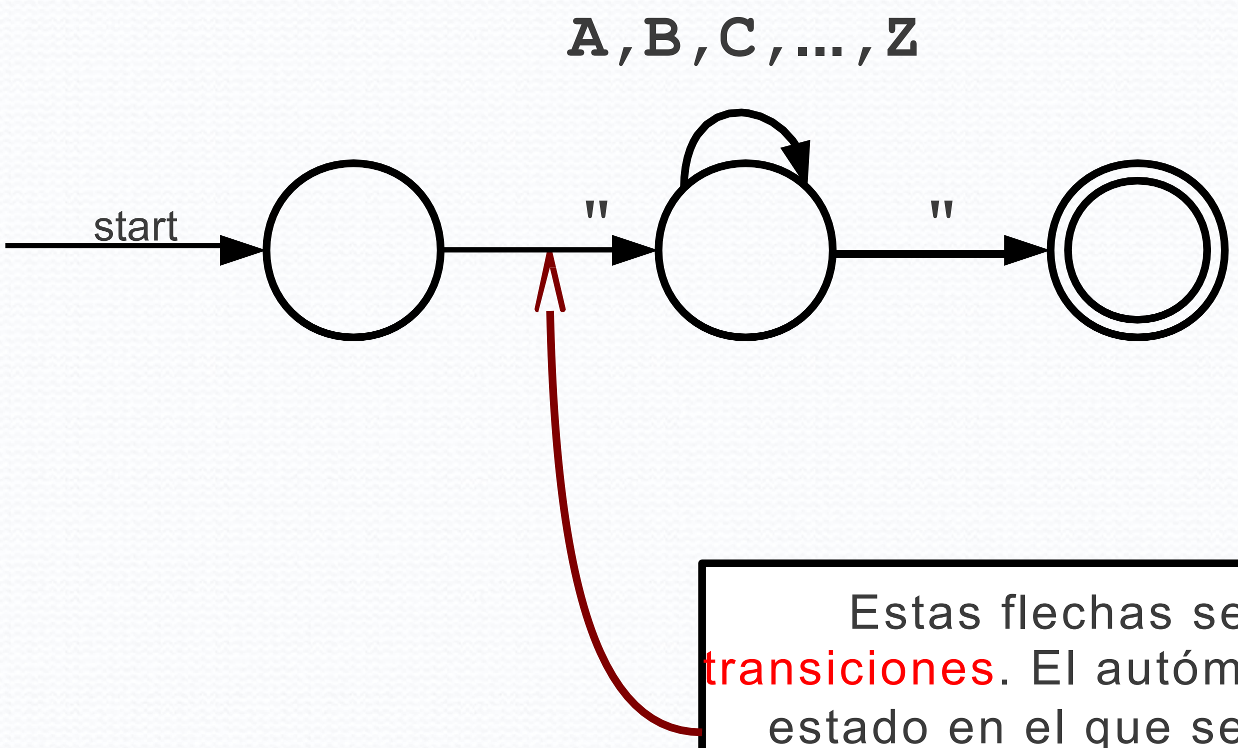


Un Autómata simple



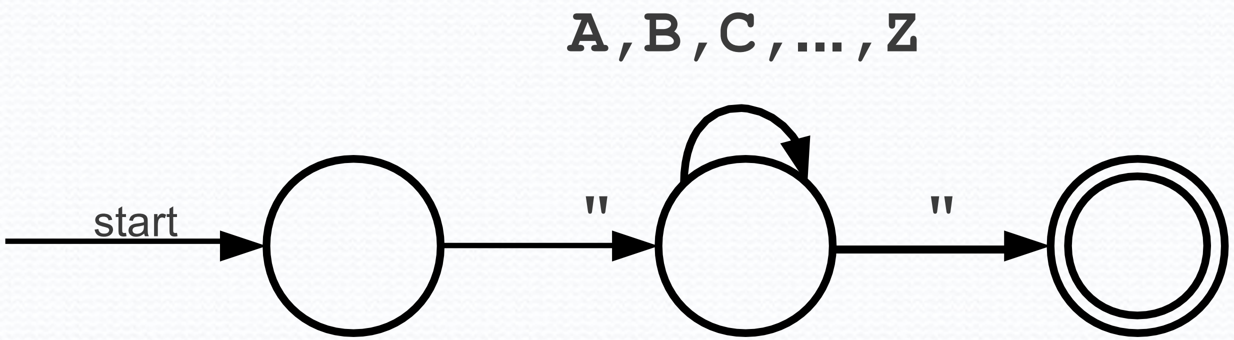
Cada círculo es un estado del autómata.
La configuración del autómata viene determinada por los estados en los que se encuentra.

Un Autómata simple



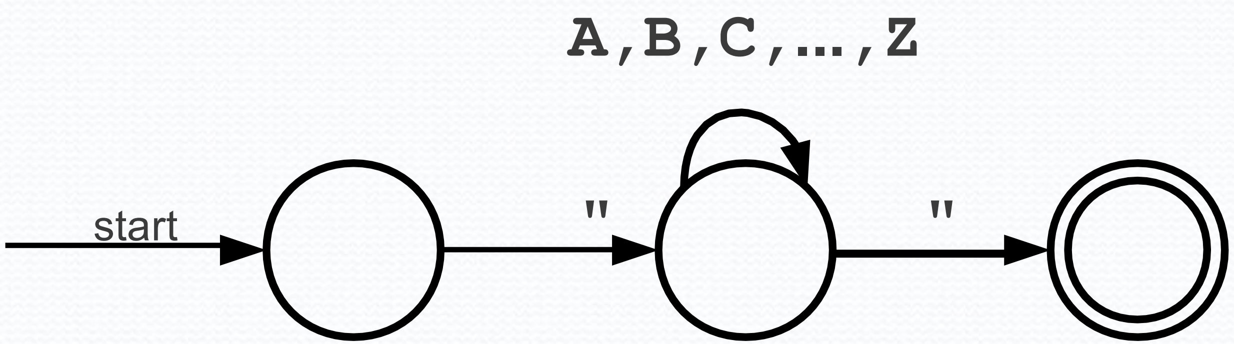
Estas flechas se llaman transiciones. El autómata cambia el estado en el que se encuentra siguiendo las transiciones.

Un Autómata simple



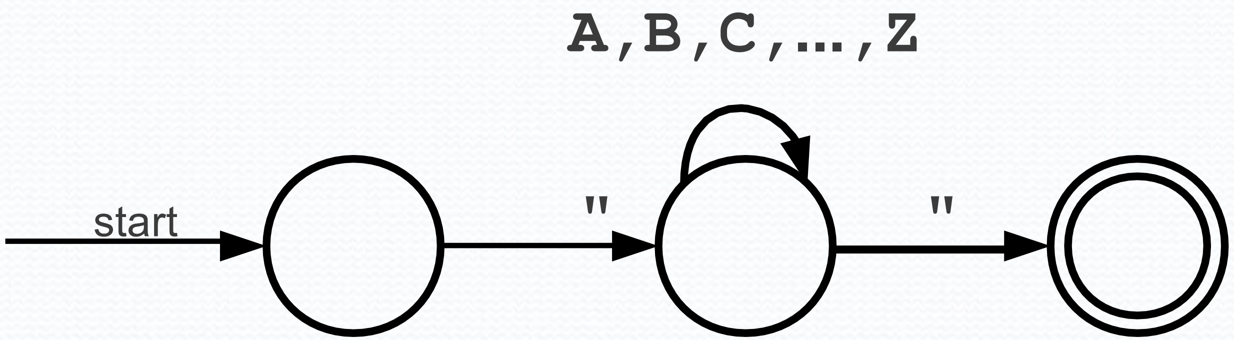
"	H	E	Y	A	"

Un Autómata simple

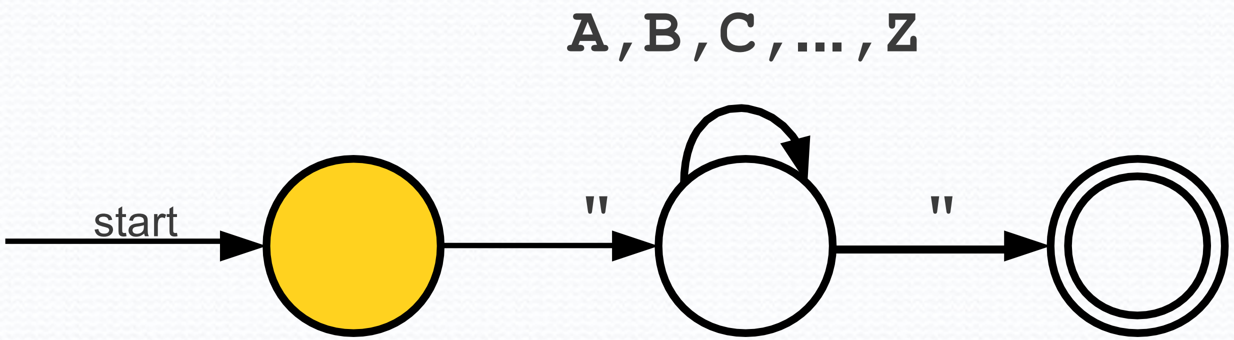


El autómata toma una cadena como entrada y decide si la acepta o la rechaza.

