



SINTAXIS Y SEMÁNTICA DEL LENGUAJE.

Lenguajes Formales.

Máquinas de estado: Autómatas Finitos.

LENGUAJES FORMALES

- Todo lenguaje, natural o artificial, está definido sobre *un alfabeto*.
- Además tiene un conjunto de *reglas sintácticas y semánticas*:
 - Las reglas sintácticas establecen la forma en que se combinan los símbolos del alfabeto sobre el que se define el lenguaje, para formar palabras válidas.
 - Las reglas semánticas establecen el significado de cada palabra o sentencia válida del lenguaje.



LENGUAJES FORMALES

- Un lenguaje de programación es una notación formal para expresar un algoritmo en términos de una computadora. Y como tal cumple con las características mencionadas.
- A continuación vamos a ver herramientas para poder *verificar o validar la sintaxis y la semántica* de los lenguajes de programación (base de la construcción de compiladores).
- Comenzamos con algunas definiciones generales para establecer un vocabulario común.



ALFABETOS

- Un alfabeto Σ es un conjunto no vacío de símbolos.
Por ejemplo $\Sigma = \{a, b, c, \dots, z\}$ es el alfabeto español
- Con los símbolos de un alfabeto se pueden formar cadenas de caracteres o **palabras**, ej: mama, mx, ff, flor,..... Depende de las reglas sintácticas como se combinan para tener significado.
- Un caso particular es la **palabra vacía**, ϵ .
- La longitud de una palabra es la cantidad de letras que la componen, y se denota $|w|$.
- Se pueden concatenar dos palabras A y C y se escribe AC
Ejemplo: Si A='rompe' y B='viento' \rightarrow AC= 'rompeviento'



ALFABETOS

- Llamamos Σ^* al conjunto de **todas** las palabras que se pueden formar con los símbolos de un alfabeto Σ .
- Σ^* es infinito pero enumerable.
- Si tomamos como ejemplo $\Sigma = \{a, b\}$ entonces

$\Sigma^* = \{\epsilon, a, aa, aaa, \dots, b, bb, bbb, bbbb, \dots, ab, aab, \dots, ba, baa, \dots\}$



LENGUAJES

- Un **lenguaje L** es un conjunto de palabras y está definido sobre un alfabeto Σ .
- Como los lenguajes son conjuntos pueden ser vacíos, de un solo elemento, de muchos elementos. En realidad, de infinitos elementos.
- Ejemplos:
 - $L=\{ca\}$ es un lenguaje
 - $L=\{ca, sa\}$ es otro lenguaje
- En todo los casos el lenguaje L es un subconjunto de Σ^* .
- $\Sigma^* = \{\epsilon, ca, sa, casa, saca, sasa, casasa, sacasa, \dots\}$

OPERACIONES SOBRE LENGUAJES

- Como los lenguajes son conjuntos se les puede aplicar operaciones de unión o intersección.
- También admiten la concatenación de conjuntos, donde se concatena cada palabra del 1º conjunto con cada una de las del 2º conjunto.
- Dados $L1=\{ca, ma\}$ y $L2=\{sa, pa\}$ entonces
 $L1 \cdot L2=\{casa, capa, masa, mapa\}$

↓
concatenado con



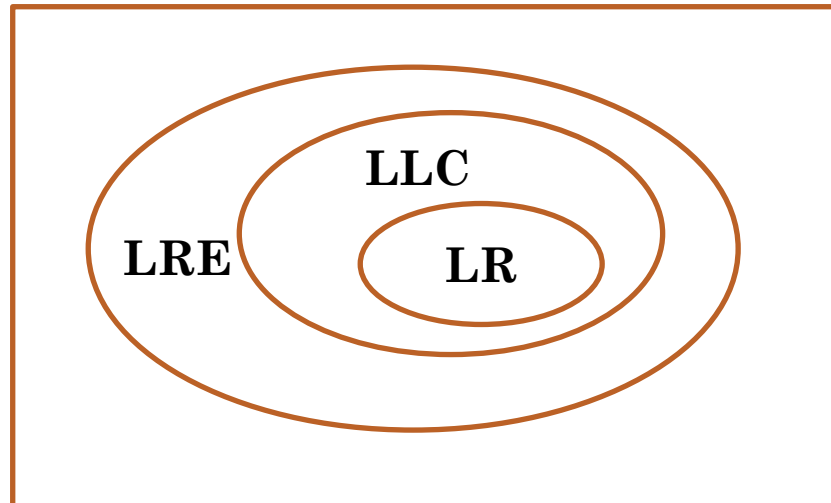
ESPECIFICACIÓN Y VALIDACIÓN DE LENGUAJES

- En función a lo visto, se puede decir que la cantidad de lenguajes que se pueden definir sobre un alfabeto es innumerable, dependiendo de las reglas de combinación de los elementos de dicho alfabeto.
- **El objetivo ahora es poder definir y validar lenguajes**, es decir especificar y verificar qué cadenas componen un lenguaje dado.
- En primer lugar vamos a hacer una clasificación de los lenguajes.



CLASIFICACIÓN DE LENGUAJES: JERARQUÍA DE CHOMSKY

- Chomsky propuso una jerarquía de lenguajes donde las clases más complejas incluyen a las más simples.
- Una clase de lenguaje abarca a aquellos lenguajes que comparten cierta característica dada.



JERARQUÍA DE CHOMSKY

- Lenguajes regulares (LR) o tipo 3: abarca a los lenguajes más simples que contienen regularidades o repeticiones en su expresión. Sus reglas de formación de palabras son las más restrictivas.

Ej. binario $L_1 = \{0, 1, 01, 11, 10, \dots\}$ o $L = \{ab, abab, ababab, \dots\}$

- Lenguajes libres de contexto (LLC) o tipo 2: abarca la mayoría de los lenguajes de programación. Sus reglas de formación de palabras permiten reemplazar símbolos sin tener en cuenta el contexto de uso de dicho símbolo.



JERARQUÍA DE CHOMSKY

- Lenguajes sensibles al contexto (LSC) o tipo 1: sus reglas de formación de palabras determinan que un símbolo se reemplaza de acuerdo al contexto de uso (depende donde aparece puede o no ser reemplazado por otro).
- Lenguajes recursivamente enumerables (LRE) o tipo 0: lenguajes con reglas poco restrictivas, lenguajes naturales basados en frases. (lengua española por ejemplo)



MÉTODOS DE ESPECIFICACIÓN Y VALIDACIÓN DE LENGUAJES

No existe ningún método que nos permita definir TODOS los lenguajes sobre un alfabeto, pero si hay formas de representar un lenguaje determinado:

- *diagramas sintácticos,*
- diferentes tipos *de máquinas de estados* (autómatas finitos, máquina de Turing)
- *gramáticas*

que permiten generar y reconocer las palabras válidas de un lenguaje.