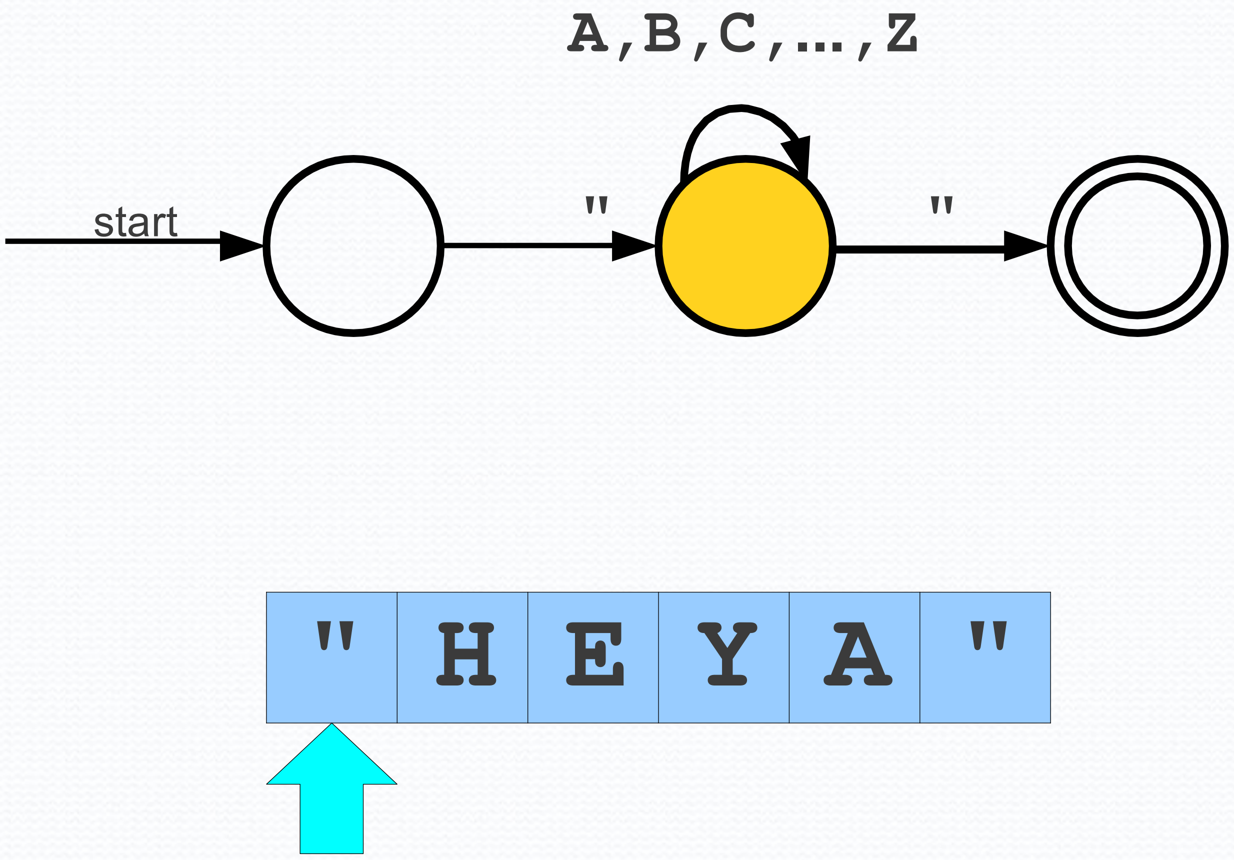
Un Autómata simple



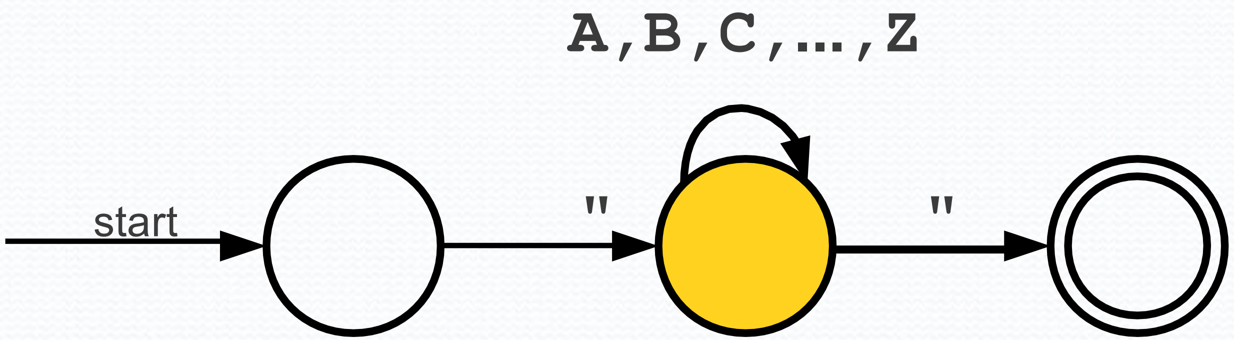
Un Autómata simple

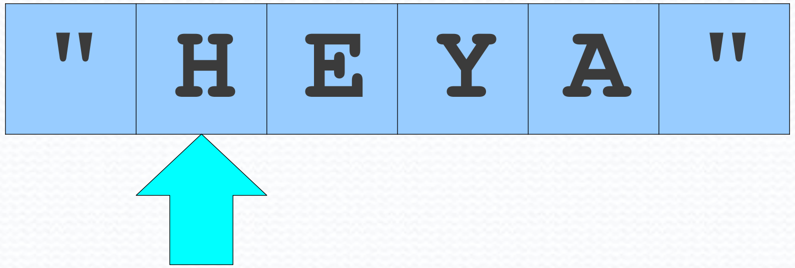


Un Autómata simple

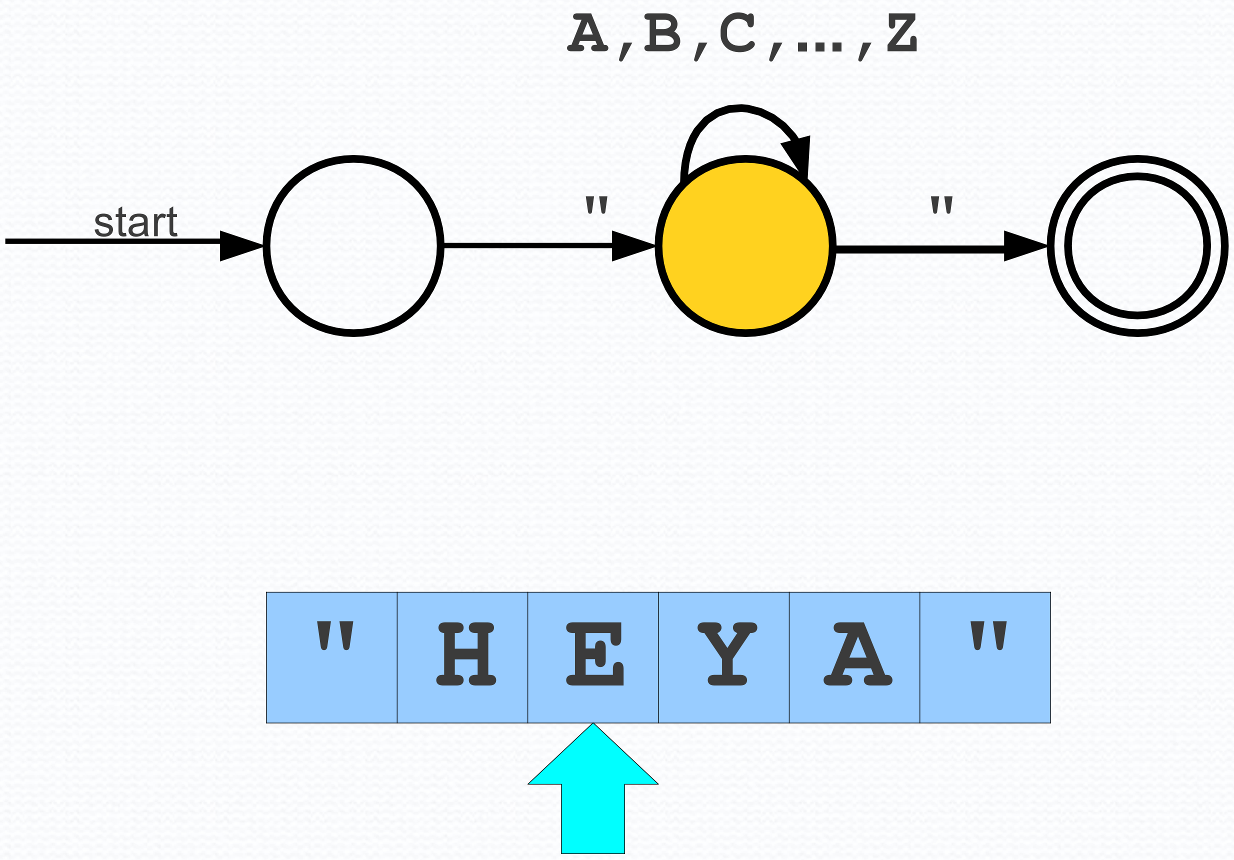


Un Autómata simple

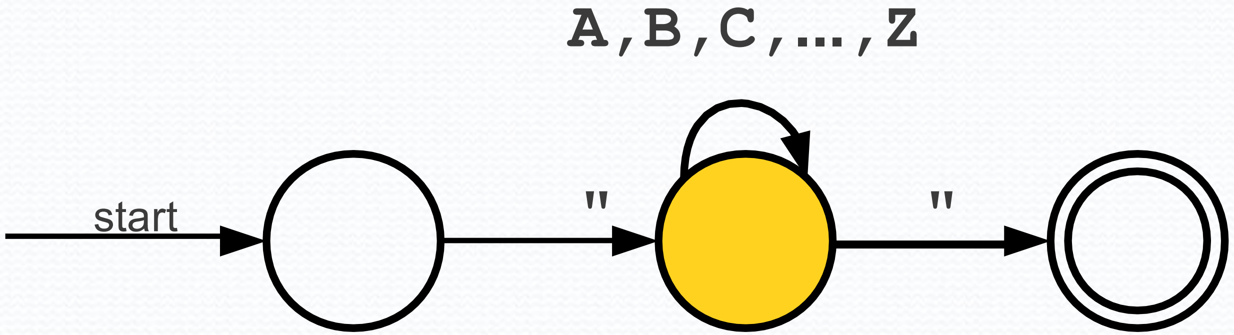


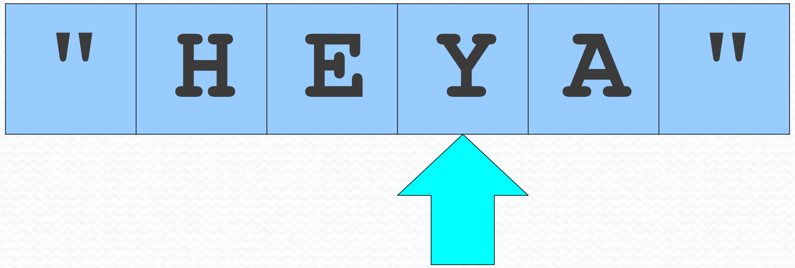


Un Autómata simple

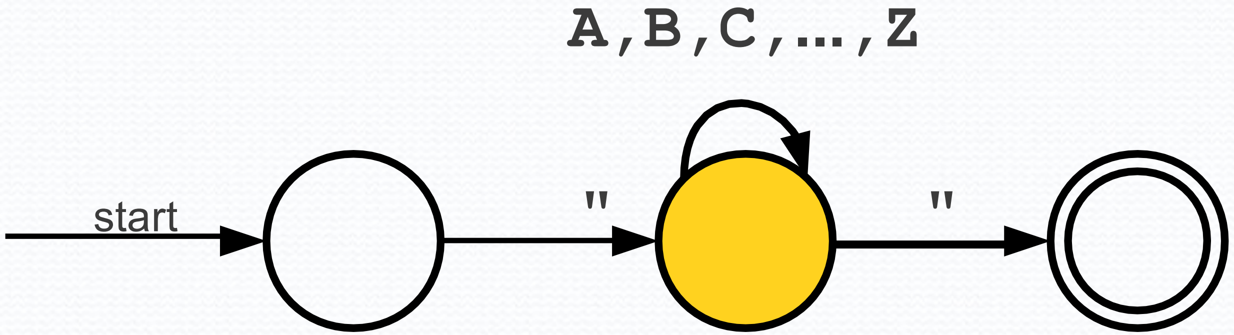


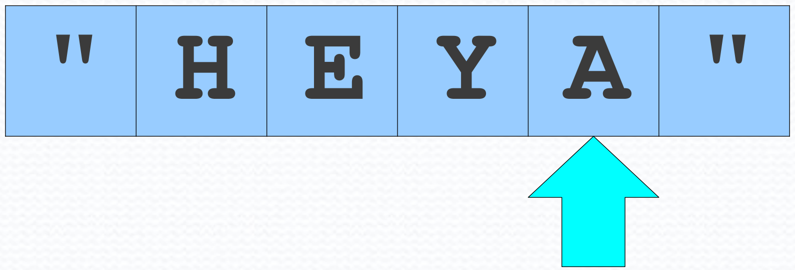
Un Autómata simple



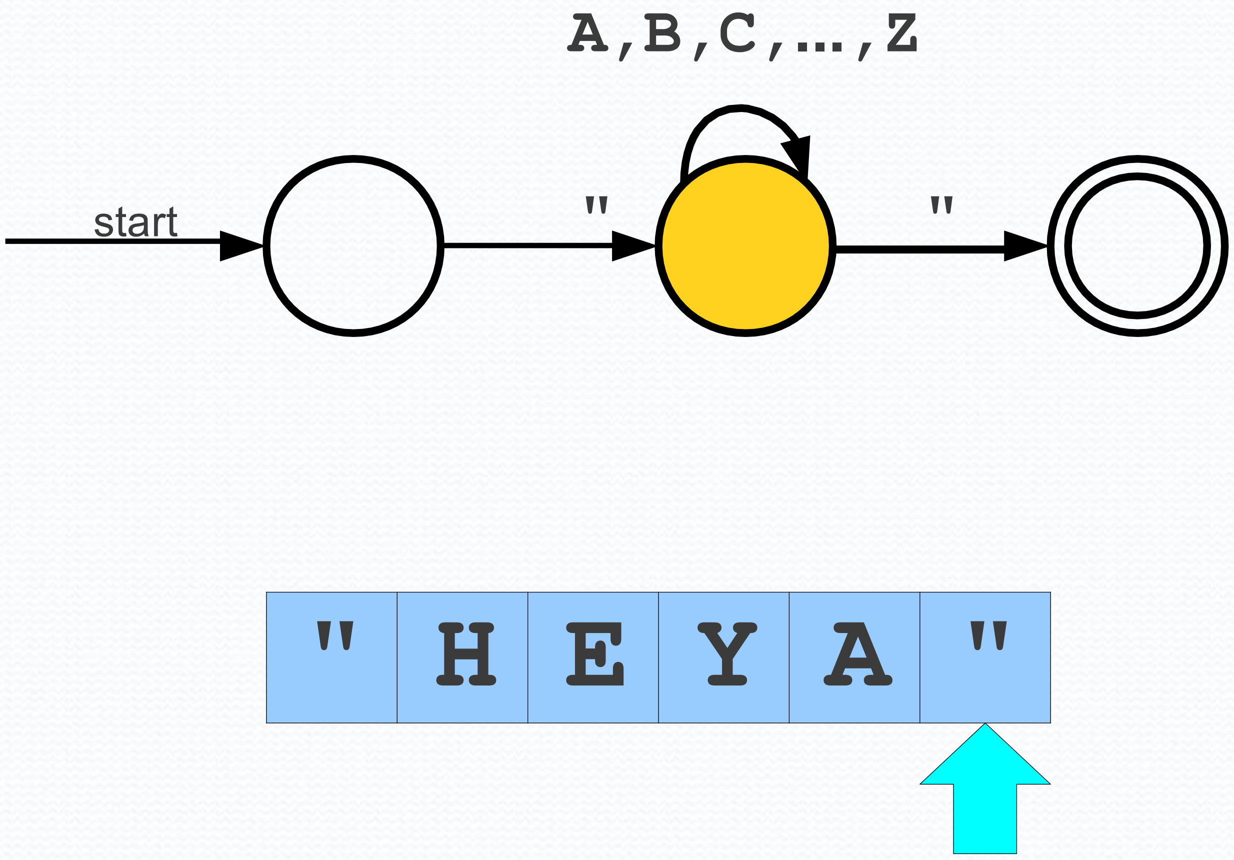


Un Autómata simple

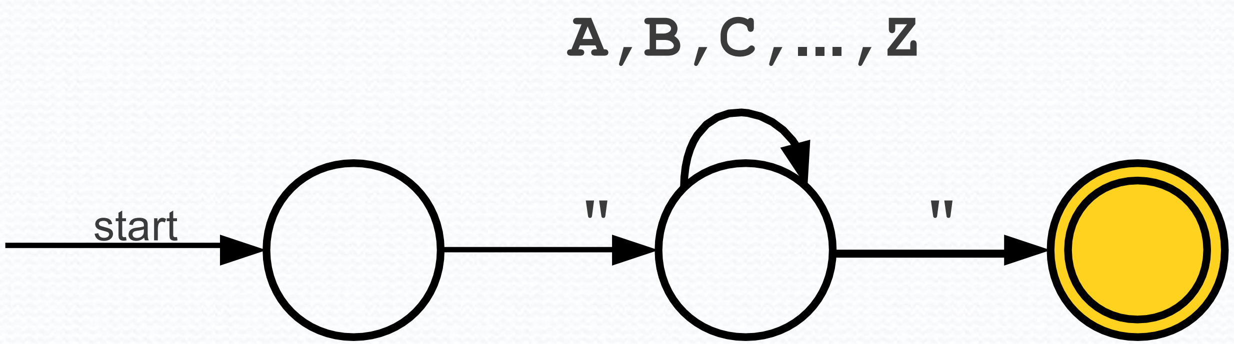




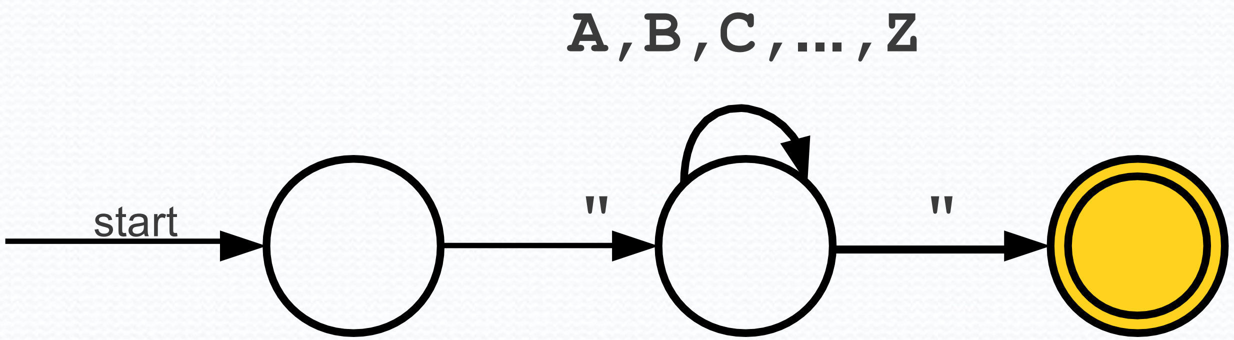
Un Autómata simple



Un Autómata simple

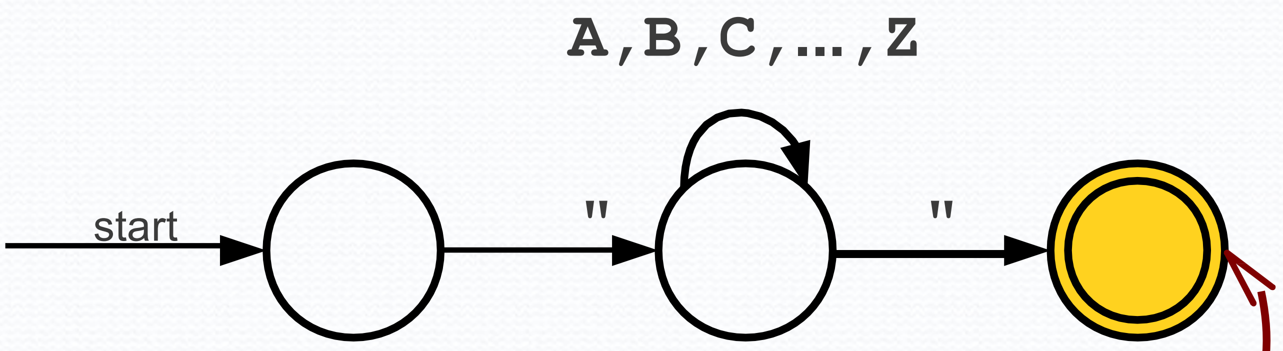


Un Autómata simple



"	H	E	Y	A	"

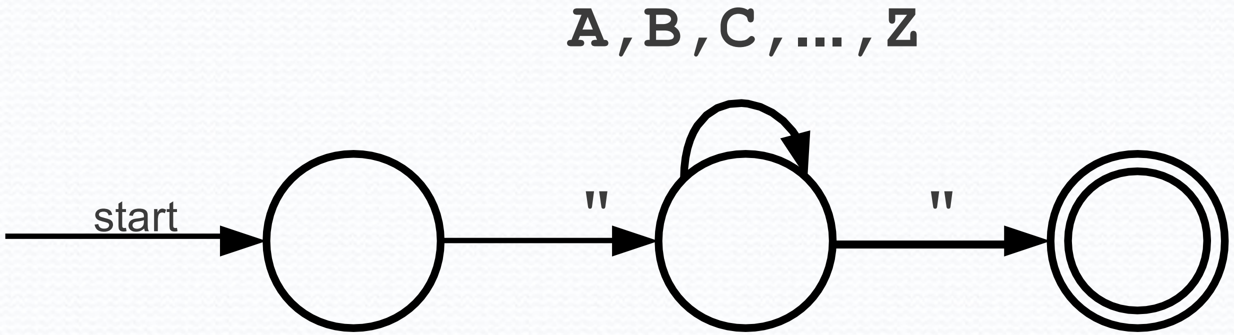
Un Autómata simple

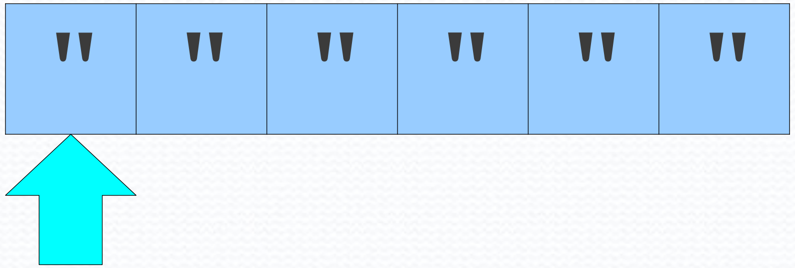


"	H	E	Y	A	"

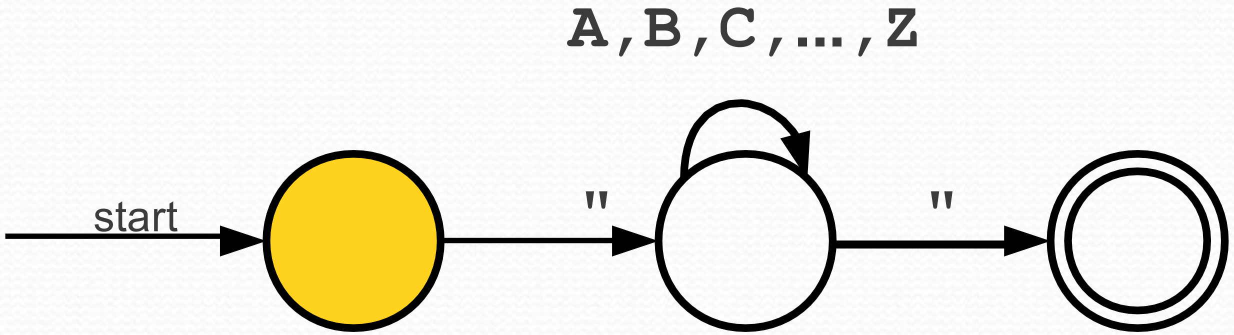
El doble círculo indica que es un estado de aceptación. El autómata acepta la cadena si termina en un estado de aceptación.