MÁQUINA DE ESTADOS

- Una máquina de estado es:

- Un dispositivo o sistema abstracto que permite modelizar un sistema discreto de la vida real.
- Un modelado gráfico de una situación de la vida real, en el que se representan una sucesión de estados y un conjunto de eventos o estímulos, aplicados a esos estados.
- Un sistema de estímulo-respuesta. El sistema cambia de un estado a otro a partir de la recepción de un estímulo externo.



CLASIFICACIÓN DE LAS MÁQUINAS DE ESTADOS

Pueden ser:

- Reconocedoras: detecta patrones sin proveer salida alguna. Pasa desde un estado inicial a un estado final o de aceptación través de un recorrido que me asegura la validez del patrón
- Traductoras: recibe una secuencia de entrada y genera una cadena de salida.
- Se aplican a muchos sistemas de control industrial, robots, lavarropas, microondas, envasadoras, ascensores, etc..



Como ejemplo, consideremos un muy simplificado sistema de control de un ascensor:



Estados: detenido, yendo arriba, yendo abajo (cierta permanencia)

Eventos: selección_piso, arriba_nuevo_piso

Las flechas reflejan los eventos o transiciones que producen el cambio de un estado al otro. Si el ascensor está detenido, y alguien selecciona un piso al presionar el botón del ascensor, de acuerdo al piso en donde se encuentre parado y al nro del piso de destino, el ascensor cambia de estado: yendo hacia arriba o yendo hacia abajo.

Condiciones: no siempre las transiciones están condicionadas

MÁQUINAS DE ESTADO

- En particular las estudiamos por su relación con los lenguajes formales ya que *se utilizan ampliamente para desarrollar analizadores léxicos, sintácticos y semánticos, siendo así la base para la implementación de compiladores.*
- Ya veremos que los *compiladores traducen código* de alto nivel a bajo nivel, pasando por dos fases: de análisis y síntesis. En dichas fases *intervienen tanto las máquinas de estados como las gramáticas.*



AUTÓMATAS FINITOS

- Las máquinas de estados más sencillas son los autómatas finitos.
- Un *autómata finito* es un modelado gráfico de una máquina abstracta que permite representar una serie de acciones o eventos y de estados.
- El gráfico tiene forma de *grafo dirigido*.
- Los nodos son *estados*: alguna situación en la que se permanece un cierto tiempo.



AUTÓMATAS FINITOS: CARACTERÍSTICAS

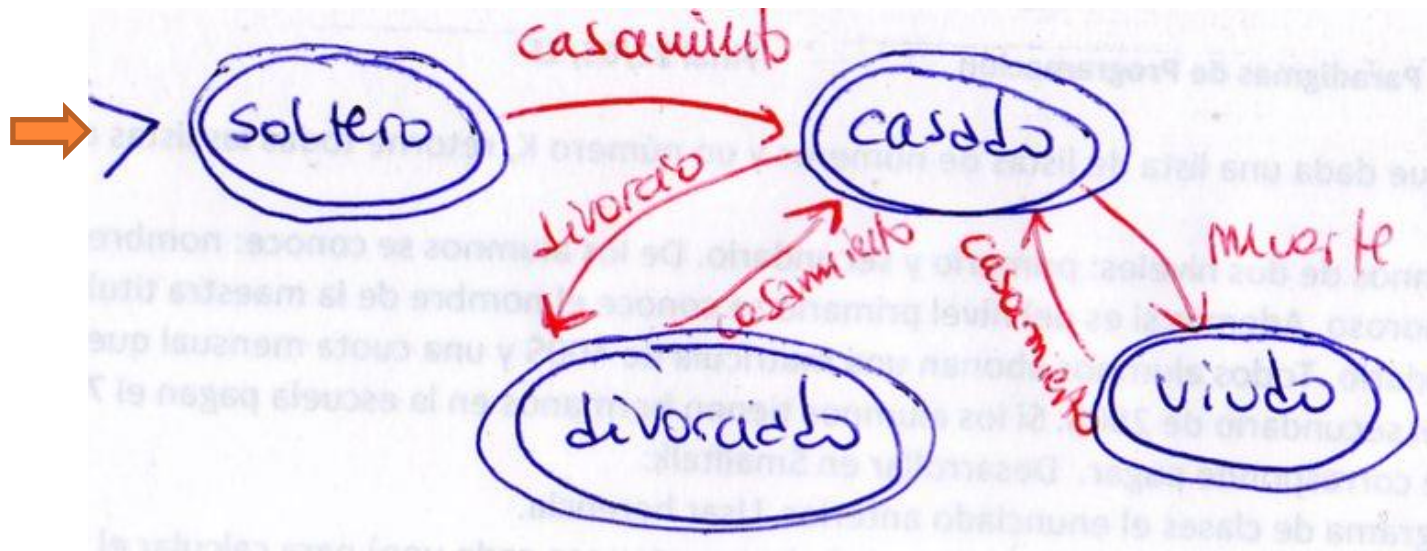
- Los nodos se unen mediante líneas llamadas *transiciones o eventos* que representan situaciones instantáneas que marcan un cambio de un estado a otro.
- Siempre existe *un solo estado inicial, uno o más estados finales* y 0 o más estados intermedios.
- El grafo se recorre a partir del estado inicial.
- Todos los estados son excluyentes entre sí.



AUTÓMATAS FINITOS: EJEMPLO 1

Veamos como ejemplo la modelización de los estados civiles de una persona argentina. Soltero, casado, divorciado, viudo son estados.