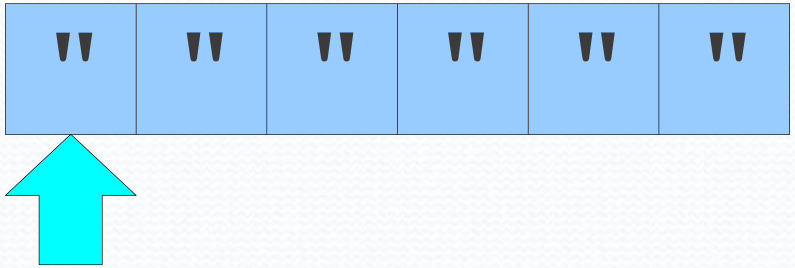
Un Autómata simple



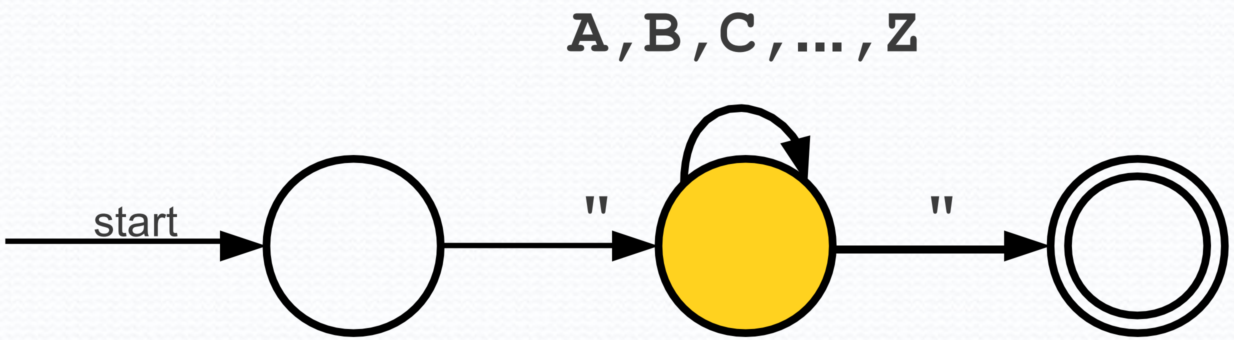


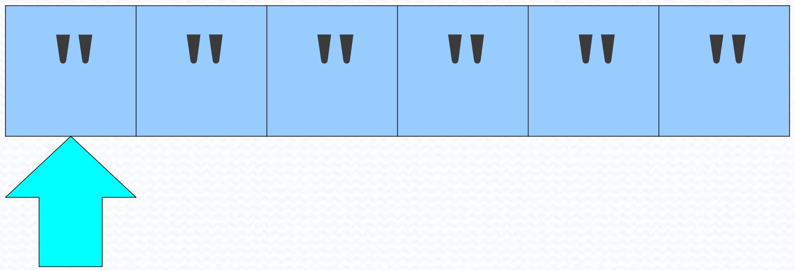
Un Autómata simple



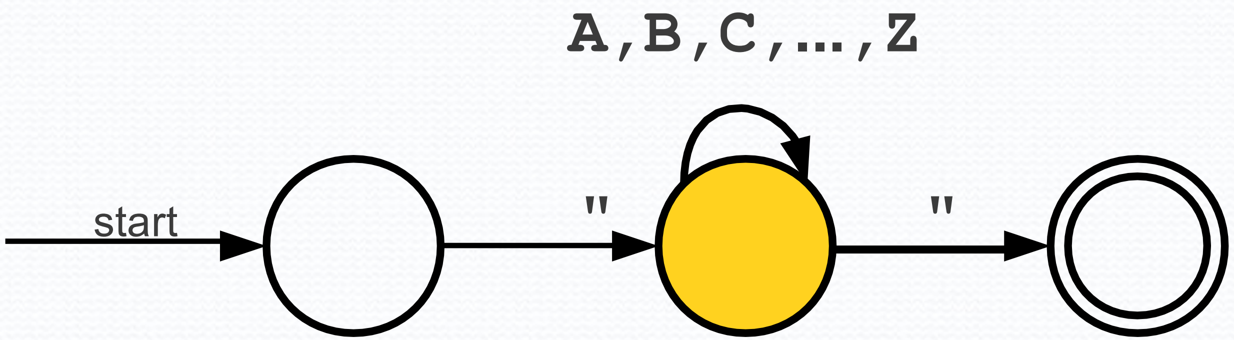


Un Autómata simple

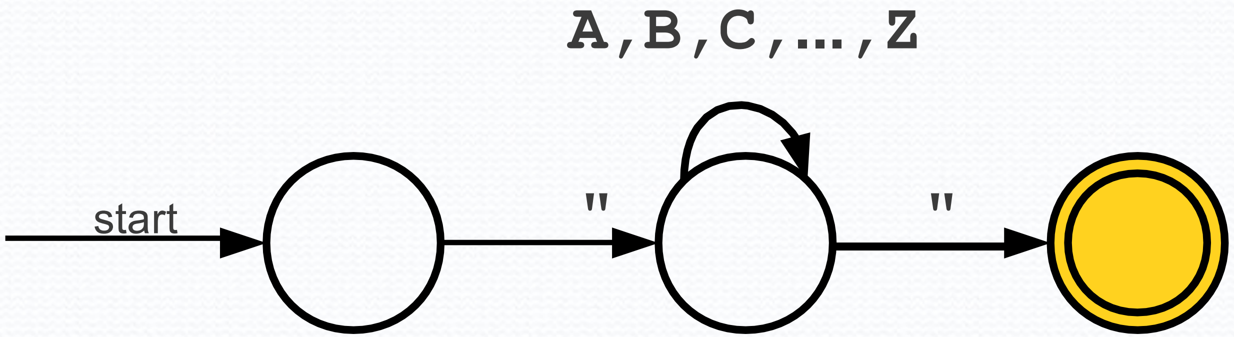




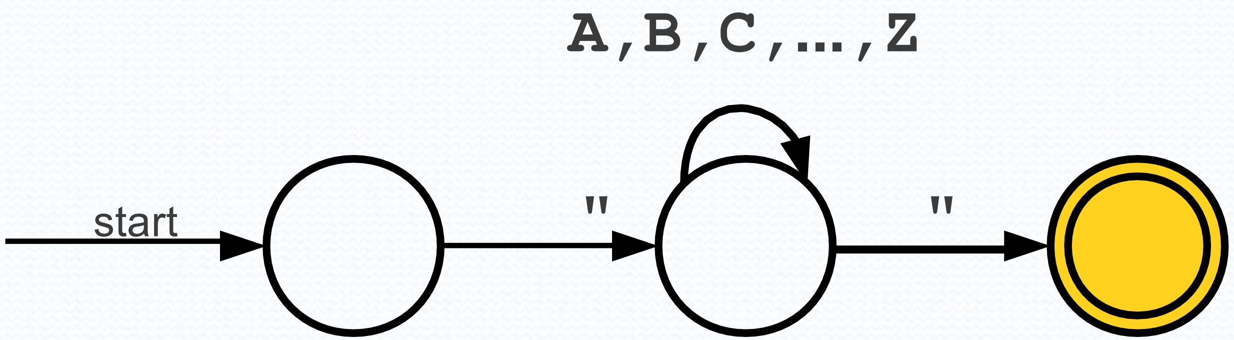
Un Autómata simple

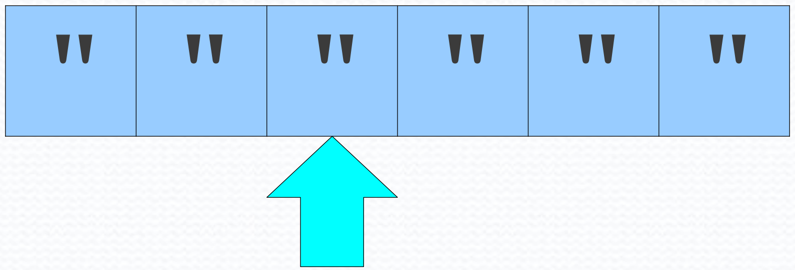


Un Autómata simple

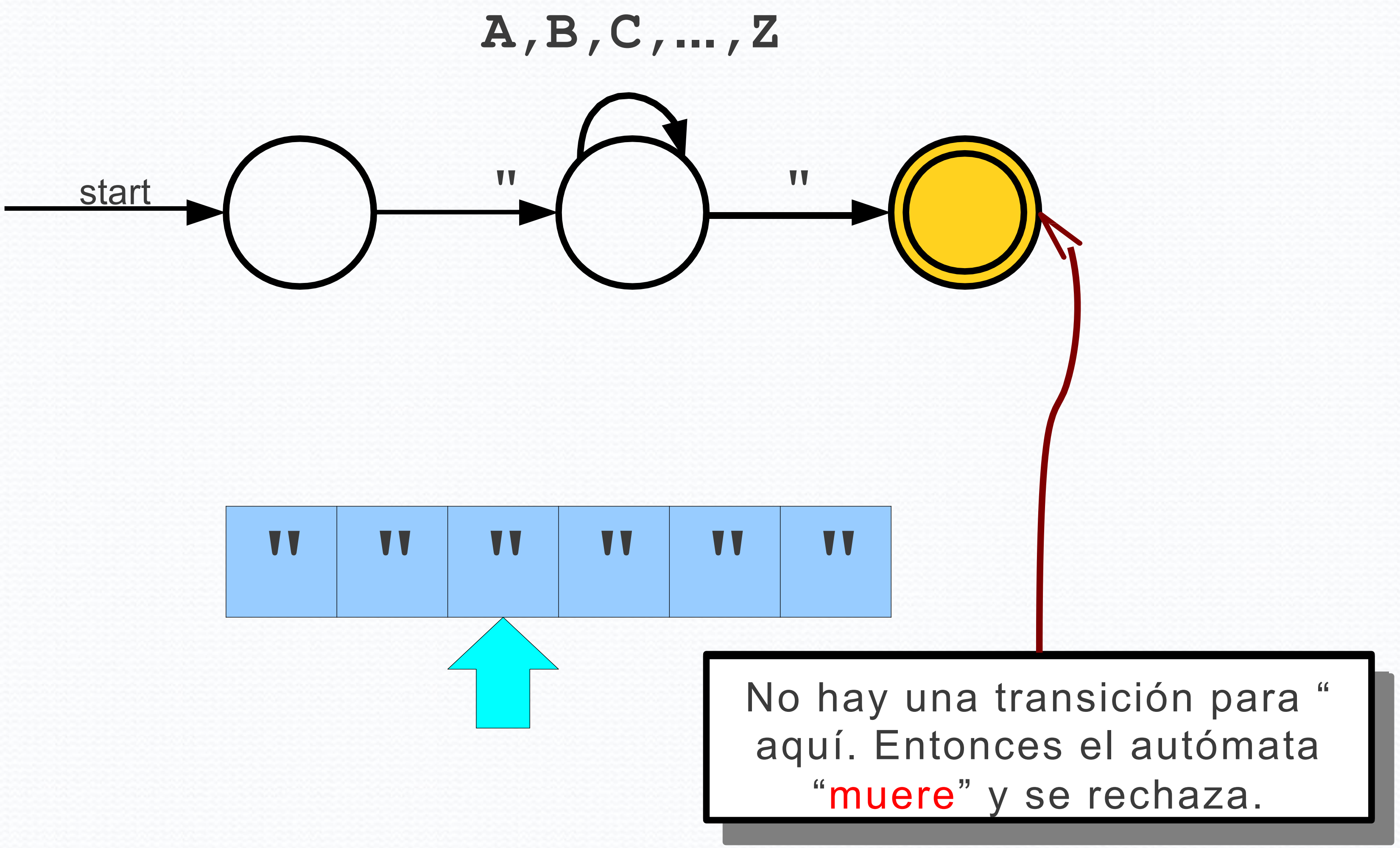


Un Autómata simple

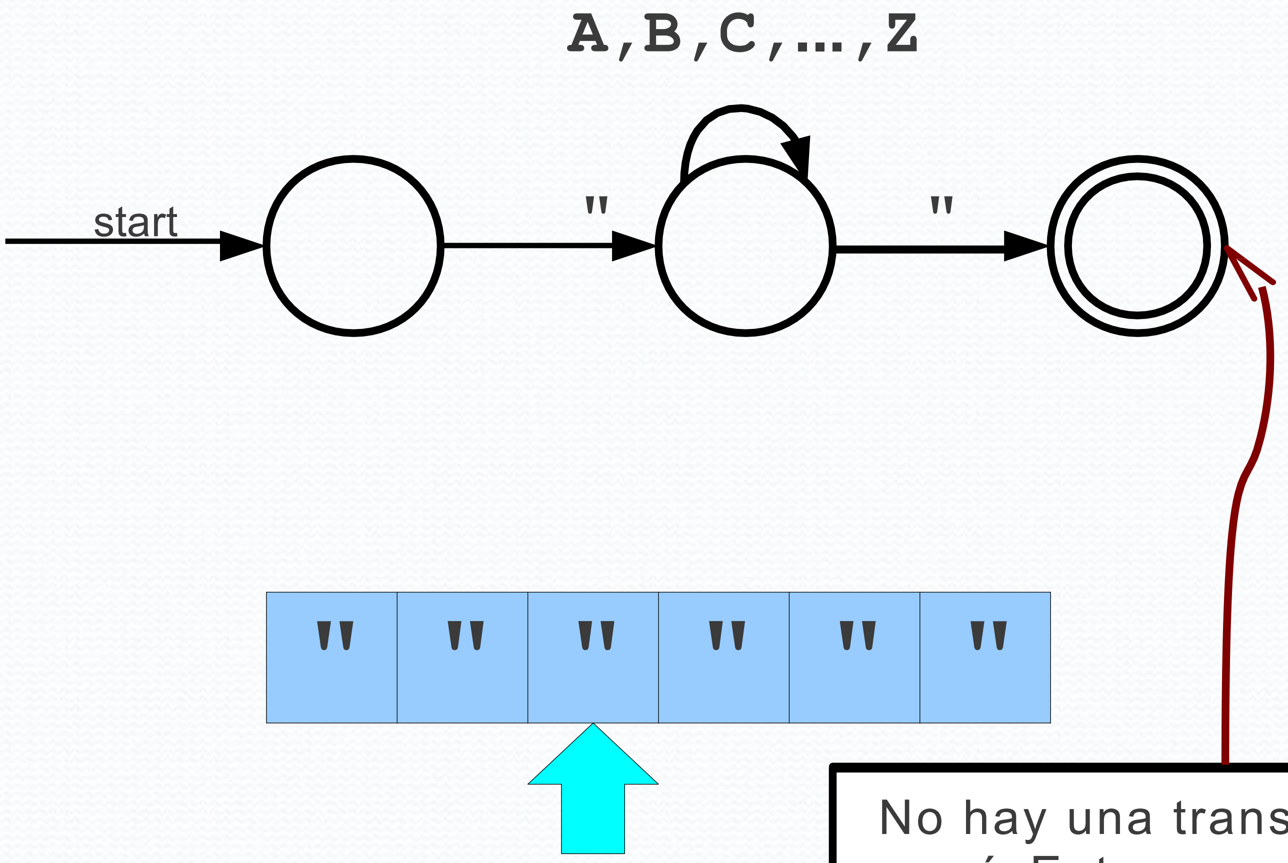




Un Autómata simple

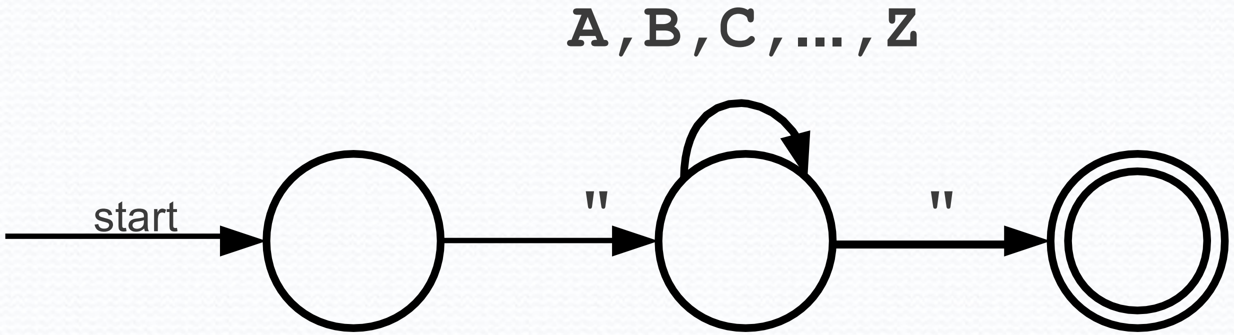


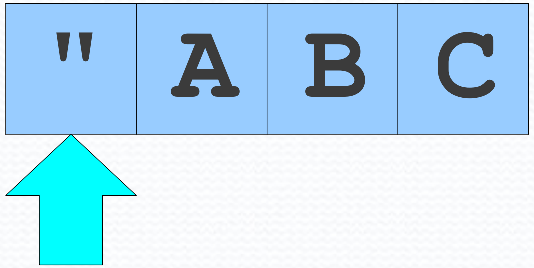
Un Autómata simple



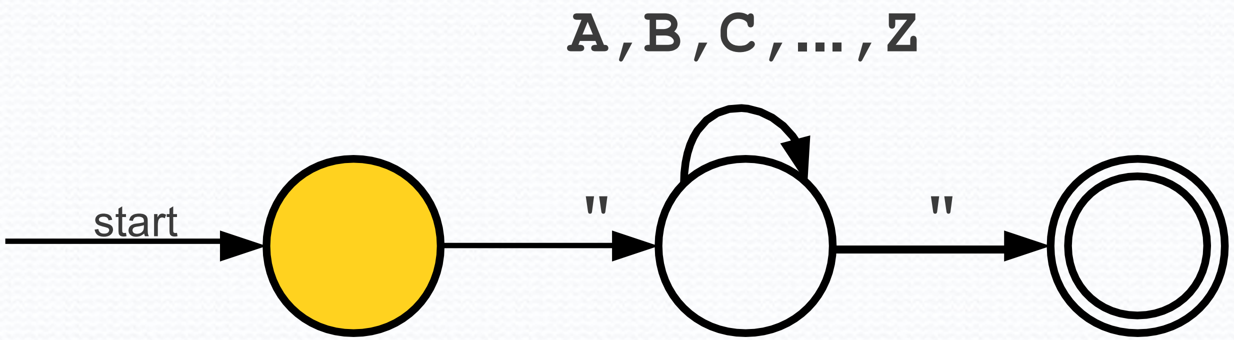
No hay una transición para "
aquí. Entonces el autómata
"muere" y se rechaza.

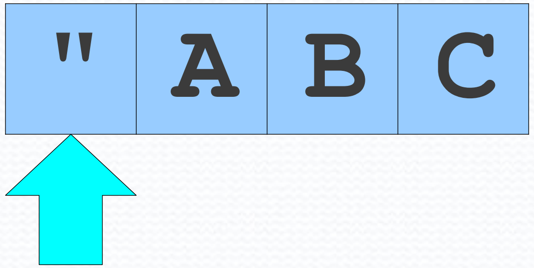
Un Autómata simple



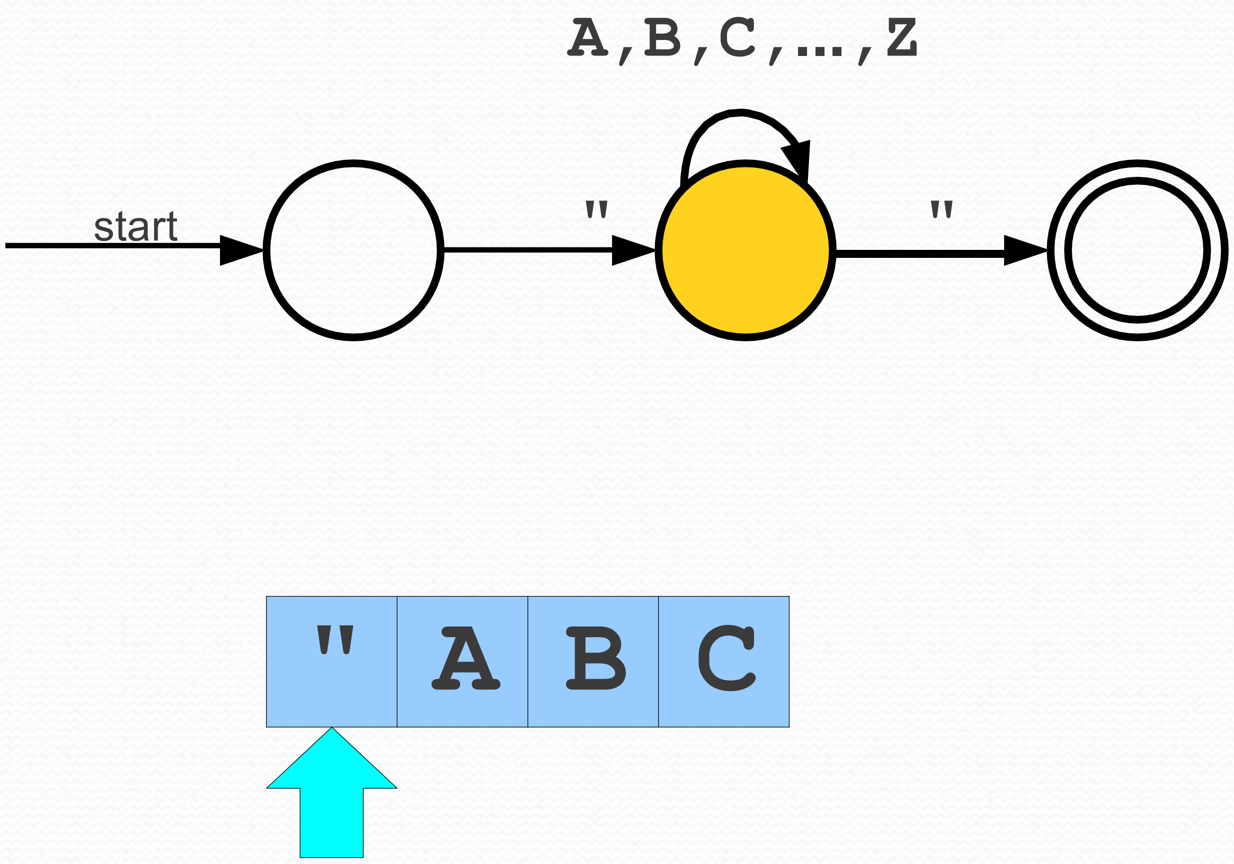


Un Autómata simple

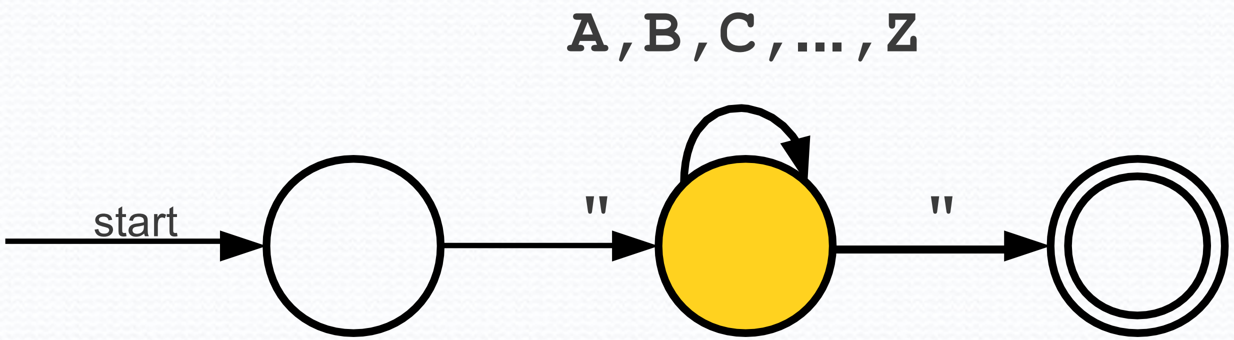


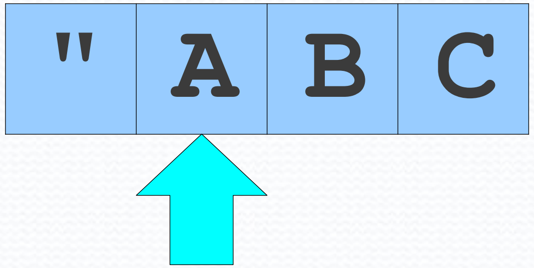


Un Autómata simple

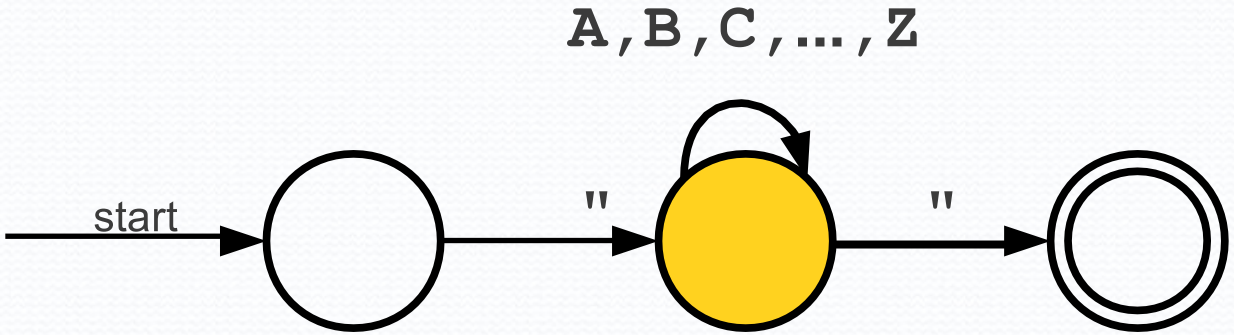


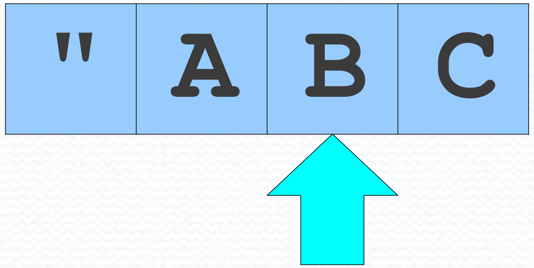
Un Autómata simple



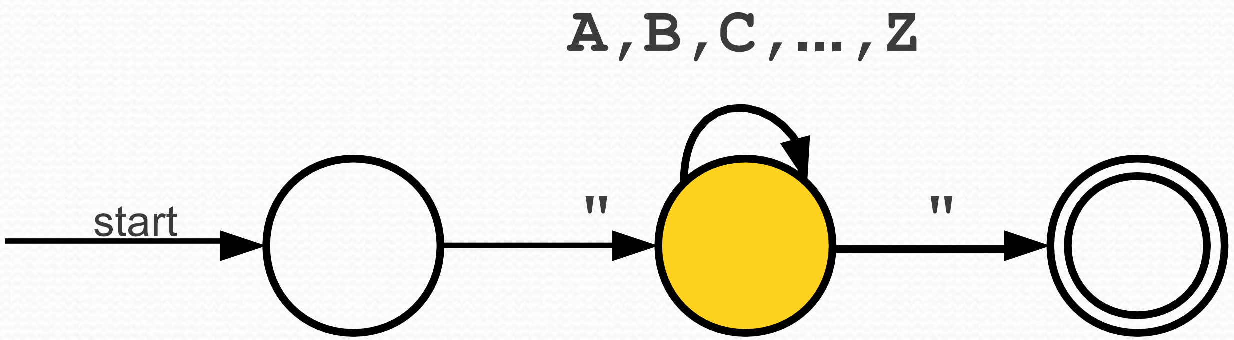


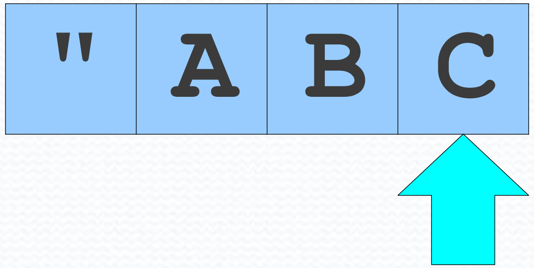
Un Autómata simple



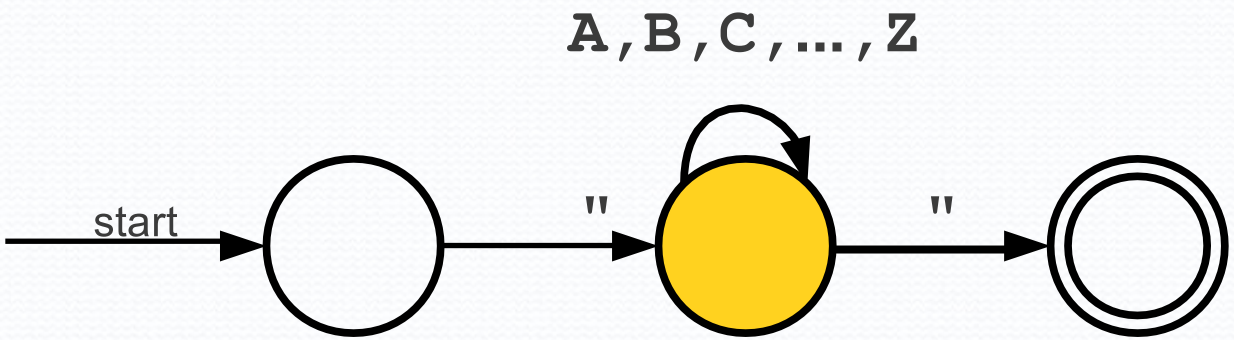


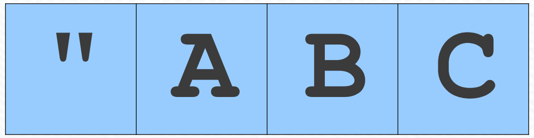
Un Autómata simple



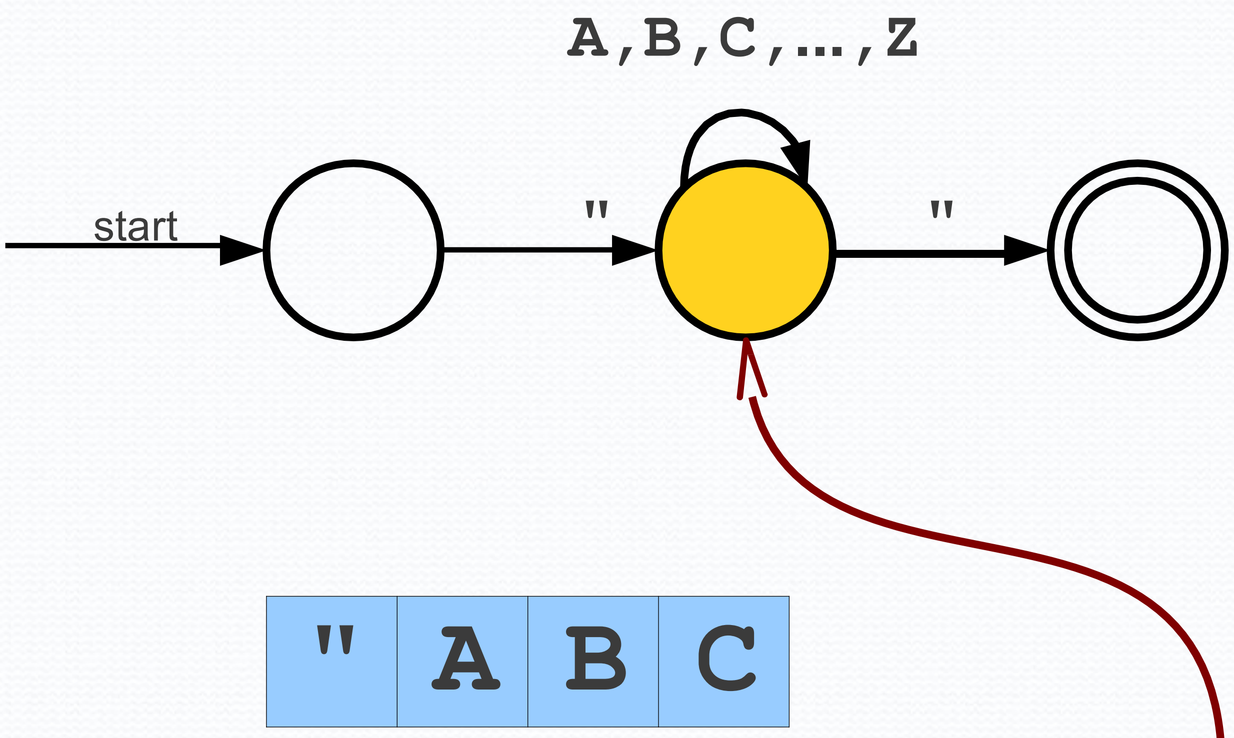


Un Autómata simple



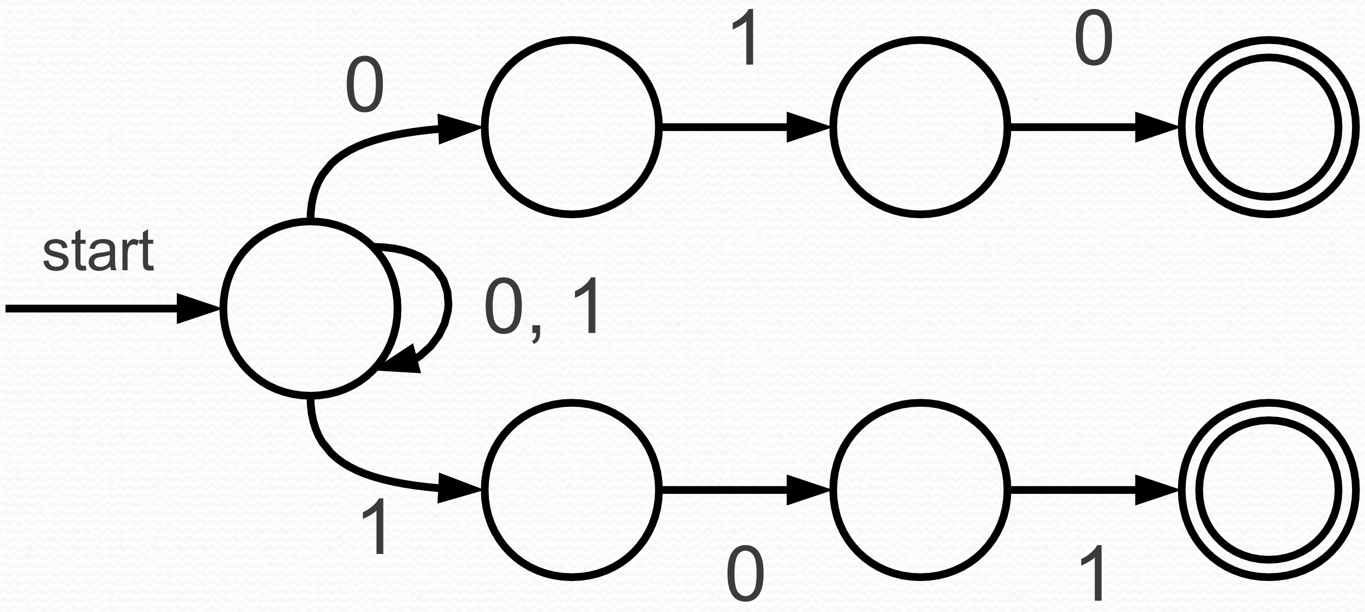


Un Autómata simple

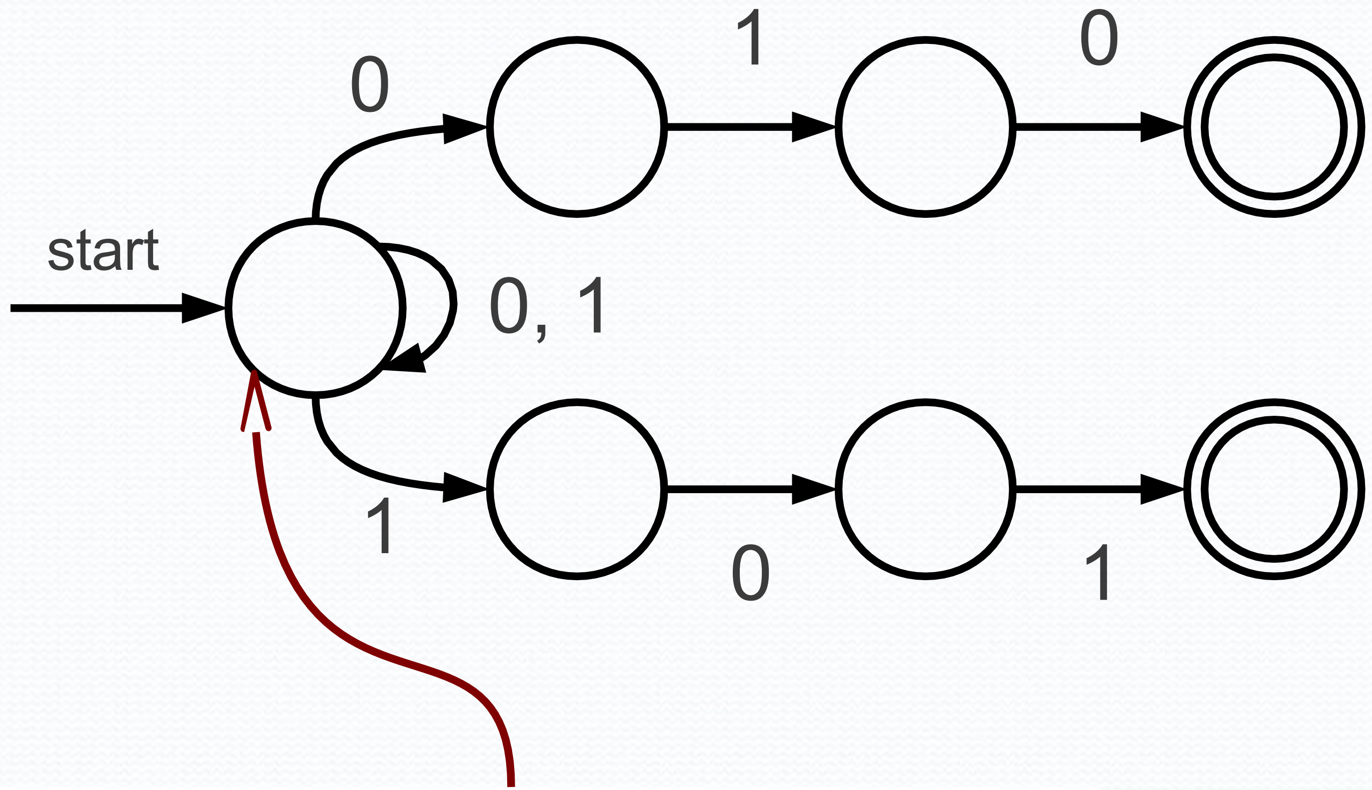


Este no es un estado de aceptación, por lo que el autómata se rechaza.