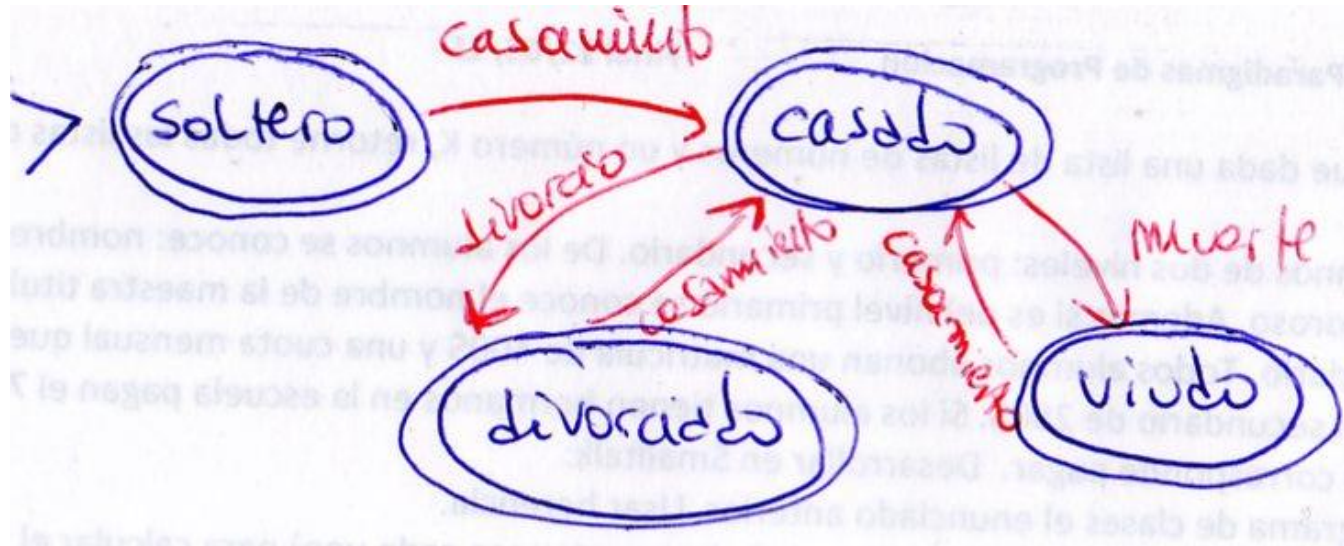
El signo 'mayor que' indica cuál es el estado inicial.



Las flechas indican a través de qué evento se pasa de un estado anterior a uno nuevo. Ej: soy soltero, me caso(evento) paso a estado casado.

Un estado final (círculo doble) indica un estado válido o aceptado por el modelo.

AUTÓMATAS FINITOS: EJEMPLO 1

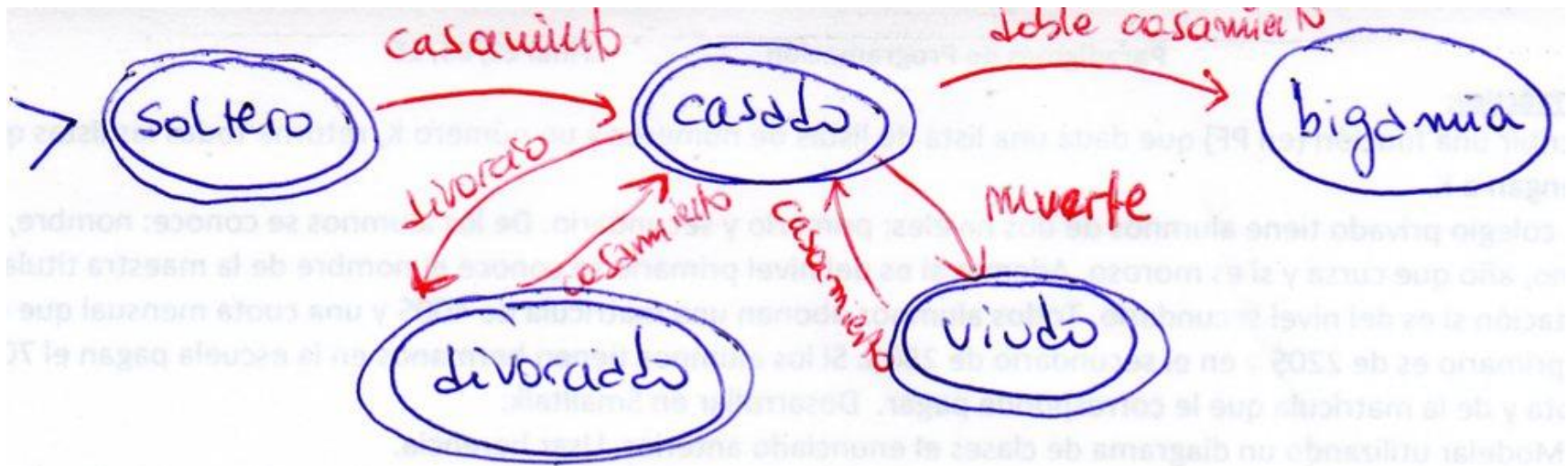


En este grafo todos los estados son finales. El estado inicial también es final (puedo quedarme soltero toda la vida).

En nuestro ejemplo no hay estados no finales o inválidos.



AUTÓMATAS FINITOS: EJEMPLO 1



Si agrego como nuevo estado bigamia veremos que es un estado no final (círculo simple), ya que muestra una situación inválida para el modelo: para las leyes de nuestro país es ilegal ese estado.

El grafo lo modela del mismo modo que en la realidad.



AUTÓMATAS FINITOS: OBJETIVO

- El propósito en estos modelos de estados y eventos es el de reconocer *secuencias de acciones legales o inválidas*:
 - Se parte de un estado inicial, se recorre el grafo pasando de un estado a otro por medio de eventos.
 - Si se llega finalmente a un estado final, se dice que la secuencia de acciones es válida. Si se termina en un estado no final, dicha secuencia es inválida.
- El camino recorrido se llama **trayectoria**.



AUTÓMATAS FINITOS Y LENGUAJES

- Este concepto de autómata finito se aplica también sobre cualquier lenguaje formal L ya sea para verificar si una palabra dada w pertenece a L (si es válida o buena) o para generar las palabras de L .
- En el grafo se modeliza de la siguiente manera:
 - Cada transición o evento (flecha) se etiqueta con un carácter del alfabeto sobre el que está definido el lenguaje.
 - Cada estado (nodo) representa una regla de formación de las palabras de L



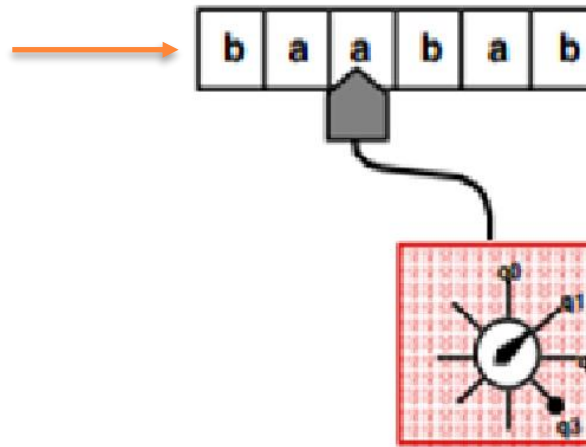
AUTÓMATAS FINITOS Y LENGUAJES

- Así, para verificar la validez de una palabra W se debe chequear que cada carácter de la palabra pertenezca al alfabeto sobre el cual está definido L .
- Luego hay que ver si la posición que ocupa dicho carácter en la palabra respeta las reglas de definición del mismo. Para hacer esta validación se usa el grafo que en este caso cumple una función reconocedora.



AUTÓMATAS FINITOS Y LENGUAJES

El autómata es una máquina reconocedora, lee/recibe de a una letra.