Autómata mas complejo

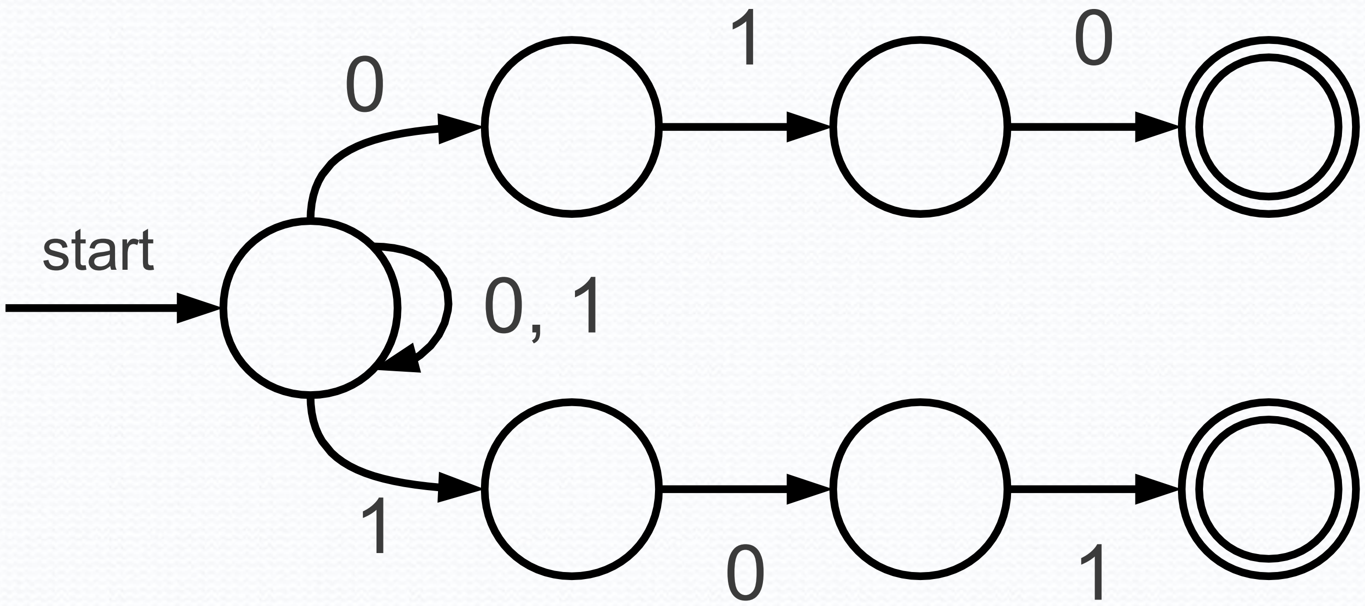


Autómata mas complejo

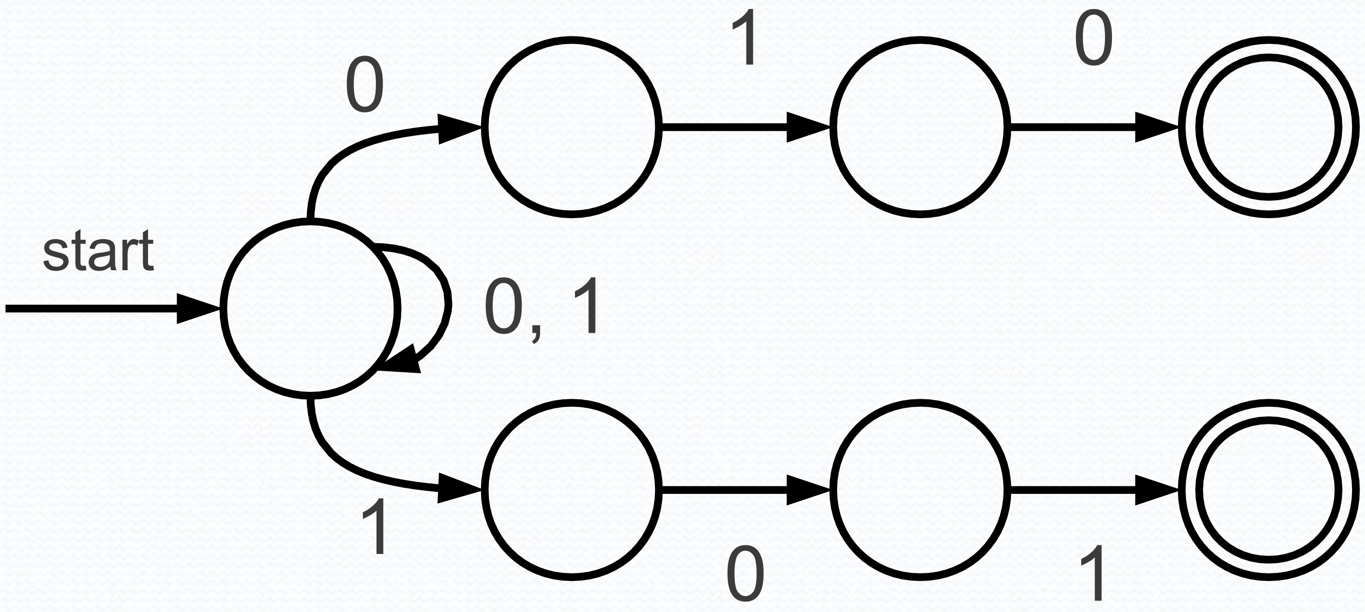


Observe que hay múltiples transiciones definidas aquí en 0 y 1. Si leemos un 0 o un 1 aquí, seguimos *ambas* transiciones y entramos en múltiples estados.

Autómata mas complejo

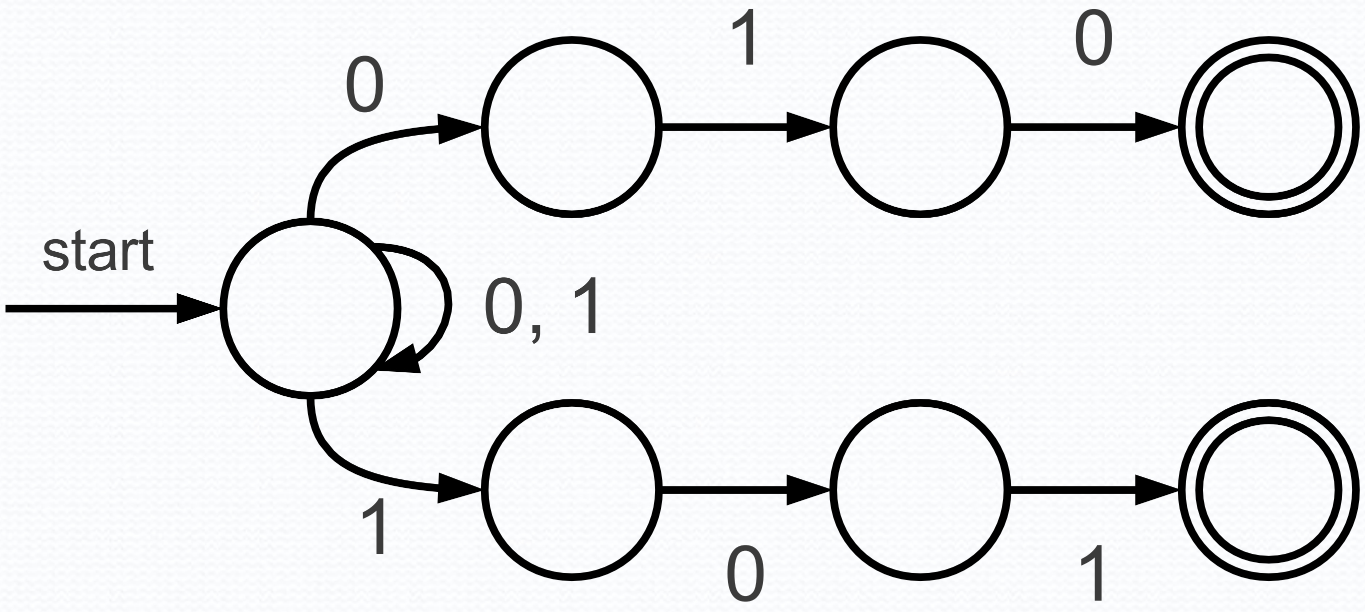


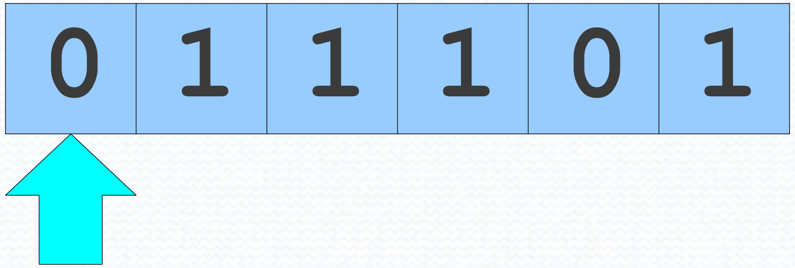
Autómata mas complejo



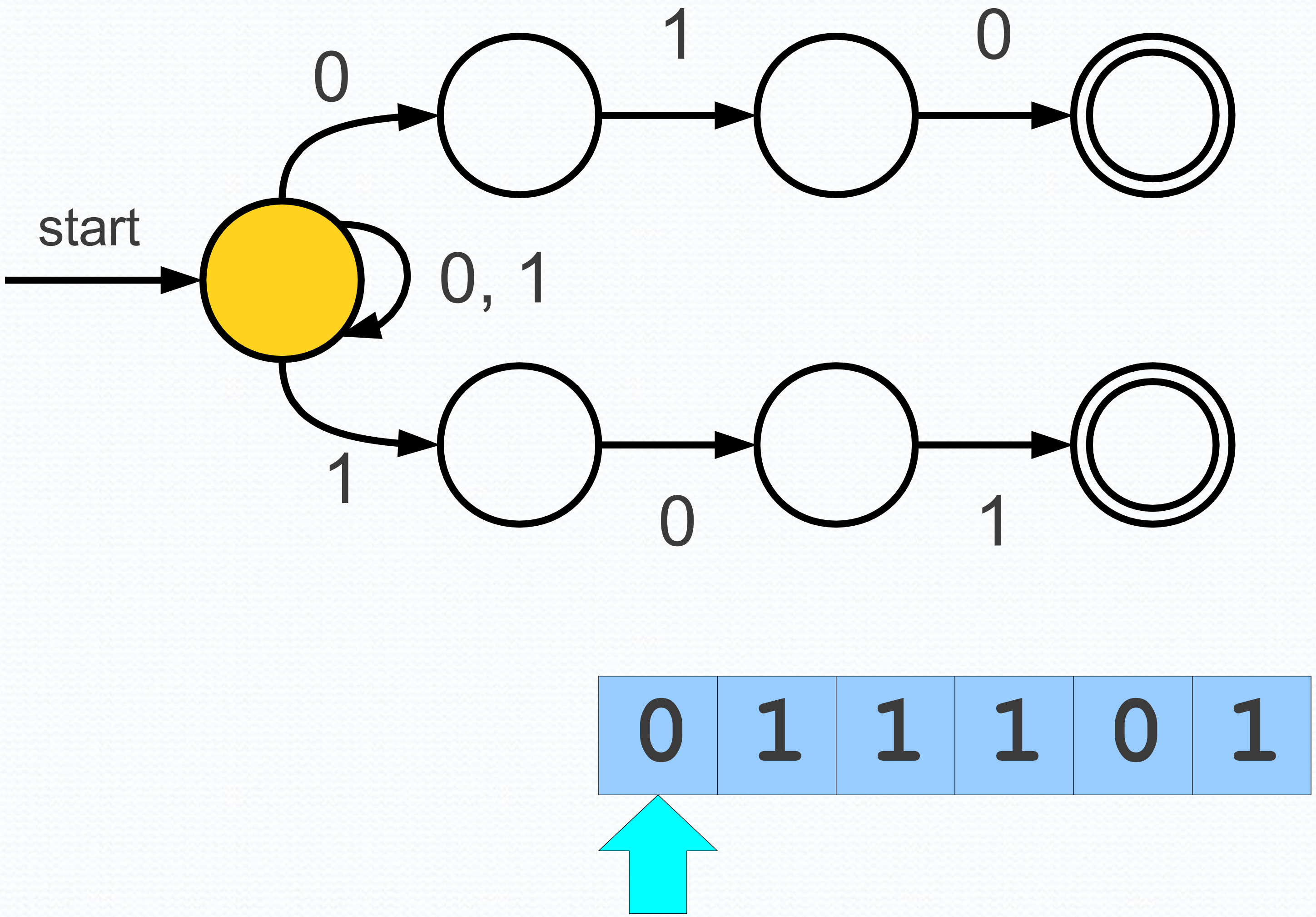
0	1	1	1	0	1

Autómata mas complejo

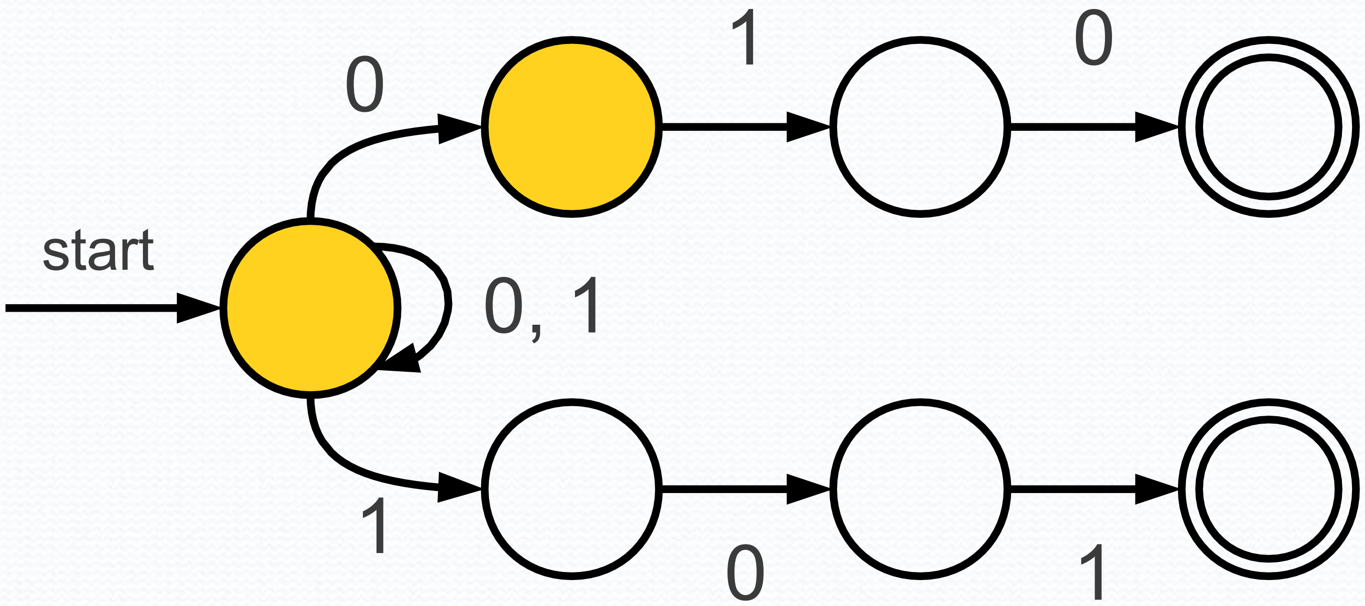




Autómata mas complejo

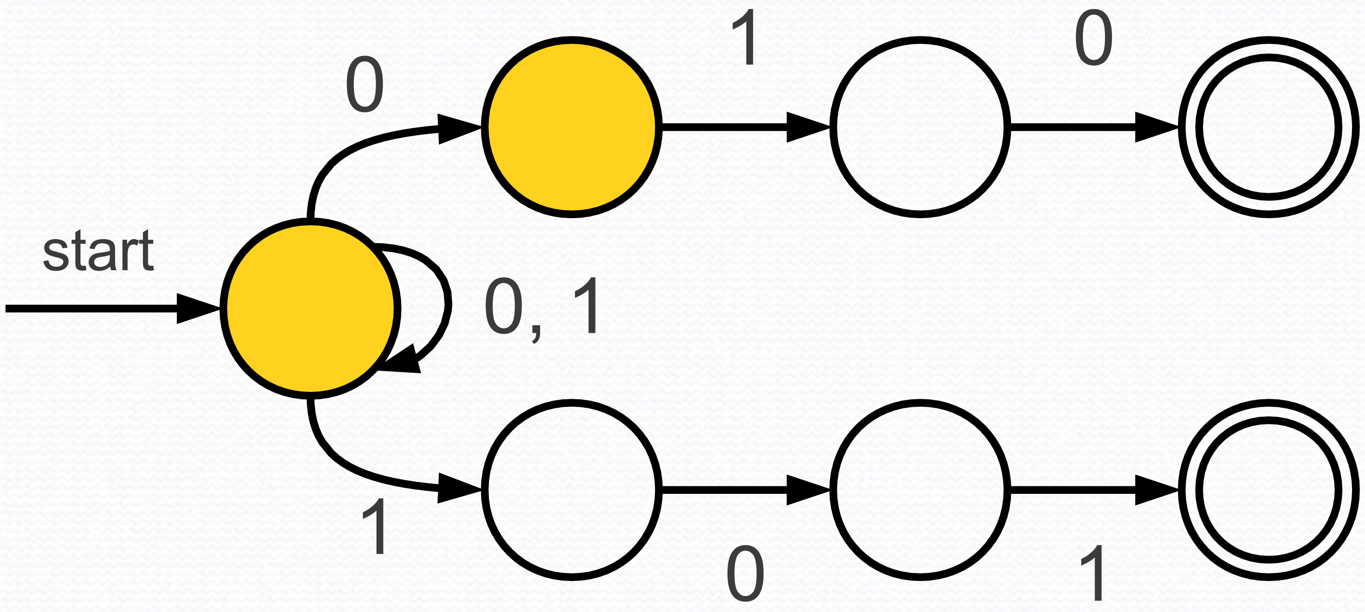


Autómata mas complejo



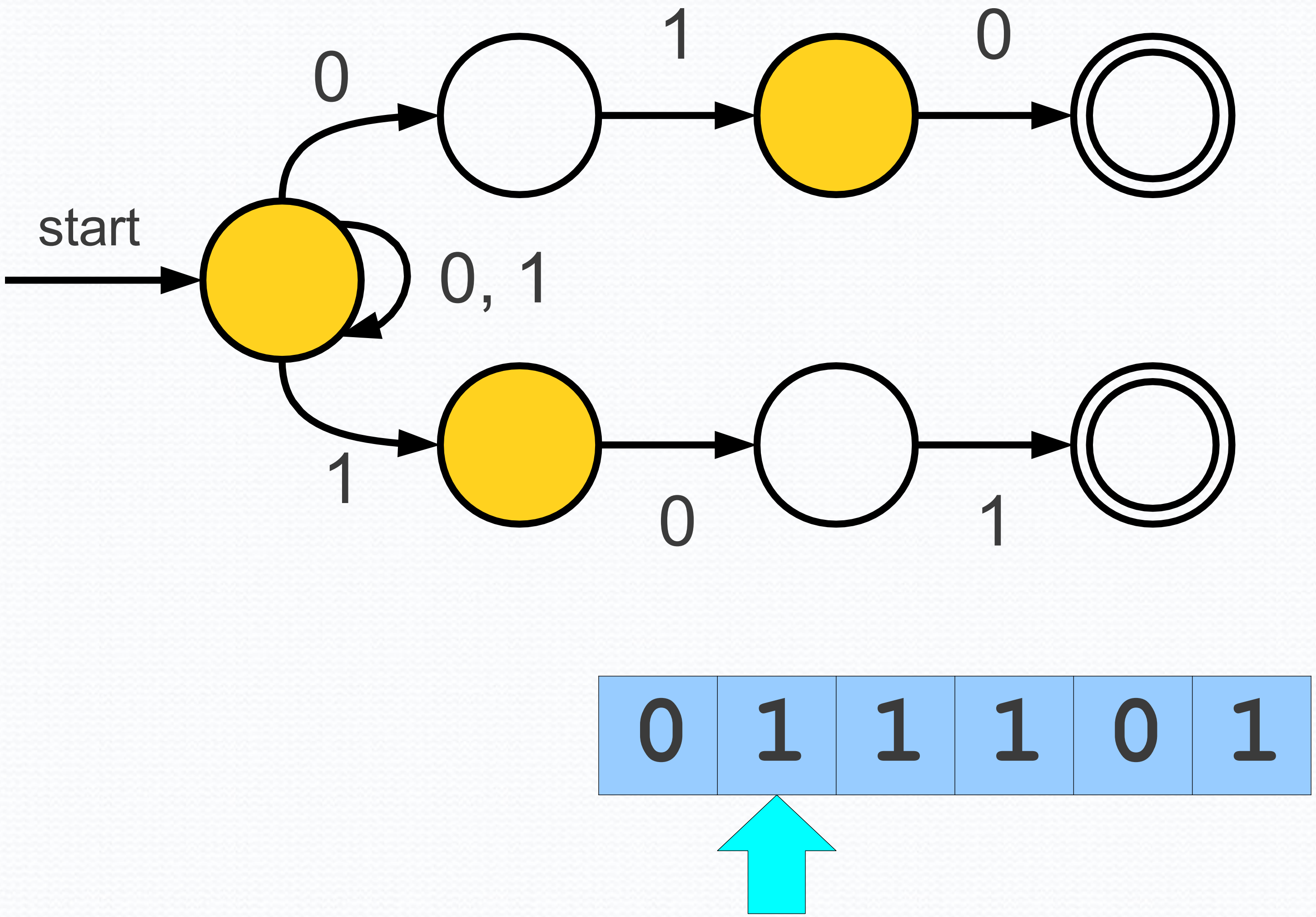
0	1	1	1	0	1

Autómata mas complejo

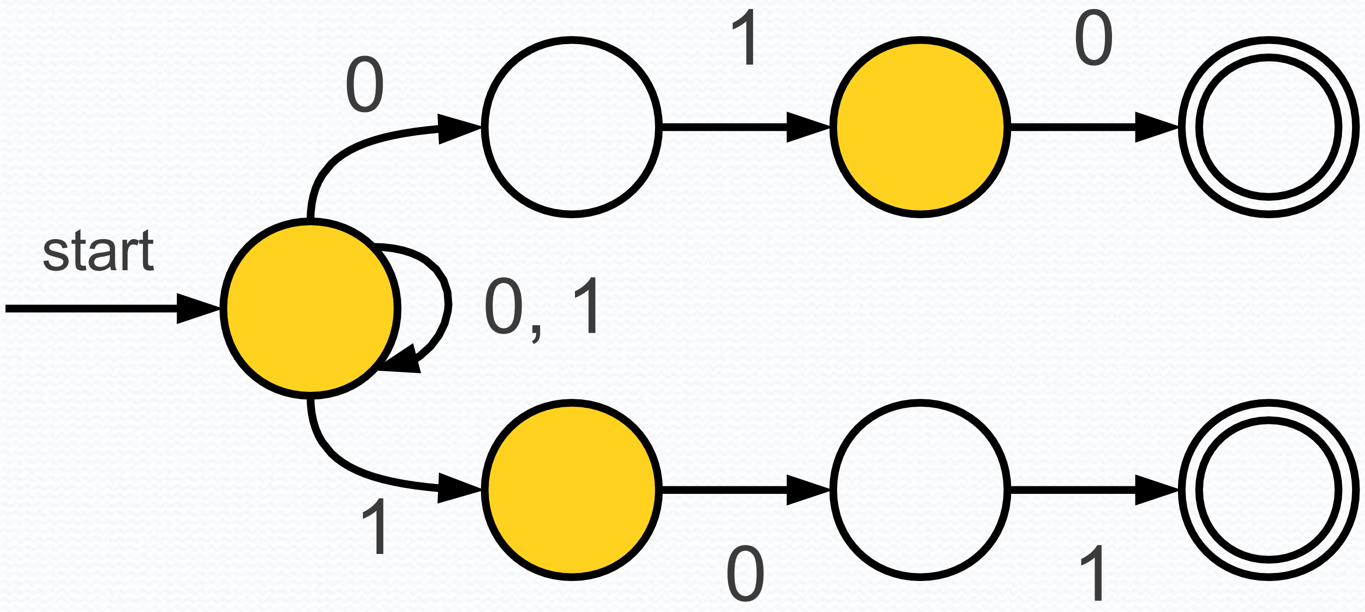


0	1	1	1	0	1

Autómata mas complejo

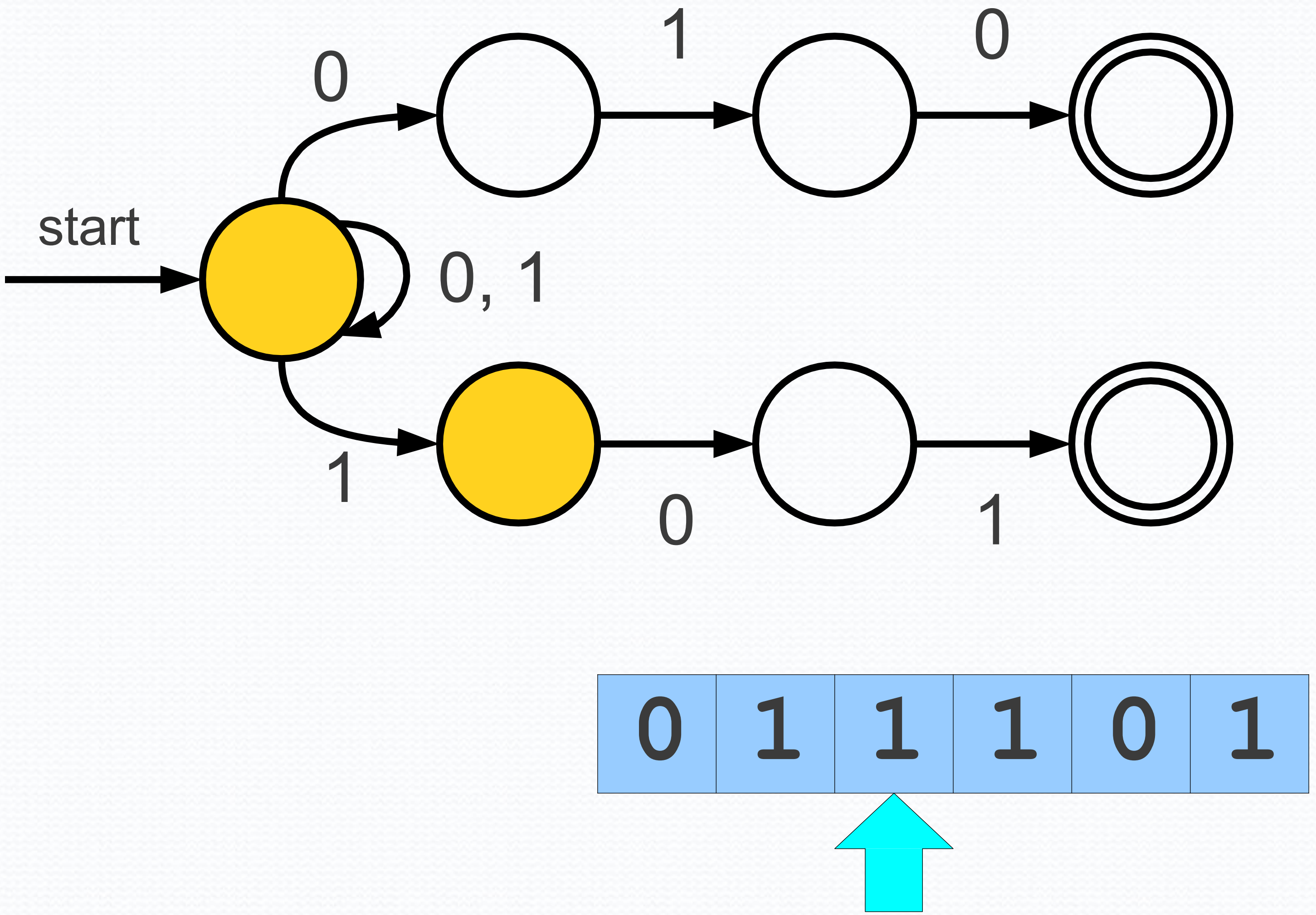


Autómata mas complejo

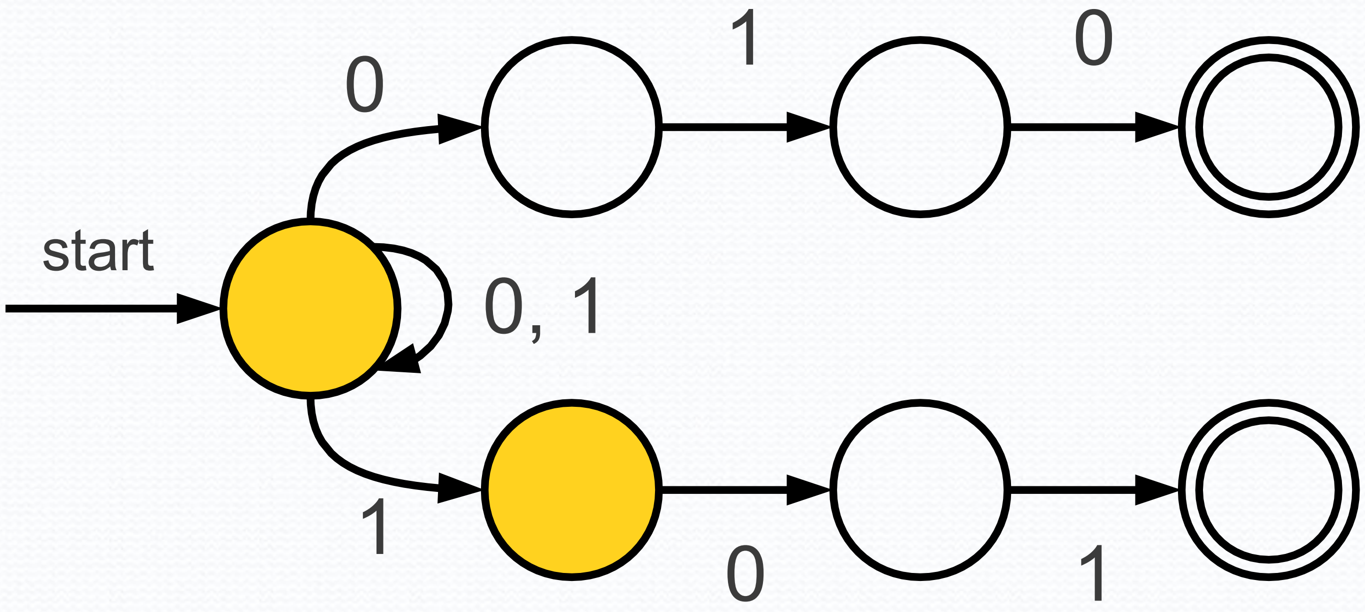


0	1	1	1	0	1

Autómata mas complejo

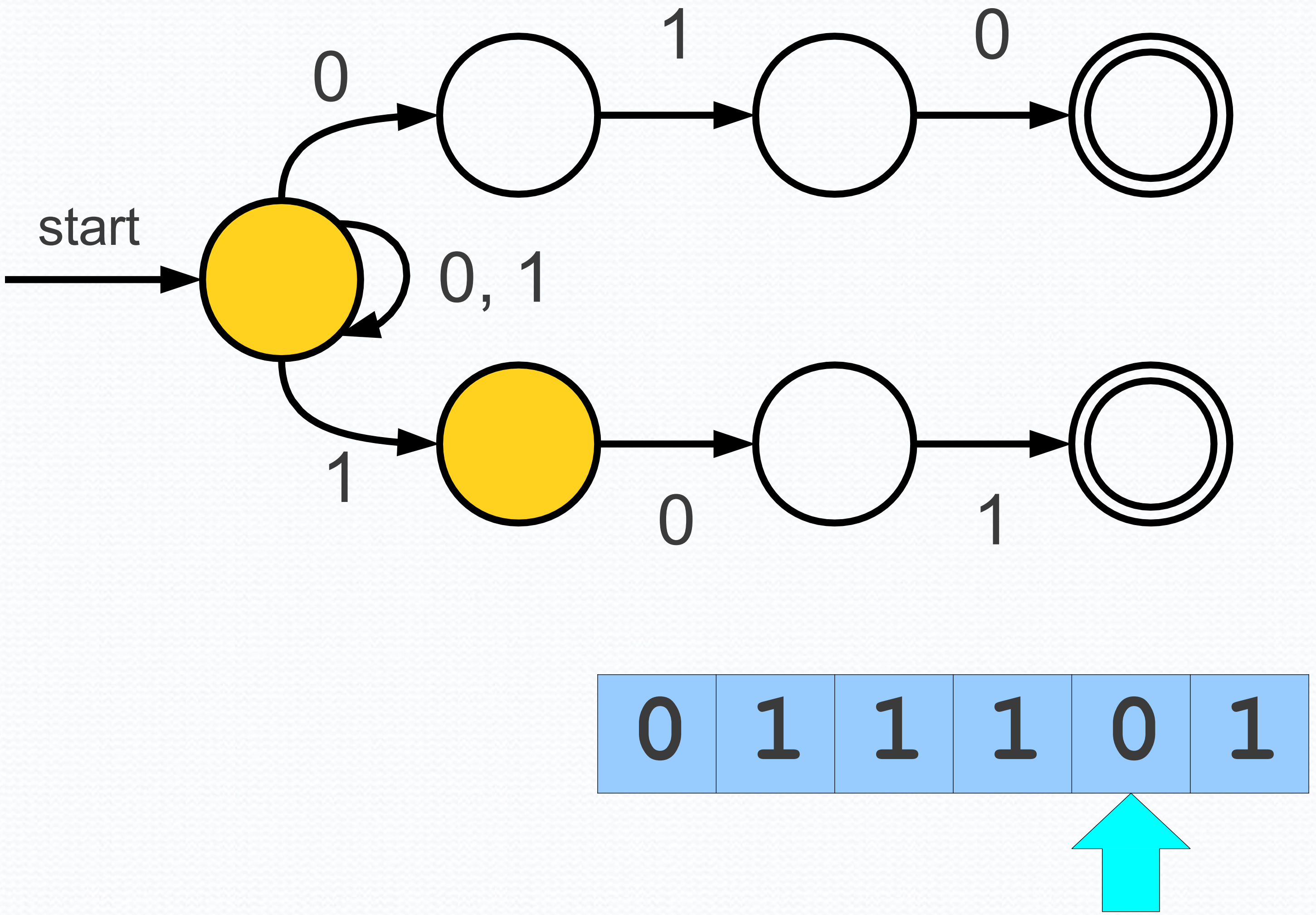


Autómata mas complejo

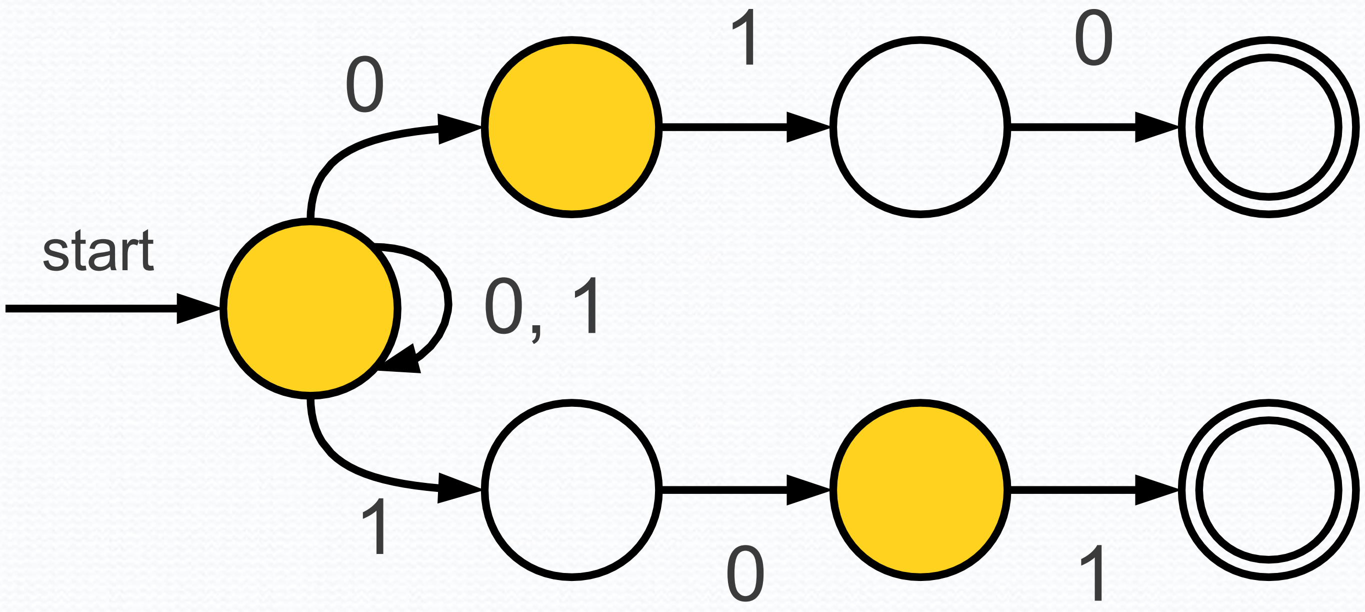


0	1	1	1	0	1

Autómata mas complejo