Por cada letra, verifica si puede pasar a algún estado.

(a) Autómata

- Se ingresa de a un carácter de la palabra al autómata finito y se intenta encontrar un camino en el grafo, es decir, que haya una flecha para cada carácter que me lleve a un estado q_i .

AUTÓMATAS FINITOS Y LENGUAJES

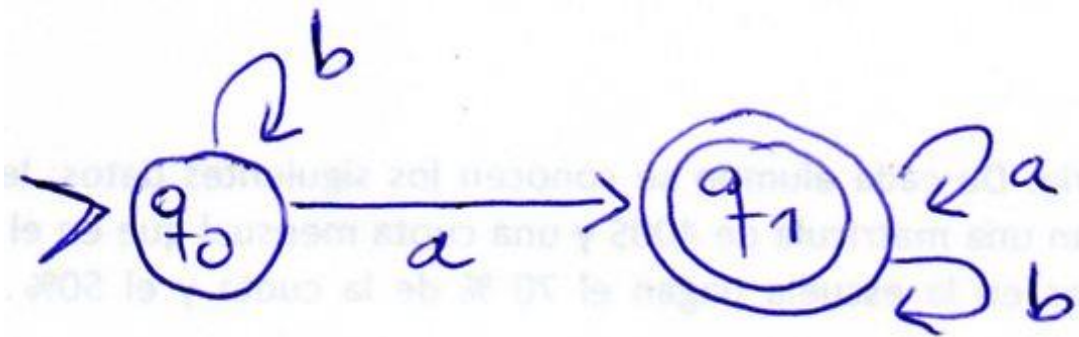
- Se inicia el recorrido en el estado inicial con el primer carácter de la palabra. Después se toma cada carácter de la palabra a verificar y debe concordar con una transición que parta del estado actual y llegue a otro estado posterior.
- Al terminar de evaluar la palabra, se debe haber recorrido parte del grafo y haber llegado a un estado final para ser considerada legal.



AF Y LENGUAJE: FUNCIONAMIENTO

EJEMPLO

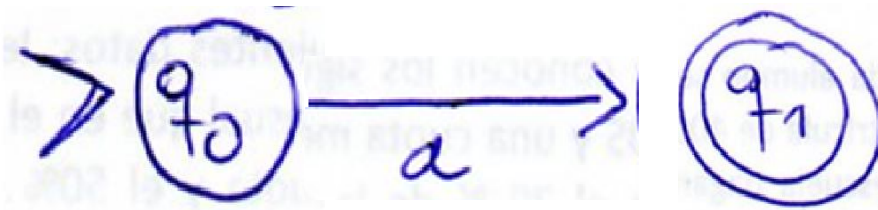
- Sea el alfabeto $\Sigma=\{a,b\}$ y el lenguaje L formado por las palabras w tal que w contiene una o más letras 'a'. $L=\{ w / w \text{ contiene al menos una letra a} \}$
- El grafo o autómatata finito (AF) que representa este lenguaje es el siguiente:



AF Y LENGUAJE: FUNCIONAMIENTO

EJEMPLO

Para dibujar el AF se comienza por el estado inicial, que representa la cadena vacía:



Luego se dibuja la restricción o regla sintáctica especificada: contiene al menos una letra 'a'. Para ello se traza una flecha del estado inicial a un estado nuevo q1.

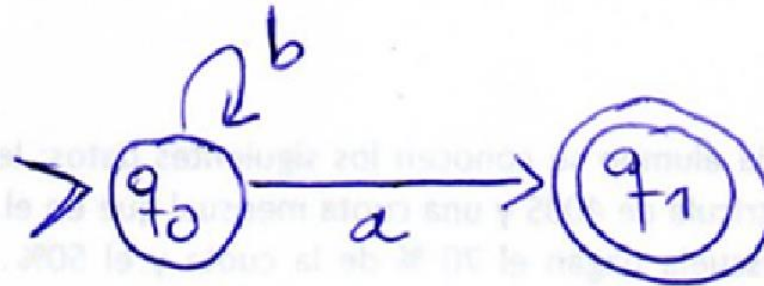
Ese estado es final, porque está representando el caso más simple de la regla sintáctica: una palabra formada por una sola 'a'.



AF Y LENGUAJE: FUNCIONAMIENTO

EJEMPLO

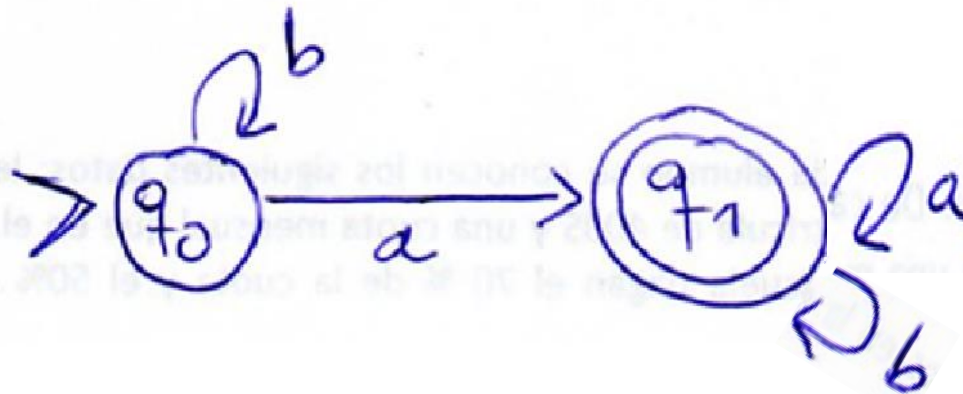
- Ahora hay que completar el grafo: se deben representar todas las posibilidades restantes. En el estado inicial puede venir una o varias letras 'b' antes que la 'a'. Me deja en un estado inválido porque una palabra solo con letras 'b' no cumple la regla sintáctica dada. Se dibuja una flecha rulo en q_0 que significa que se queda en ese estado hasta que llegue una letra 'a'.



AF Y LENGUAJE: FUNCIONAMIENTO

EJEMPLO

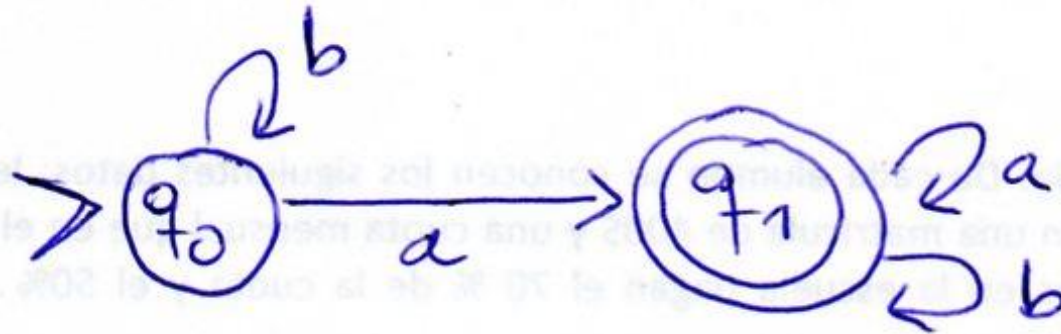
- Ya no hay más letras del alfabeto para usar en q_0 . Pasamos al nodo q_1 y completamos con flechas para la letra 'a' y la letra 'b'.