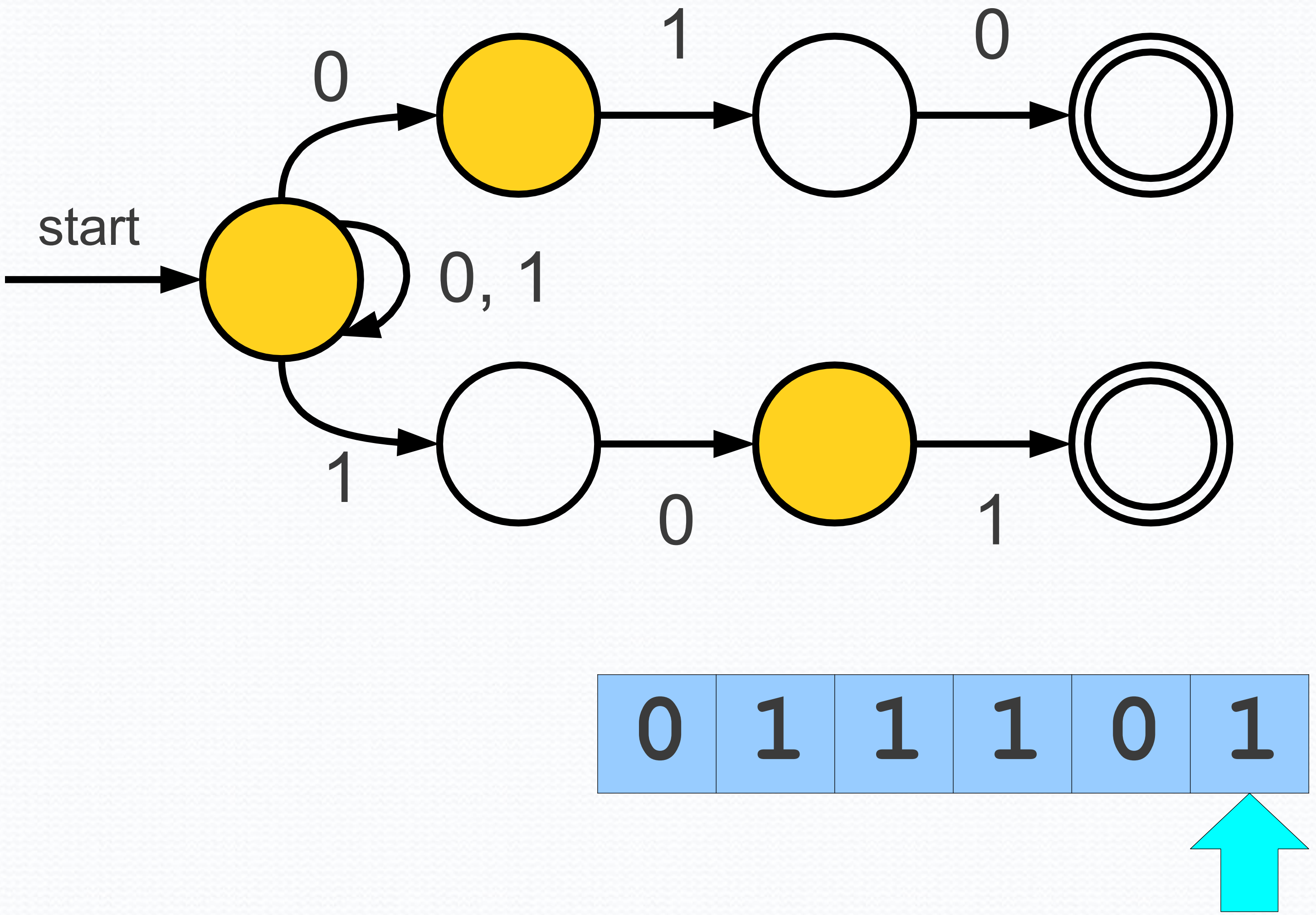


Autómata mas complejo

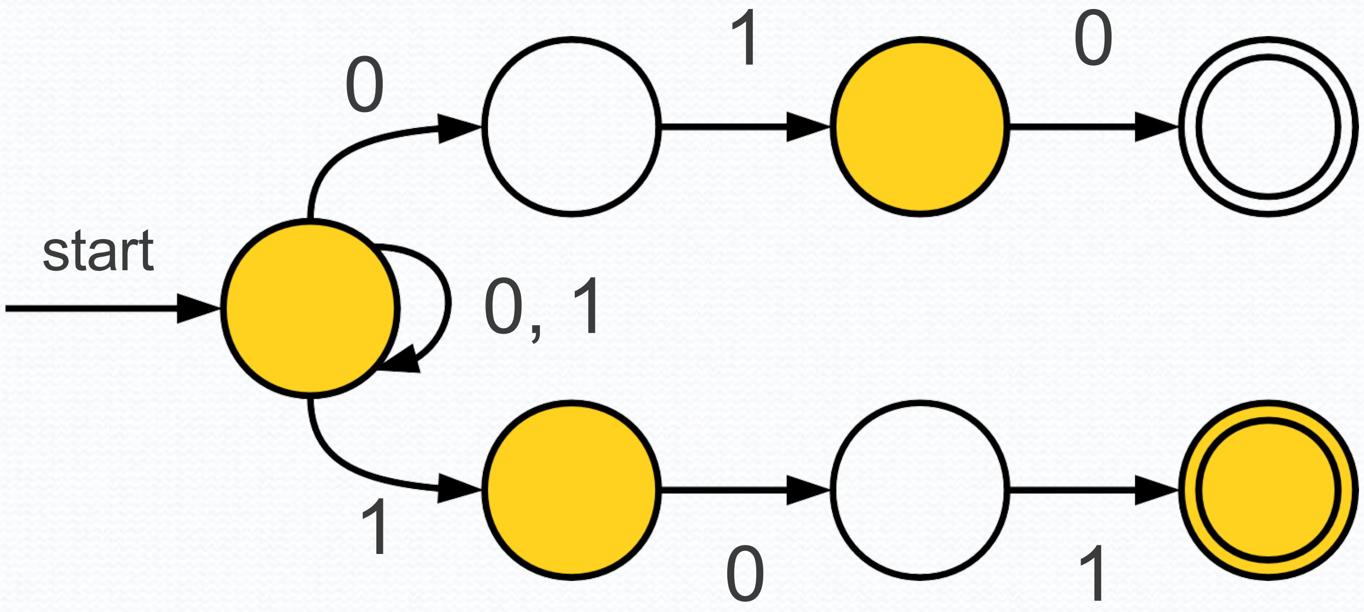


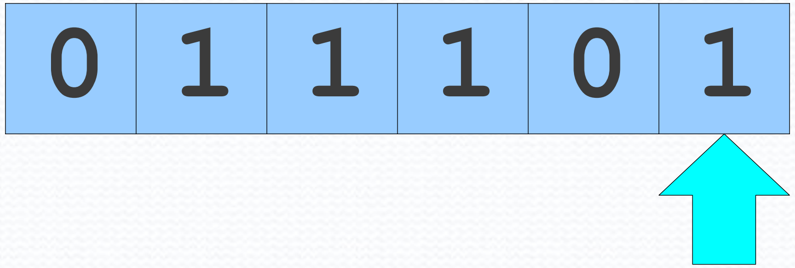
0	1	1	1	0	1

Autómata mas complejo

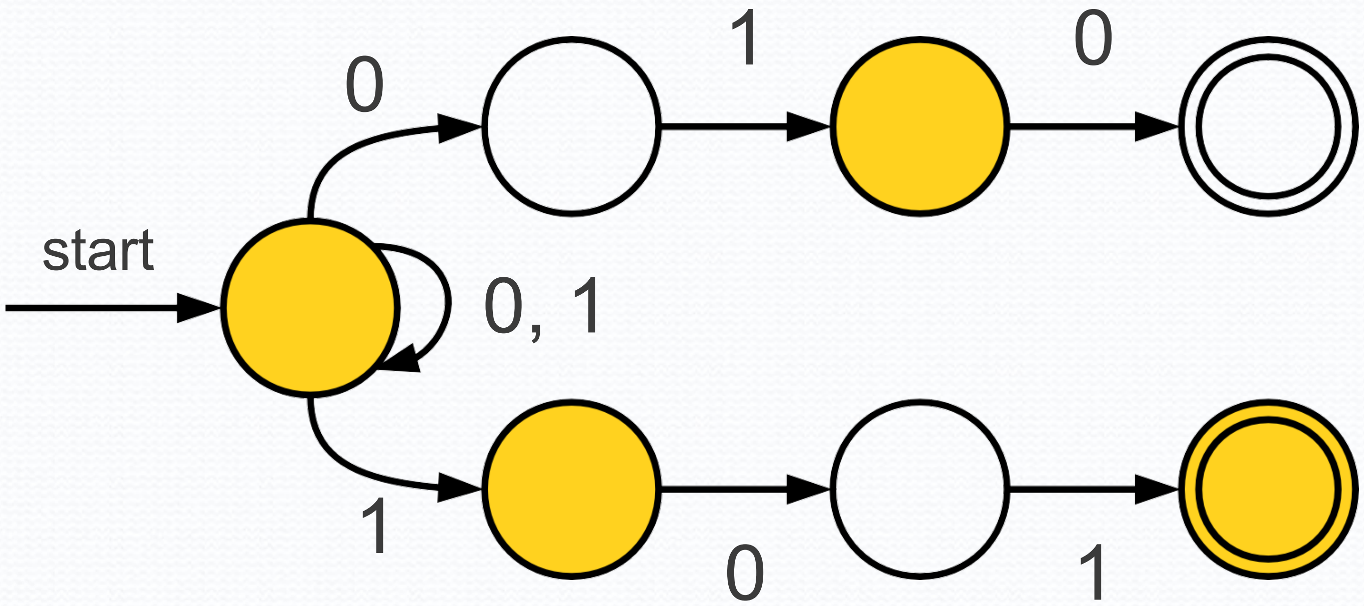


Autómata mas complejo



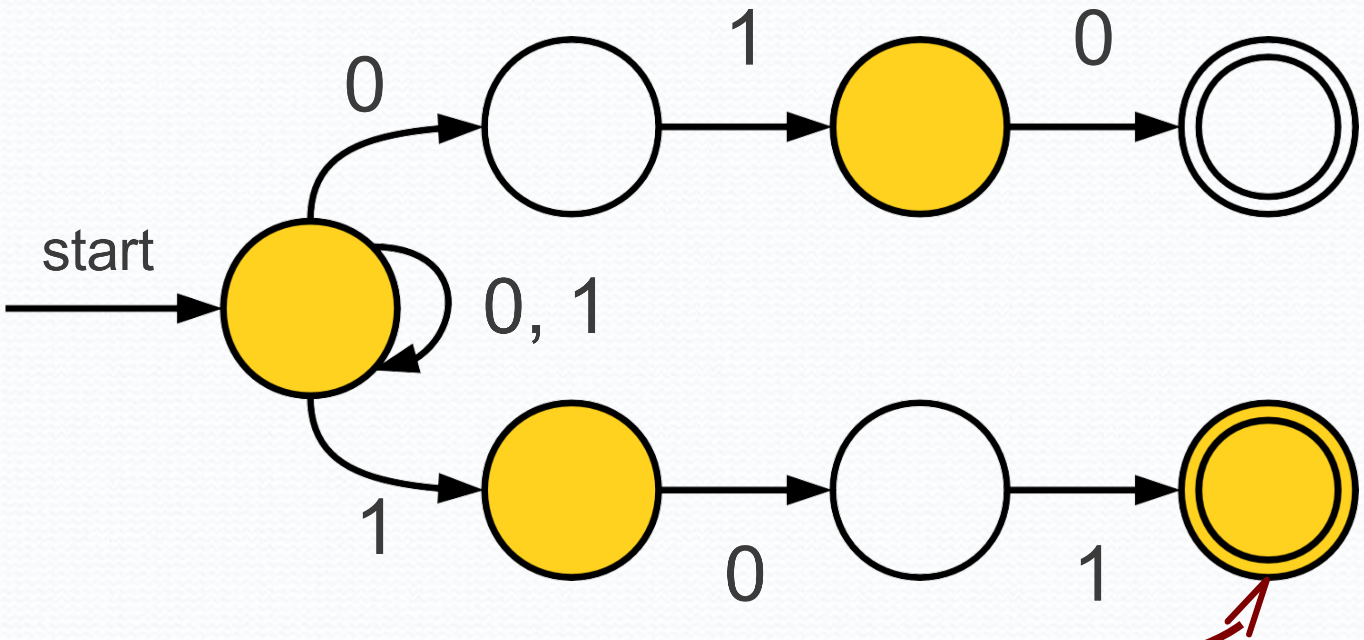


Autómata mas complejo



0	1	1	1	0	1

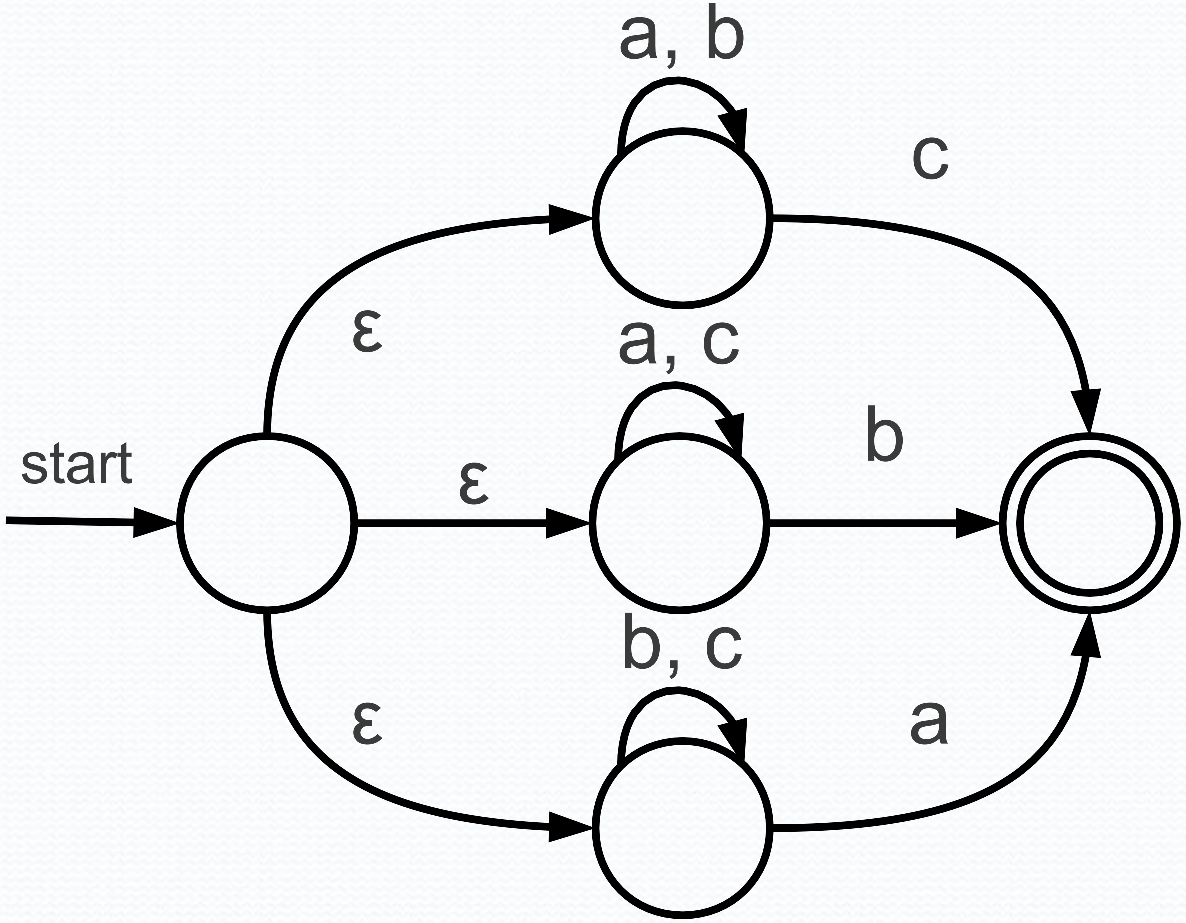
Autómata mas complejo



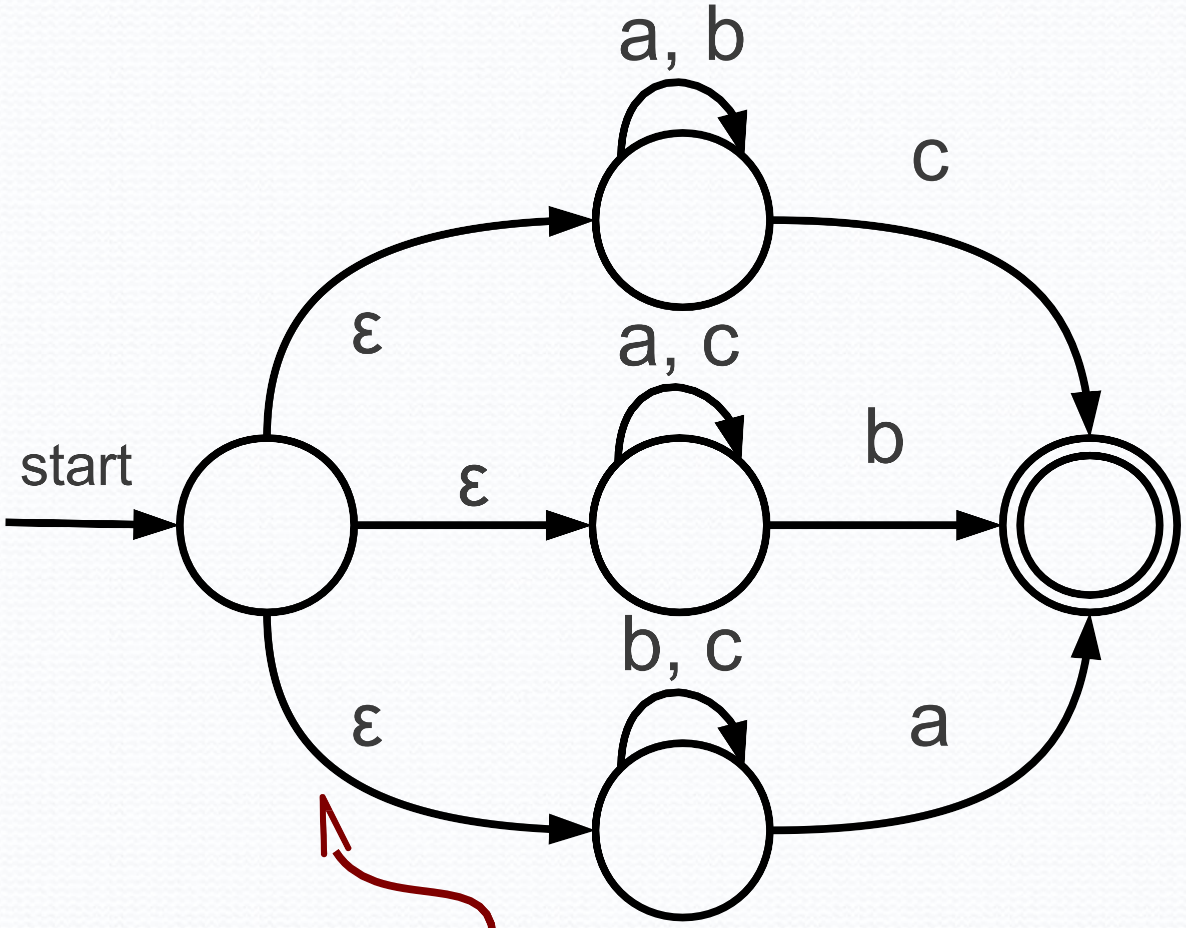
0	1	1	1	0	1

Como estamos en al menos un estado de aceptación, el autómata se acepta.

Un autómata aún más complejo

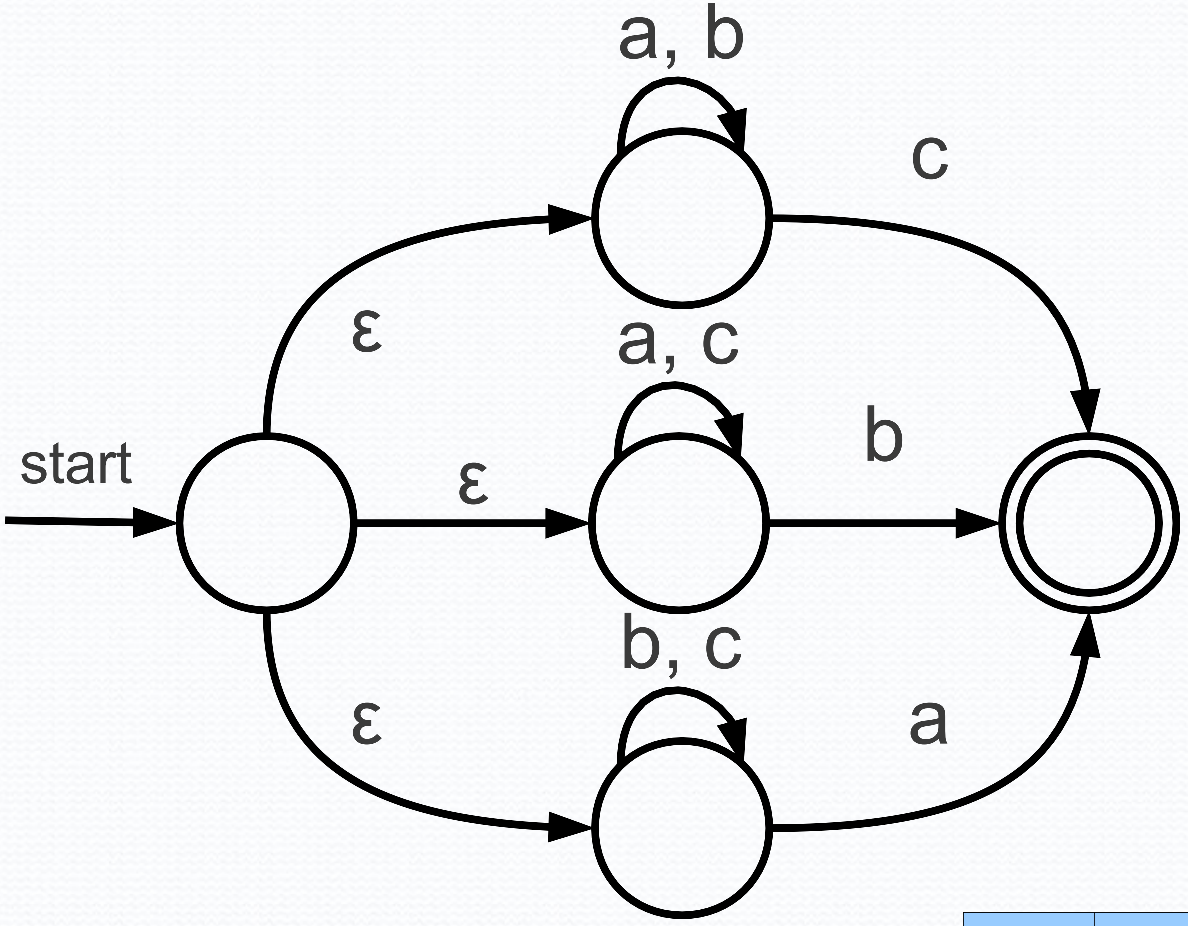


Un autómata aún más complejo



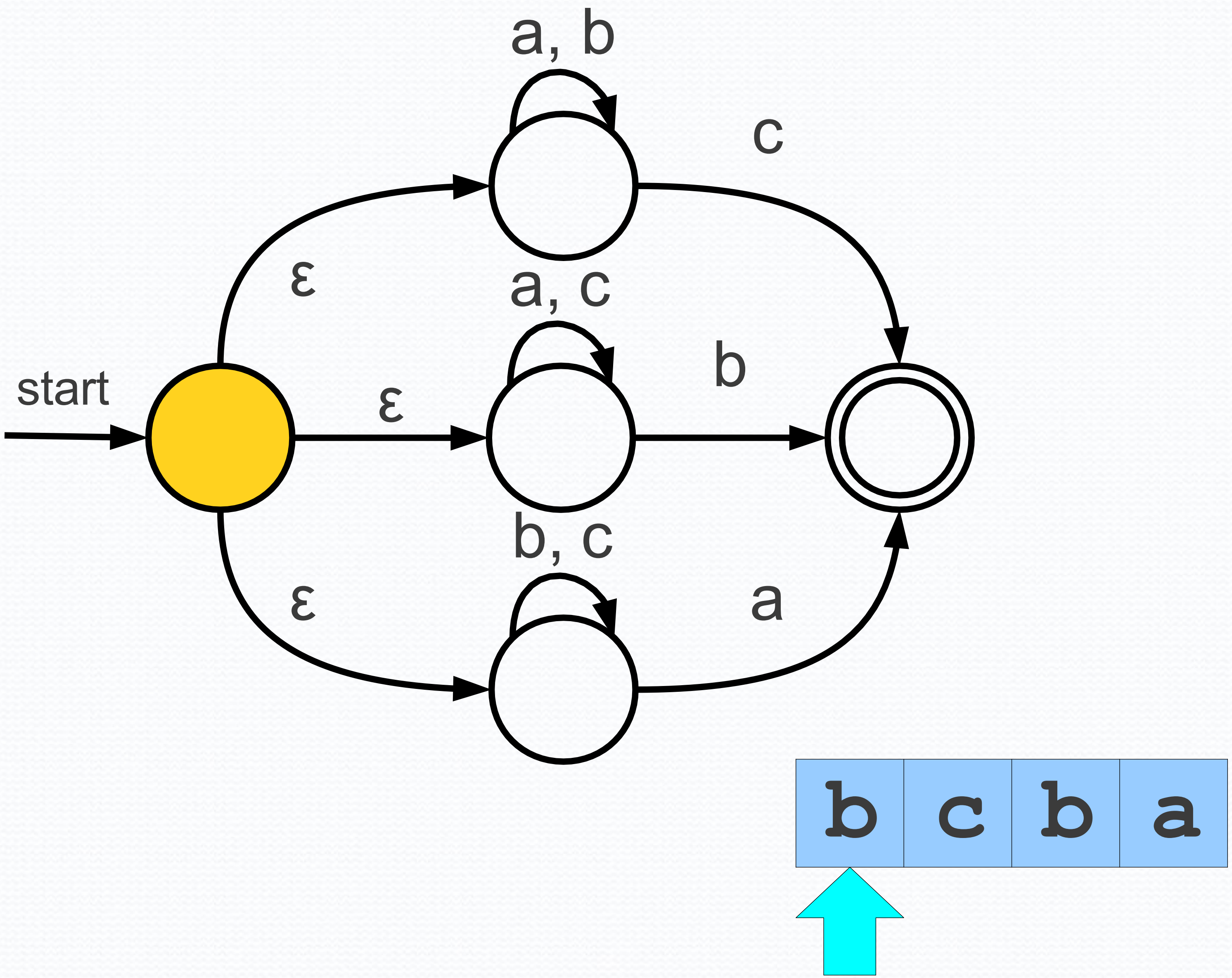
Estas son llamadas transiciones vacías (**ε-transitions**). Estas transiciones son seguidas automáticamente y sin consumir ninguna entrada