- Se supone que luego de que llegó una letra 'a' pasamos al estado q_1 , ahora no importa si llega una o varias letras 'a' y/o letras 'b'. La regla ya está cumplida y formarán palabras válidas.
- El autómata está completo.

AF Y LENGUAJE: FUNCIONAMIENTO

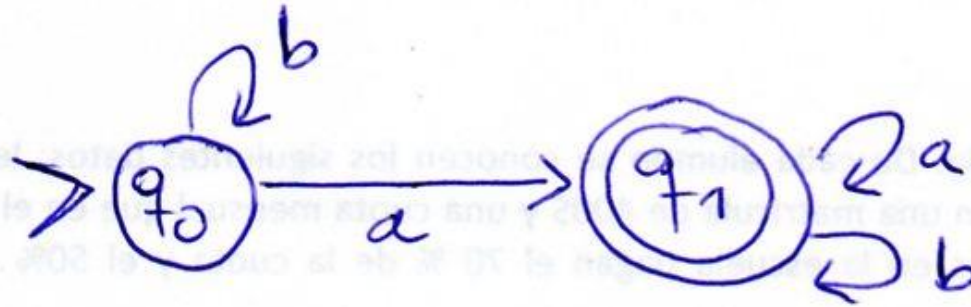
EJEMPLO



- Esto se interpreta así: si llega una o varias 'b' sucesivas, la palabra es inválida (q0 no es estado final).
- Si llega una o varias letras 'b' seguidas de una única letra 'a' la palabra ya es válida (llego a q1 que es final).
- Después de que haya llegado la primer letra 'a', cualquier palabra es válida (me quedo siempre en q1 que es final).



AF Y LENGUAJE: TRAYECTORIAS VÁLIDAS E INVÁLIDAS



- Para la palabra 'ab' la trayectoria es $q_0q_1q_1$ (está en q_0 y recibe una 'a', pasa a q_1 . Ahora en q_1 recibe una 'b' y se queda en q_1). Salió de un estado inicial y terminó en uno final \rightarrow es válida
- La palabra 'bbb' tiene la trayectoria: $q_0q_0q_0q_0$. Salió de un estado inicial y terminó en un estado no final \rightarrow es inválida
- La palabra 'baababba' es válida. Su trayectoria es: $q_0q_0q_1q_1q_1q_1q_1q_1$



AUTÓMATAS FINITOS

○ Se usan para:

- hacer simulaciones, verificación de hipótesis, efectuar predicciones,
- desarrollo de analizadores léxicos (compiladores)
- soft de verificación de circuitos digitales
- soft para explorar páginas web o grandes volúmenes de texto en busca de la aparición de una palabra, frase o patrón
- desarrollo de soft para comprobar protocolos de comunicación y seguridad



AF: CLASIFICACIÓN

- Los autómatas finitos se clasifican en *determinísticos y no determinísticos*, dependiendo de cómo se define la capacidad de cambiar de estado.
- Es *determinístico* si el n° de transiciones (flechas) que salen de un estado coincide con la cantidad de caracteres del alfabeto dado. (AFD)
- Es *no determinístico* si admite que de cada nodo salga un n° de flechas mayor o menor que la cantidad de caracteres del alfabeto. (AFN)



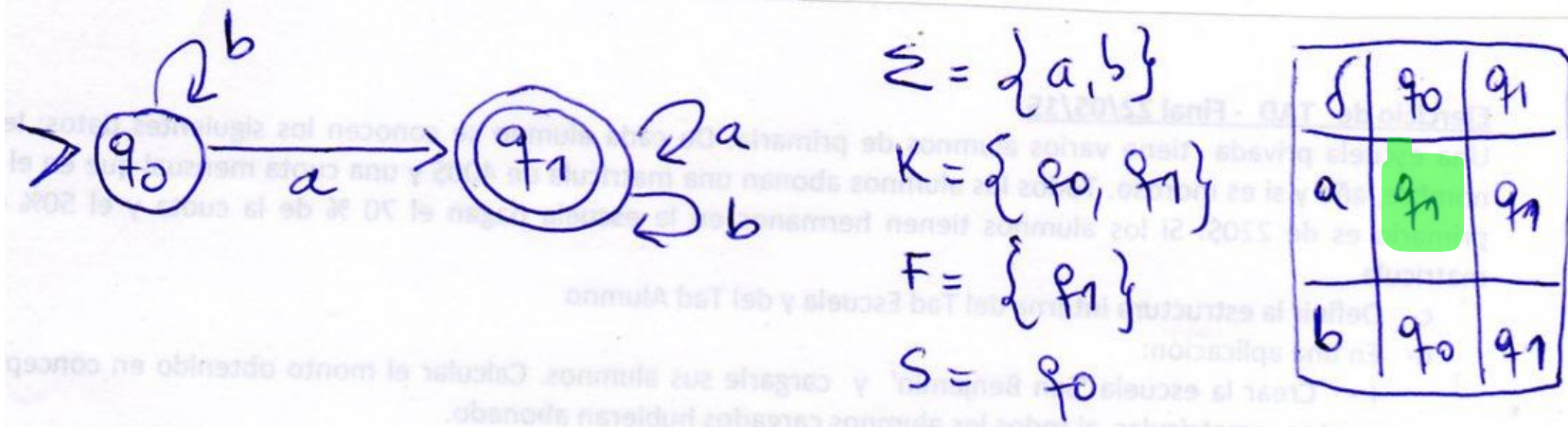
AFD: DEFINICIÓN FORMAL

- Un AFD es una quintupla $(K, S, F, \Sigma, \delta)$ donde:
- K es el conjunto de todos los estados posibles
- S es el estado inicial
- F es el conjunto de estados finales
- Σ es el alfabeto dado sobre el que se construye el lenguaje L
- δ es la función de transición, que a partir de un estado y un símbolo del alfabeto se llega a un nuevo y único estado



AFD: EJEMPLO

Retomemos el ejemplo anterior $\Sigma=\{a,b\}$ y $L=\{w/ w \text{ contiene al menos una letra } a\}$ y desarrollemos su definición formal:



Siempre el estado inicial S es único (por lo tanto NO es un conjunto)

Puede haber 0 o más estados finales F .