Un autómata aún más complejo

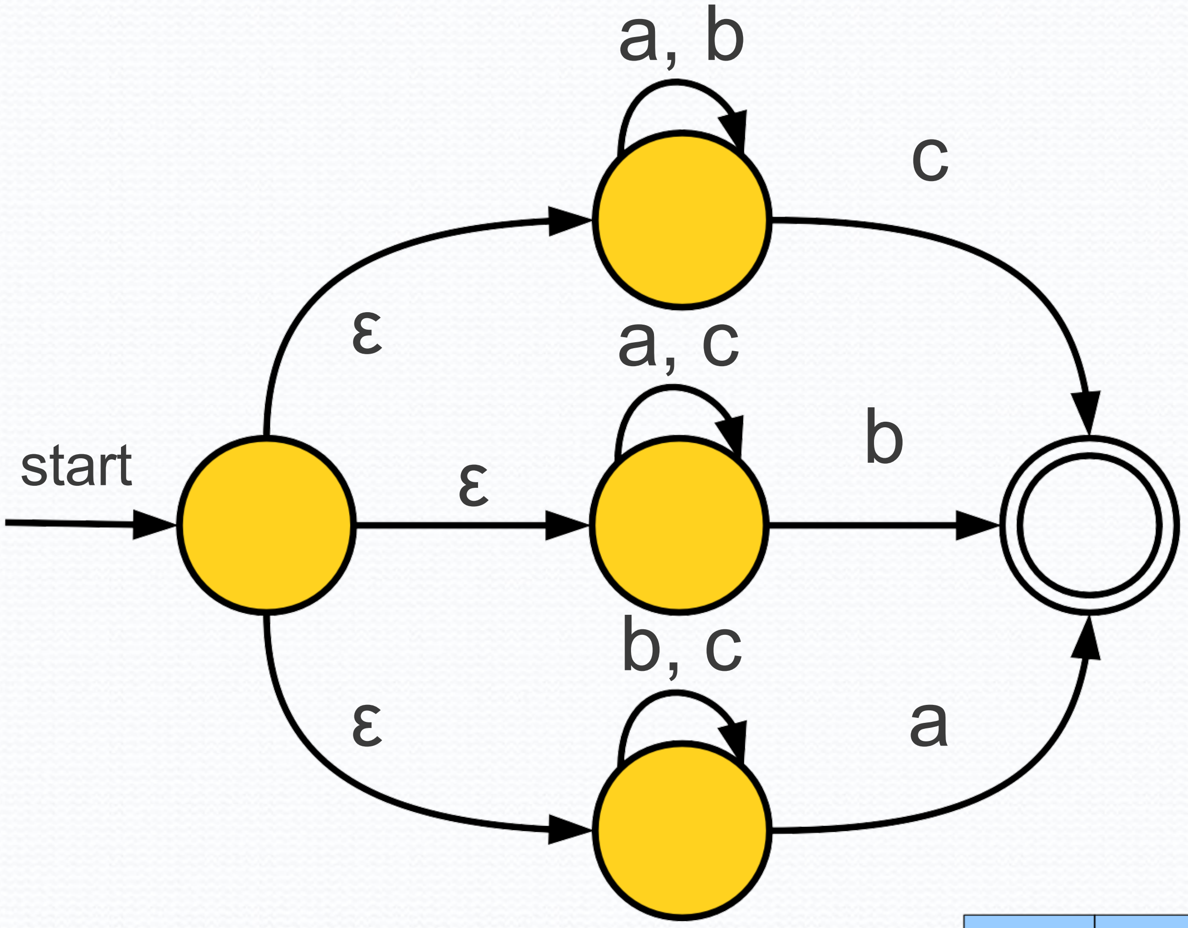


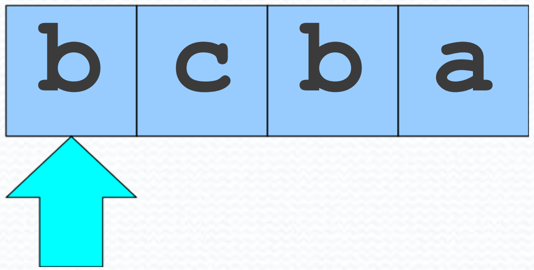
b	c	b	a

Un autómata aún más complejo

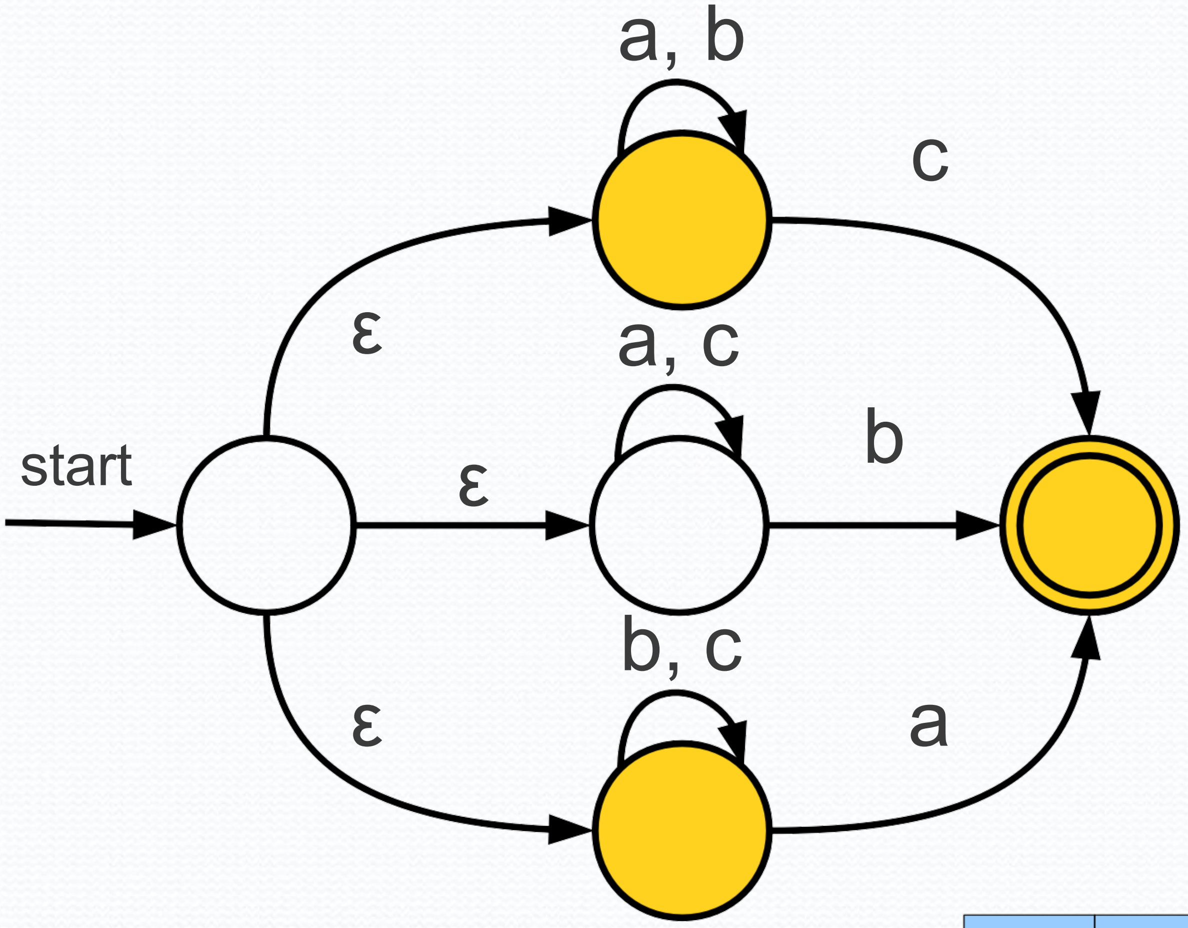


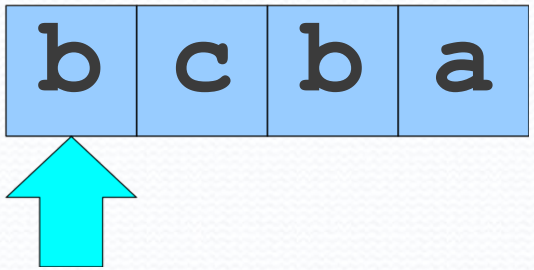
Un autómata aún más complejo



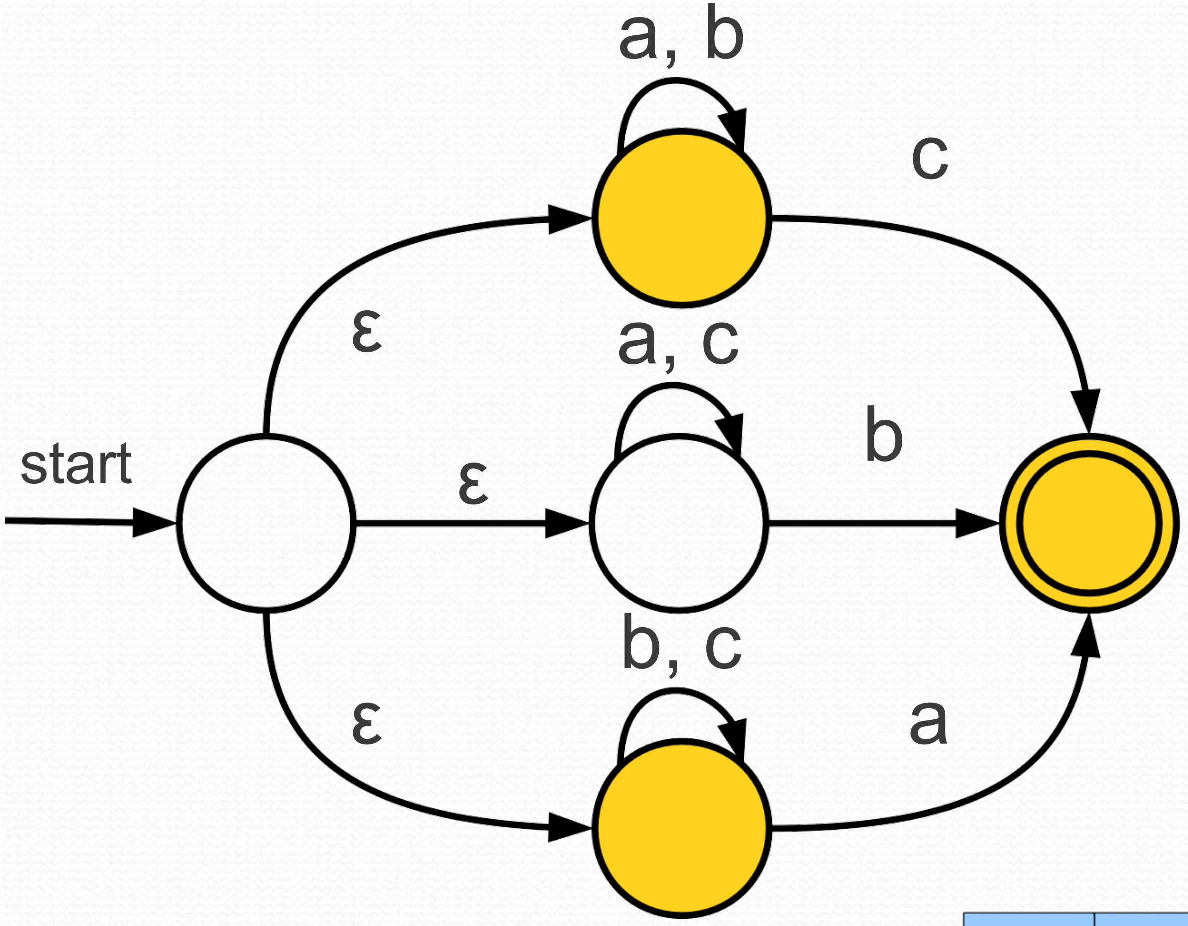


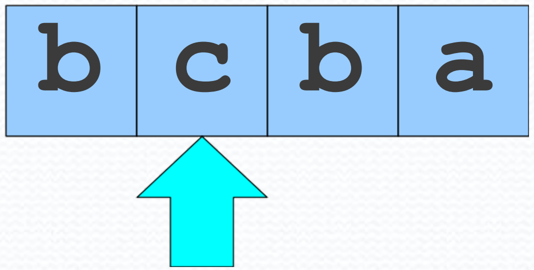
Un autómata aún más complejo



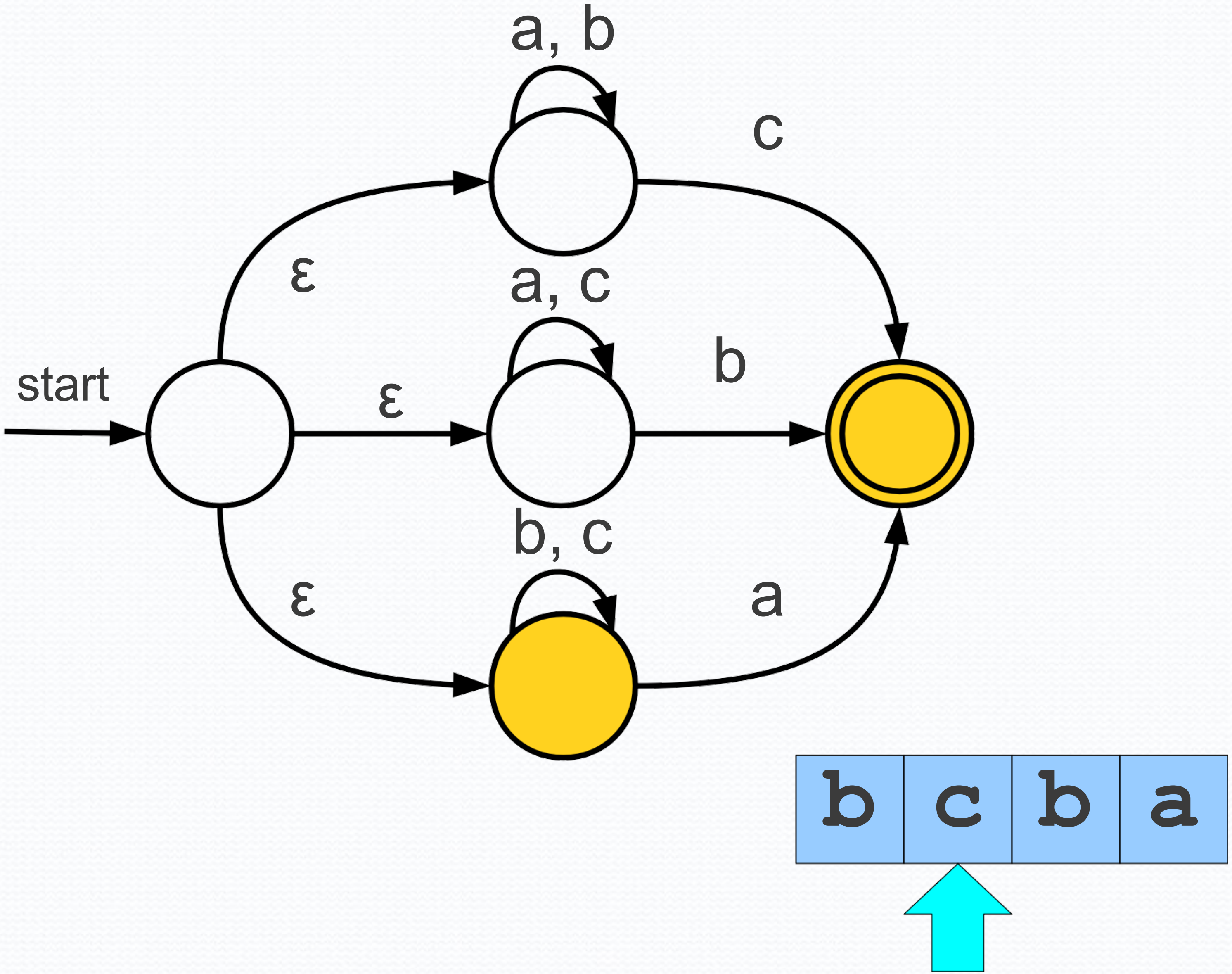


Un autómata aún más complejo

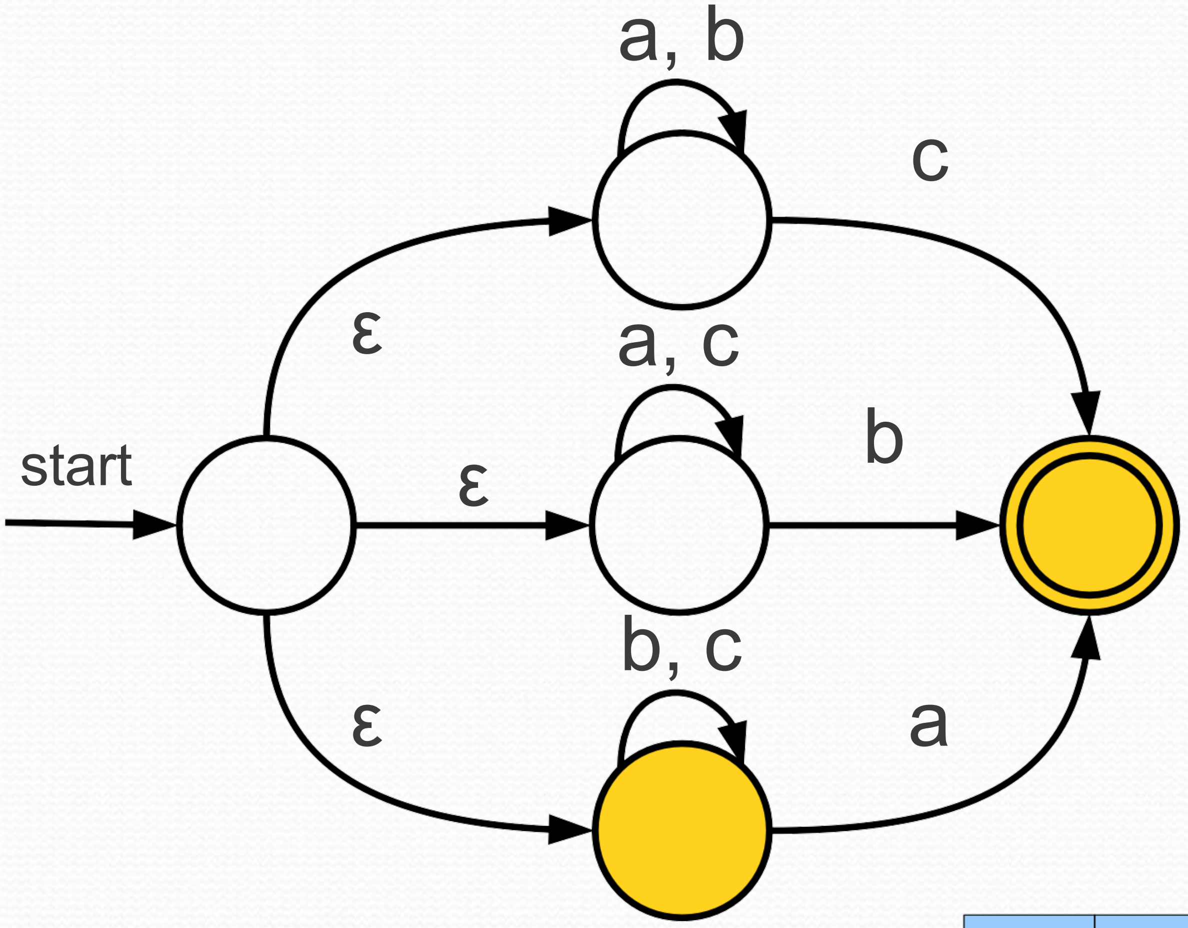


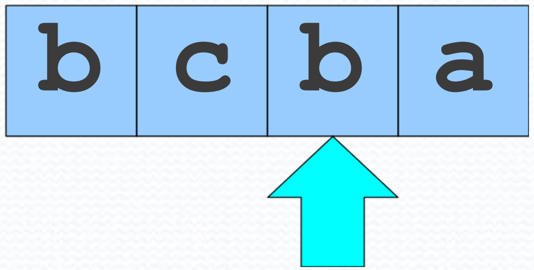


Un autómata aún más complejo

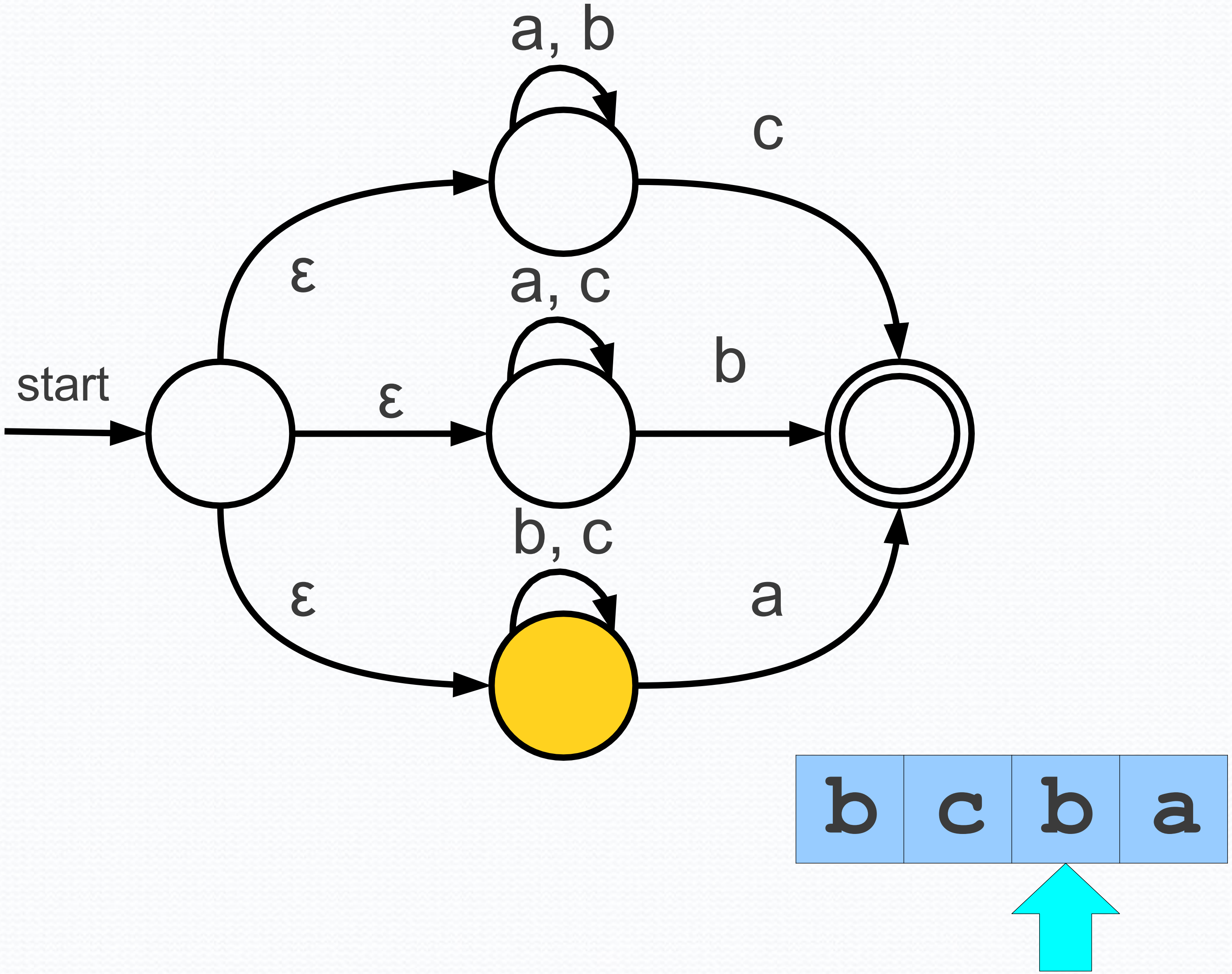


Un autómata aún más complejo

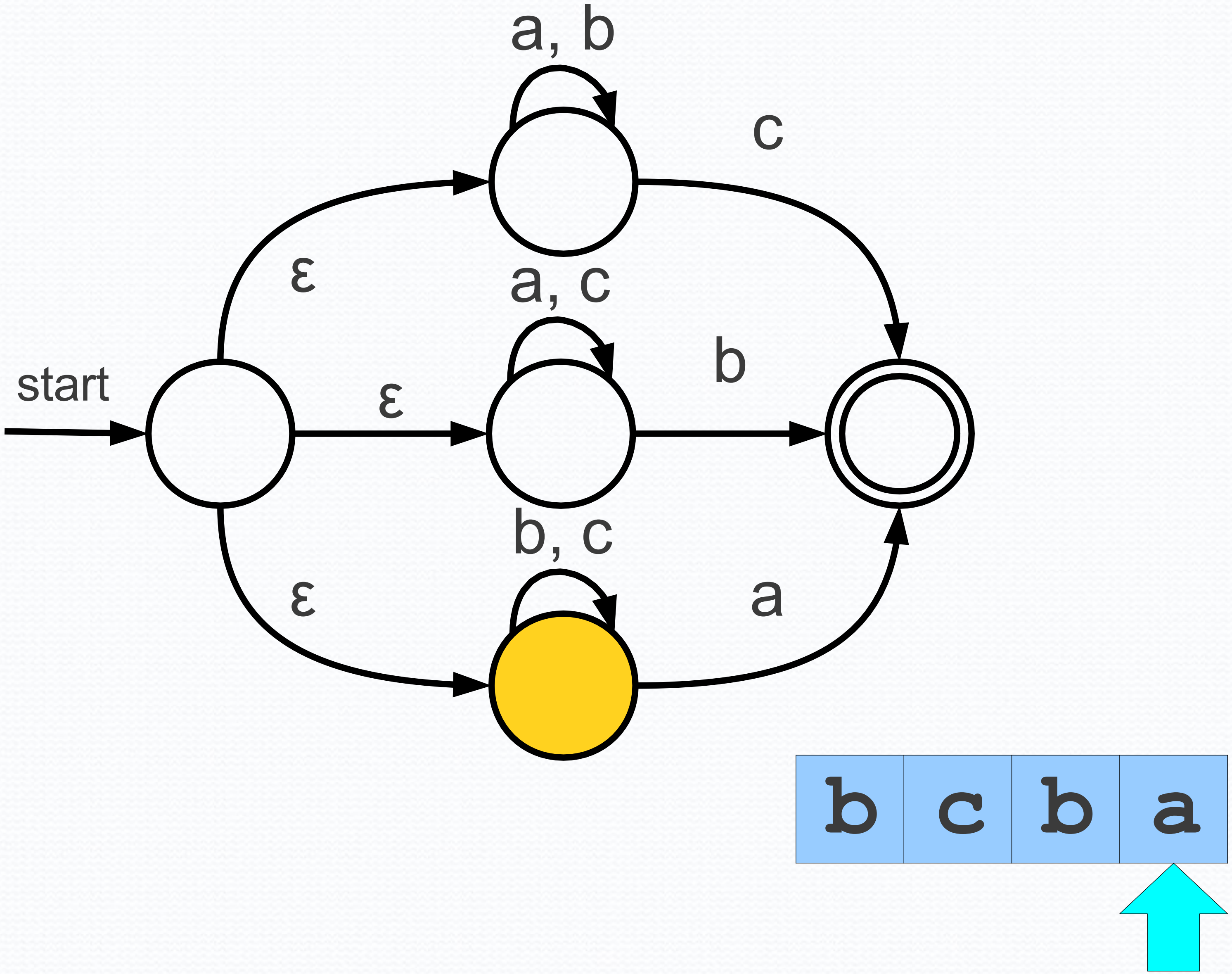




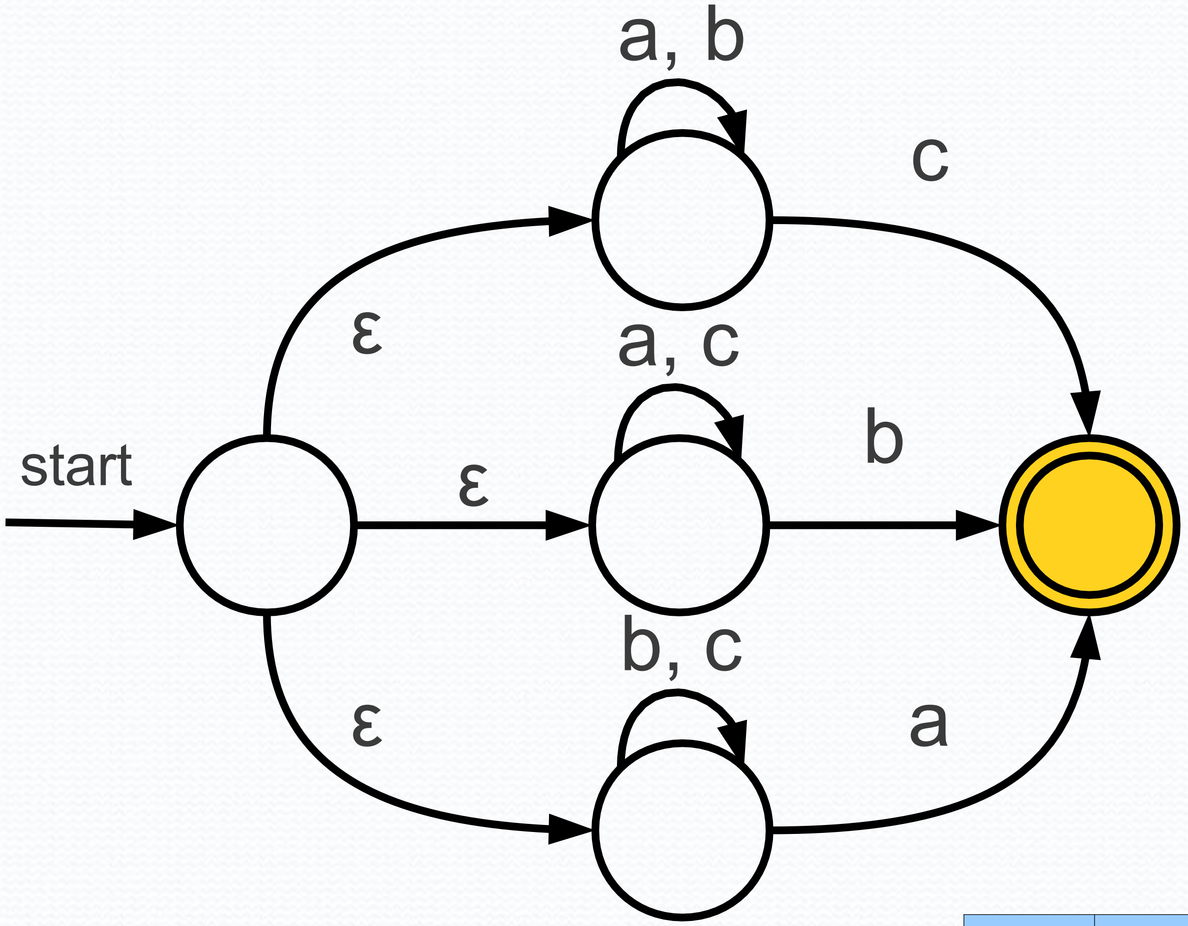
Un autómata aún más complejo

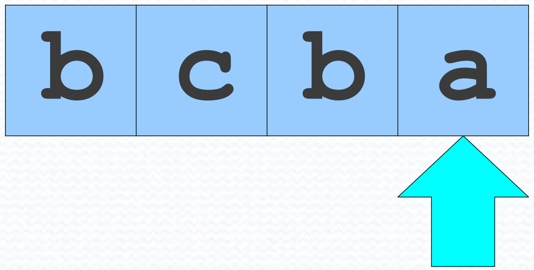


Un autómata aún más complejo

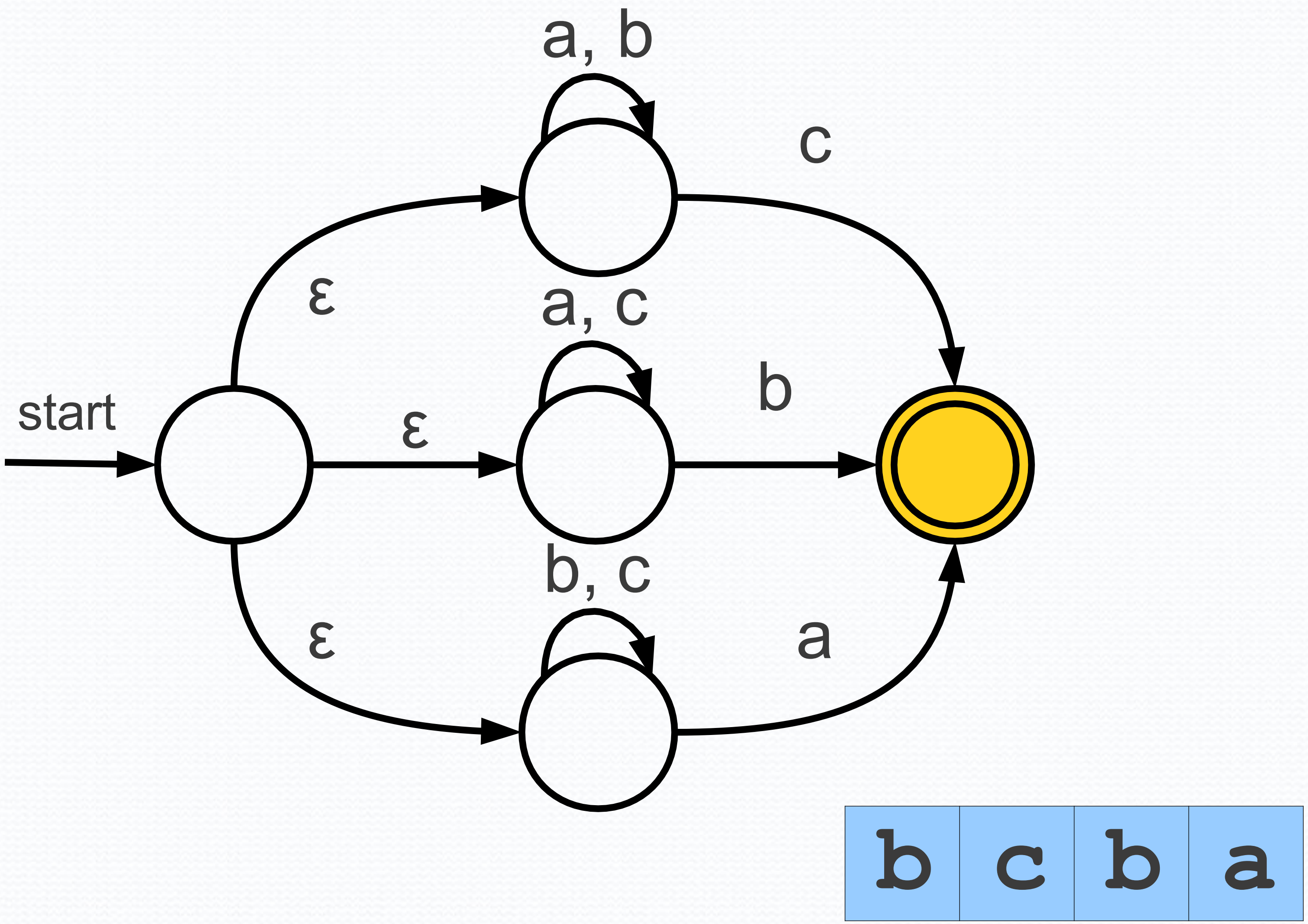


Un autómata aún más complejo

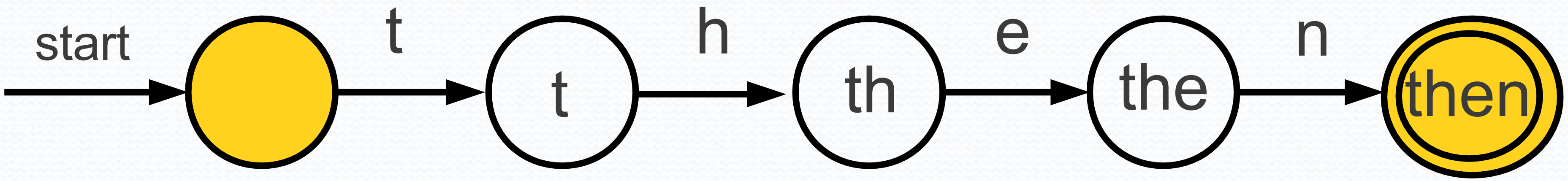




Un autómata aún más complejo



Reconociendo una keyword



Autómatas finitos

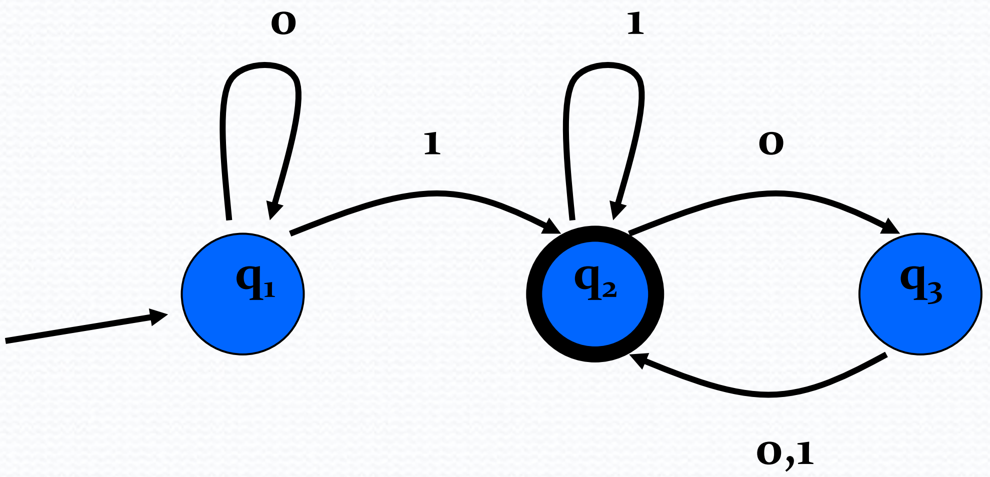
- Un autómata finito tiene un conjunto de estados y su "control" pasa de un estado a otro en respuesta a las "entradas" externas
- dicho control es "determinista",
  - el autómata no puede encontrarse en más de un estado a un mismo tiempo
  - Util para programar soluciones con un lenguaje de alto nivel
- o "no determinista",
  - sí puede estar en varios estados a la vez

Autómatas finitos deterministas

- Un autómata finito determinista (DFA) M se define por la tupla
  - $M = (Q,\Sigma,\delta,q_0,F)$

    – Q: conjunto finito de estados
    – Σ: alfabeto finito
    – δ: función de transición $\delta:Q\times\Sigma\rightarrow Q$
    – $q_0 \in Q$: estado inicial
    – $F \subseteq Q$: conjunto de estados de aceptación

$M = (Q,\Sigma,\delta,q,F)$



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

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

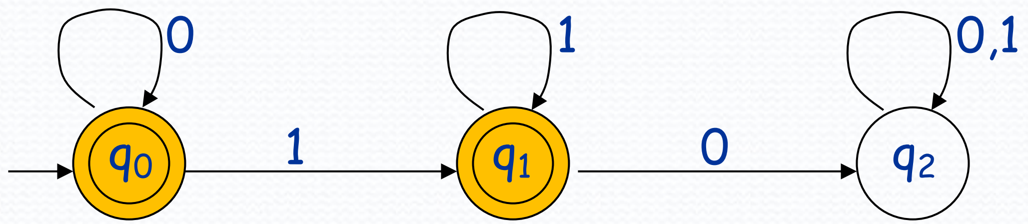
Estado inicial q_1

Estados aceptación $F=\{q_2\}$

Función transición δ:

	0	1
$\rightarrow q_1$	q_1	q_2
$* q_2$	q_3	q_2
q_3	q_2	q_2

AFD: Autómatas finitos deterministas



Alfabeto $\Sigma = \{0, 1\}$
Estados $Q = \{q_0, q_1, q_2\}$
Estado inicial q_0
Estados de aceptación $F = \{q_0, q_1\}$

Función de transición δ:

estados \ input	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_2	q_2

AFD: Configuración en un momento dado

La *configuración* del AFD en un instante dado estará dada por su estado interno y por el string que le queda por leer: (q,w), con $q \in Q$ y $w \in \Sigma^*$.

De modo que las transiciones del autómata, cuando le damos la palabra $w=w_1 \ldots w_m$ como input, serán

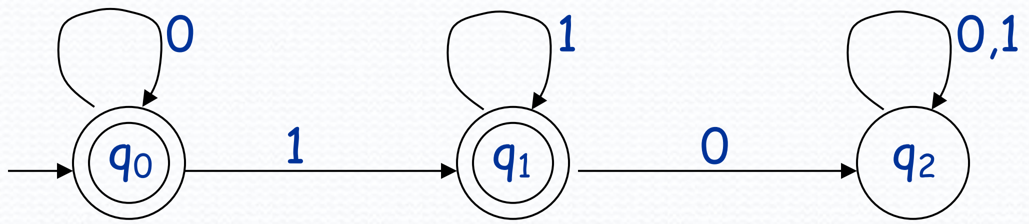
$$
\begin{aligned}
& (q_0, w_1 \ldots w_m) \\
\rightarrow\ & (\delta(q_0,w_1), w_2 \ldots w_m) \\
\rightarrow\ & \ldots \\
\rightarrow\ & (q_k, w_m) \\
\rightarrow\ & (q_r, \varepsilon)
\end{aligned}
$$

para algún q_k y q_r.

O sea: el AFD se va "comiendo" el input de a una letra, y va cambiando su estado interno según eso

AFD: Configuración en un momento dado

$$
\begin{aligned}
& (q_0, 0010011) \\
\rightarrow\ & (q_0, 010011) \\
\rightarrow\ & (q_0, 10011) \\
\rightarrow\ & (q_1, 0011) \\
\rightarrow\ & (q_2, 011) \\
\rightarrow\ & (q_2, 11) \\
\rightarrow\ & (q_2, 1) \\
\rightarrow\ & (q_2, \varepsilon)
\end{aligned}
$$



Ejercicio

- Obtenga un AFD dado el siguiente lenguaje definido en el alfabeto Σ= {0,1}. El conjunto de cadenas que contienen a la sub-cadena “01”.

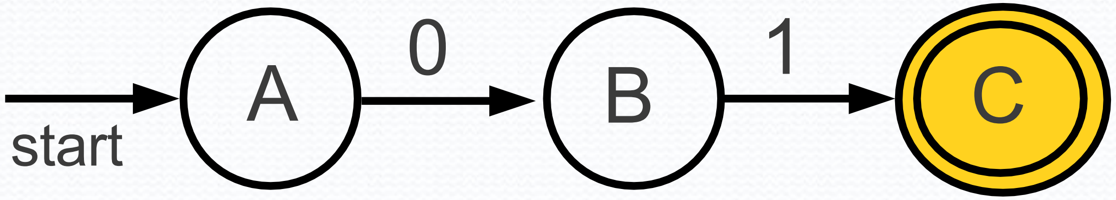
Cadenas válidas	Cadenas inválidas
01	0
001	00
00001	10
10001	1
1000001111011111	10000

http://madebyevan.com/fsm/

Ejercicio

- Obtenga un AFD dado el siguiente lenguaje definido en el alfabeto Σ= {0,1}. El conjunto de cadenas que contienen a la sub-cadena “01”.

Caso mínimo

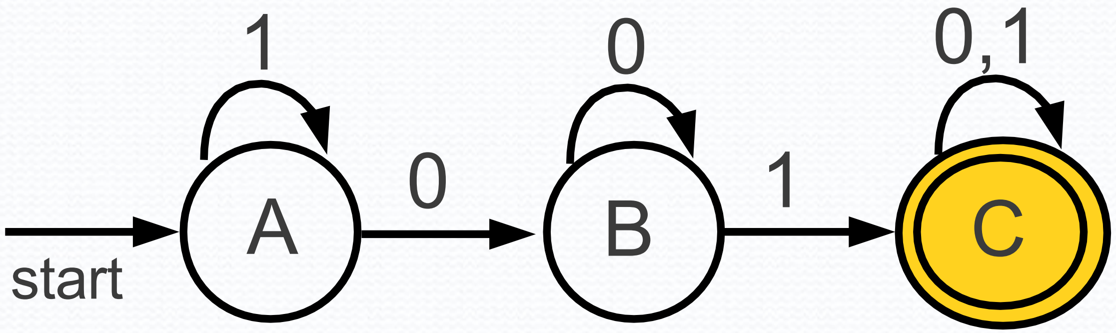


Añadir transiciones del alfabeto faltantes

http://madebyevan.com/fsm/

Ejercicio

- Obtenga un AFD dado el siguiente lenguaje definido en el alfabeto Σ= {0,1}. El conjunto de cadenas que contienen a la sub-cadena "01".

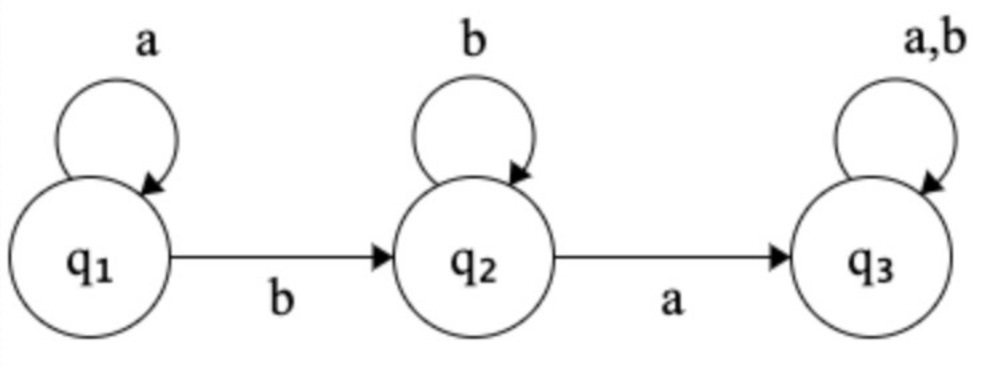


	0	**1**
>A	B	A
B	B	C
***C**	C	C

http://madebyevan.com/fsm/

Ejercicio

- Obtenga un AFD dado el siguiente lenguaje definido en el alfabeto Σ= {a,b}. El conjunto de cadenas que contienen a la sub-cadena "ba".



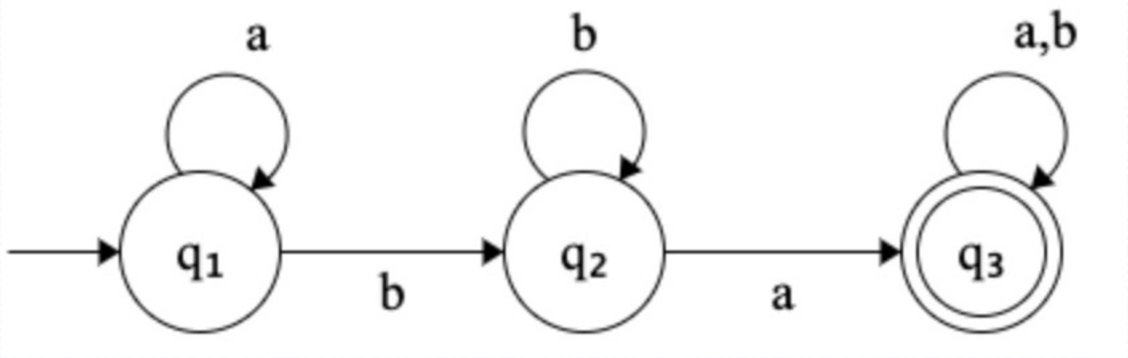
Algún problema?

Falta el estado de aceptación y estado inicial

http://madebyevan.com/fsm/

Ejercicio

- Obtenga un AFD dado el siguiente lenguaje definido en el alfabeto $\Sigma=\{a,b\}$. El conjunto de cadenas que contienen a la sub-cadena "ba".

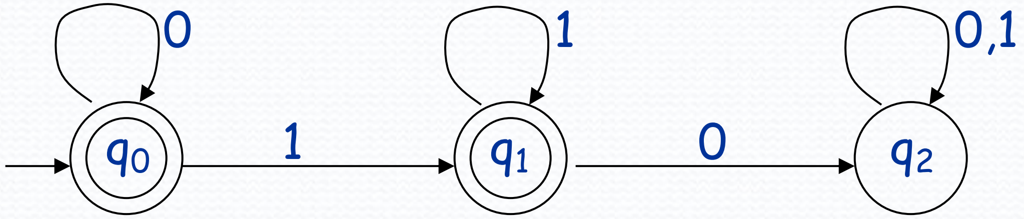


ok

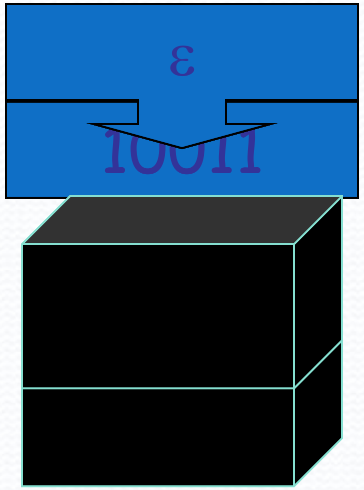
	a	**b**
>q1	q1	q2
q2	q3	q2
***q3**	q3	q3

http://madebyevan.com/fsm/

AFD: Autómatas finitos deterministas



OJO, no confundirse: los nodos del grafo no son “partes” del autómata; son sus posibles estados.



AFD: extensión de δ

Construyamos δ' de la siguiente manera (recursiva):

- $\delta'(q,\varepsilon)=q$
- $\delta'(q,w\alpha)= \delta(\delta'(q,w), \alpha)$

$\rightarrow$ La función δ' toma un estado y una palabra, y me dice a qué estado voy a llegar una vez que haya procesado con δ todas las letras de la palabra.

AFD: extensión de δ

- $\delta'(q,\varepsilon)=q$
- $\delta'(q,w\alpha)= \delta(\delta'(q,w), \alpha)$

En particular se tiene

$$\delta'(q,\alpha) = \delta(\delta'(q,\varepsilon), \alpha) = \delta(q, \alpha)$$

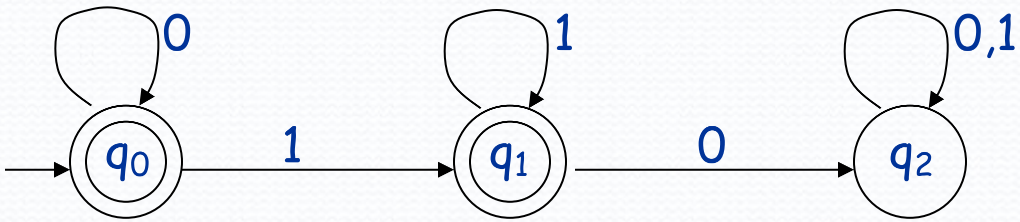
de modo que δ' es una *extensión* de δ

→ por lo tanto no necesitamos distinguirla

→ escribiremos δ para ambas.

AFD: extensión de δ

- $\delta'(q,\varepsilon)=q$
- $\delta'(q,w\alpha)= \delta(\delta'(q,w), \alpha)$



$\delta(q_0,011) = ?$

$$\begin{aligned}
\delta(q_0,011) & \\
&= \delta(\delta(q_0,01),1) \\
&= \delta(\delta(\delta(q_0,0),1),1) \\
&= \delta(\delta(q_0,1),1) \\
&= \delta(q_1,1) \\
&= q_1
\end{aligned}$$