K es el conjunto de todos los estados del grafo.

La tabla grafica la función de transición: cómo paso de un estado a otro.

Se lee: estoy en q_0 y llega una 'a' paso a q_1

AFD: REGLAS DE DISEÑO

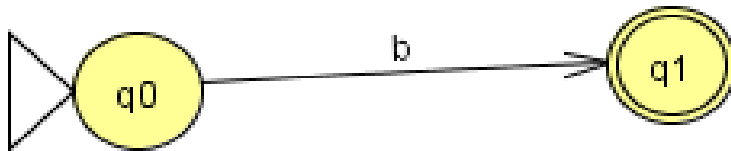
- Todo autómata se debe diseñar a partir de un conjunto de estados que recuerden situaciones o condiciones importantes del problema considerado y luego completar con las transiciones que correspondan. Es decir, siempre se empieza dibujando las restricciones (la regla sintáctica).



AFD: EJEMPLO

Ej. diseñar el AFD que acepte palabras en el alfabeto $\Sigma=\{a,b\}$ que comienzan con una letra 'b'.

Acá la condición a recordar o la restricción es que la primer letra en llegar debe ser una 'b'. Luego se completa con las transiciones que correspondan.



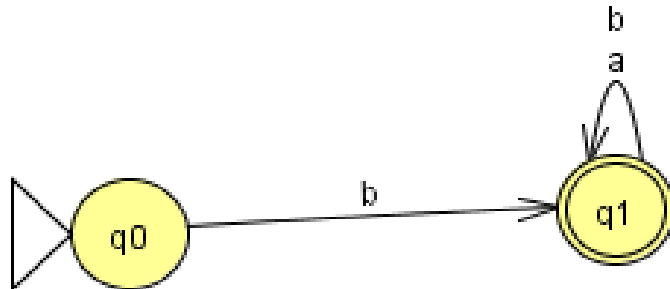
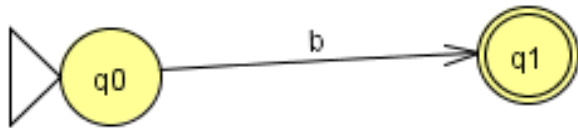
Estado inicial de reposo: q_0

Llega una letra 'b' pasa al estado final q_1 .



AFD: EJEMPLO

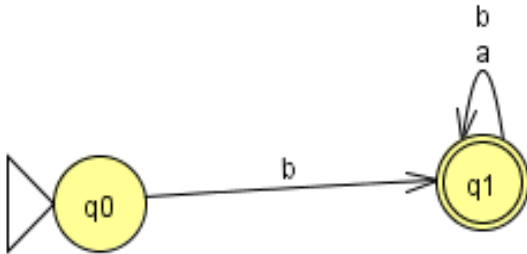
Si estando en q1 llegan letras 'a' y/o letras 'b', se queda en q1 y son todas palabras válidas.



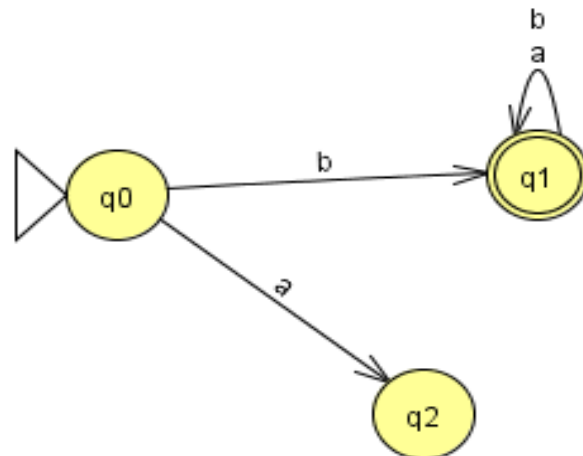
Observar que en q1 se dibujó una sola flecha rulo, pero que tiene 2 etiquetas: 'a' y 'b', que es equivalente a dibujar un rulo para 'a' y otro para 'b'



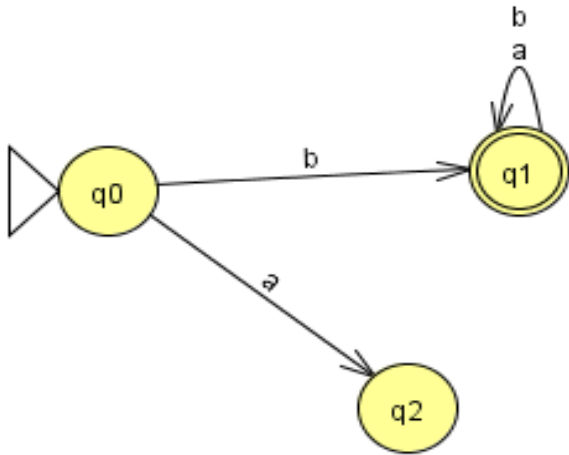
AFD: EJEMPLO



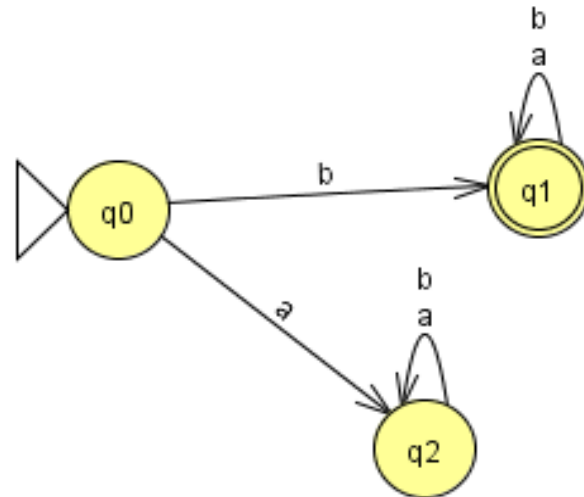
Si estando en q_0 llega una o varias 'a', se va a un estado q_2 como estado inválido: ya que antes de la primer letra 'b' no puede haber otra letra. Por esta razón acá no se usa la flecha rulo con 'a' porque debo mostrar que es incorrecto que haya letras previas a 'b' y que una palabra que comience de esa forma será rechazada..



AFD: EJEMPLO



Finalmente, si estando en q2 llega una o mas letras 'a' y/o letras 'b', se queda en estado q2 como estado inválido.

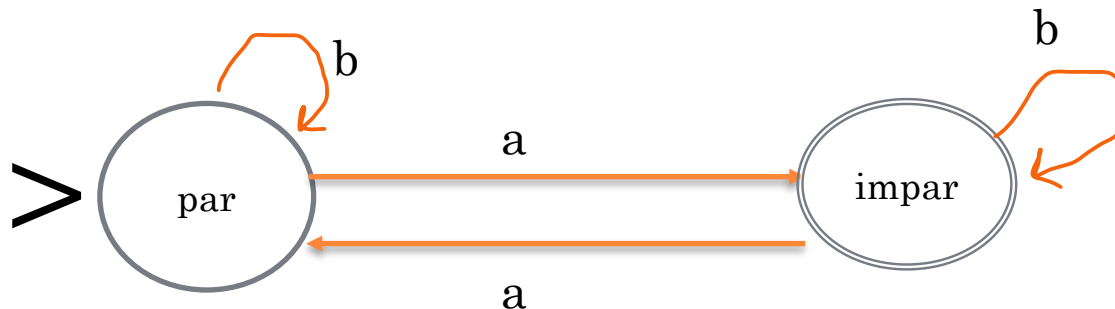


Así el AFD queda completo



AFD: DISEÑO

- Ej. diseñar el AFD que acepte palabras en el alfabeto $\Sigma=\{a,b\}$ con una cantidad impar de letras 'a'
- Acá la condición a recordar o la restricción es la cantidad de letras a recibidas. Luego se completa con las transiciones que correspondan.




- La palabra puede contener cualquier combinación de letras 'a' y 'b', pero debe cumplir con la cantidad impar de 'a'. Ej. ba,abaa,bbbab, etc.
- El estado impar es final.

AFD DISEÑO

- Los errores más comunes en el diseño de autómatas finitos:
 - Confundir estados con transiciones (dibujar un nodo en lugar de una flecha)
 - Poner estados no excluyentes
 - Que sobren estados
 - Que falten estados



AUTÓMATA FINITO NO DETERMINÍSTICO - AFN

- Son una extensión de los AFD y surgen de la posibilidad de permitir que de cada nodo salgan un nro de transiciones menor o mayor que la cantidad de caracteres del alfabeto, además permiten que las transiciones tengan como etiquetas palabras de varias letras o la palabra vacía.
 - Pueden faltar flechas o salir varias de 1 solo nodo con la misma etiqueta.
 - Admite etiquetar las flechas con cadenas de caracteres. En un AFD las etiquetas de las flechas tienen un solo carácter.
- 

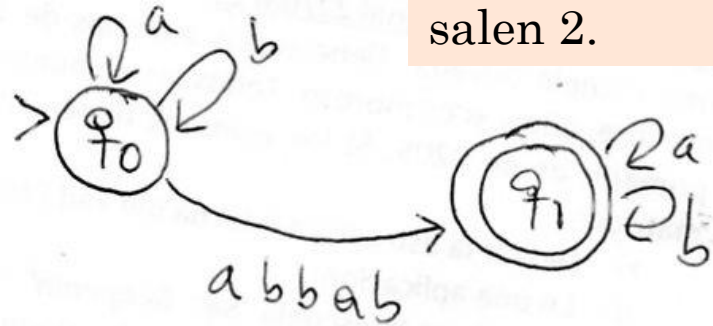
AUTÓMATA FINITO NO DETERMINÍSTICO - AFN

- El problema es que en un AFN no se puede saber qué camino o trayectoria tomar a partir de un estado dado ya que puede haber varias opciones. A veces ocurre que de acuerdo al camino elegido una misma palabra es válida e inválida para el mismo lenguaje L.
- Por eso se los denomina no determinísticos, porque para verificar si una palabra pertenece a un lenguaje dado, es necesario recorrer todas las trayectorias posibles del AFND y comprobar si existe al menos una que valide la palabra.
- Esta es la razón por la cual los compiladores se construyen usando AFD para la etapa de análisis sintáctico.



AFN: EJEMPLO

Del estado q_0 salen 3 flechas y de q_1 salen 2.



- Este autómata refleja un lenguaje formado por las palabras que contienen la cadena 'abbab' en el alfabeto $\{a, b\}$
- Podemos encontrar más de una trayectoria para pasar de un estado a otro:

Sea la palabra 'abbaba'

T1: $q_0 \ q_1 \ q_1$ válida

T2: $q_0 \ q_0 \ q_0 \ q_0 \ q_0 \ q_0 \ q_0$ inválida

Al haber trayectorias válidas e inválidas para la misma palabra, no puede determinar si pertenece o no al lenguaje hasta que encuentre la trayectoria válida



AFD: EJERCICIOS

- Sea $\Sigma = \{0, 1, 2\}$ definir un AFD que permita construir palabras que empiecen con 2 y admita la palabra vacía.
- Primero dibujamos la palabra vacía.

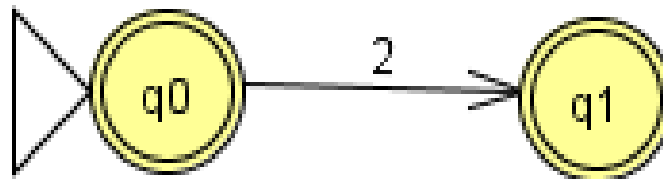


El estado inicial es también estado final, quiere decir que si no ingresa ningún carácter, la palabra vacía es válida.



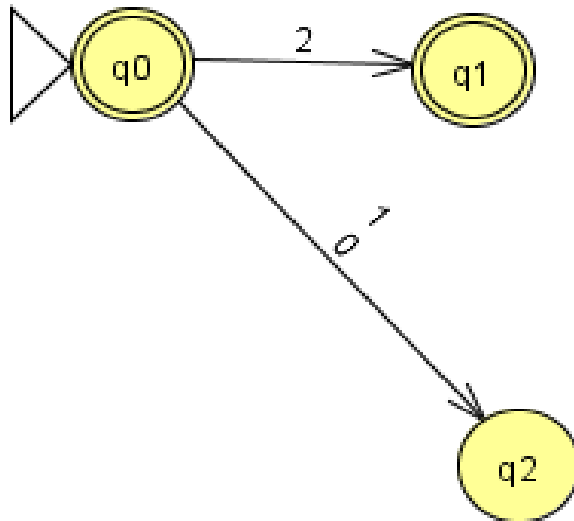
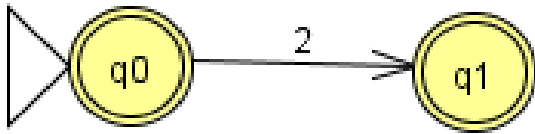
AFD: EJERCICIOS

- Luego dibujamos la palabra más corta que cumpla con la regla. Si se ingresa un 2 la palabra es válida, por lo cual el estado q_1 es estado final.



AFD: EJERCICIOS

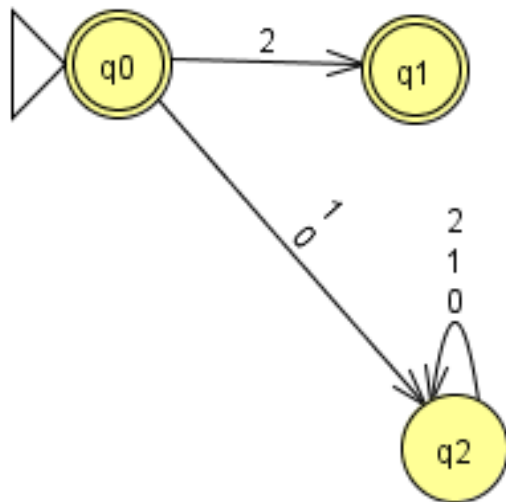
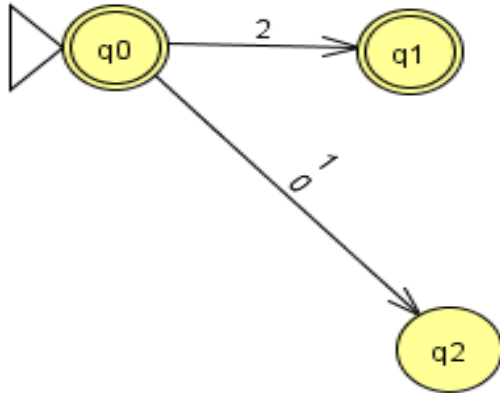
- Ahora completamos las flechas que faltan:



Si en el estado inicial q_0 ingresa un 0 o un 1, la palabra es incorrecta y va a un estado no final q_2 .



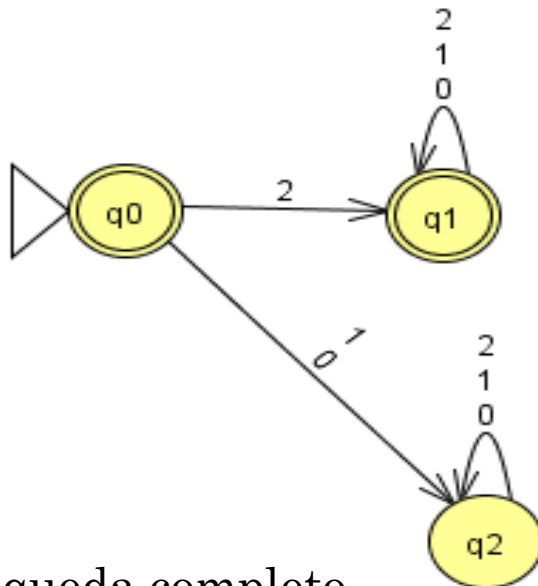
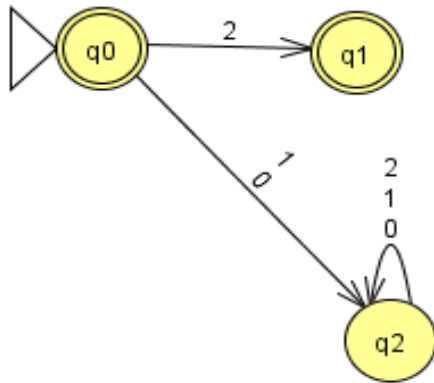
AFD: EJERCICIOS



Estando en q2 cualquier carácter que se ingrese, me deja en ese estado inválido. Se puede dibujar una sola flecha y poner todos los caracteres encolumnados o bien seguidos pero separados por comas (0,1,2)



AFD: EJERCICIOS



Así el AFD queda completo

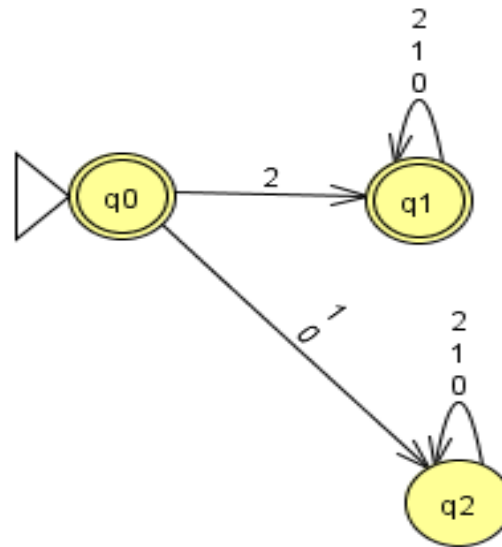
Del mismo modo, estando en q1, cualquier carácter que ingrese formará una palabra válida pues ya empieza con 2.

Se dibuja un solo rulo con tres etiquetas, lo cual equivale a 3 flechas.



AFD: EJERCICIOS

- Veamos la definición formal del AFD graficado:



- $\Sigma = \{0, 1, 2\}$
- $K = \{q0, q2, q1\}$
- $S = q0$
- $F = \{q0, q1\}$
- $\delta = \{ ((q0,2),q1), ((q0,0),q2), ((q0,1),q2), ((q1,0),q1), ((q1,1),q1), ((q1,2),q1), ((q2,0),q2), ((q2,1),q2), ((q2,2),q2) \}$



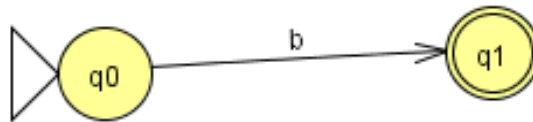
AFD: EJERCICIOS

- Sea $\Sigma = \{a, b\}$ definir un AFD que permita construir palabras que contengan 0 o más letras 'a' seguidas de una única letra 'b'.

$a^n b$ con $n \geq 0$

Ejemplos: b, ab, aab, aaab, aaaab,

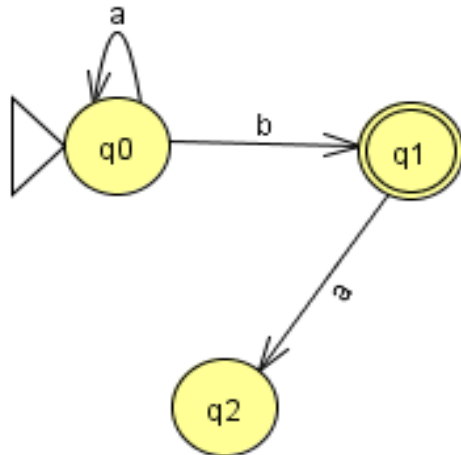
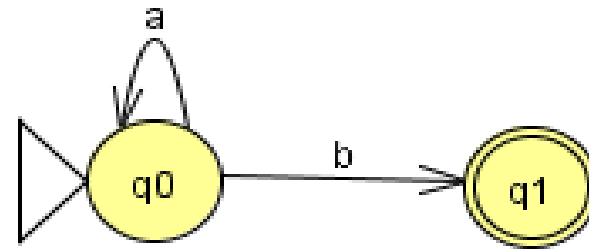
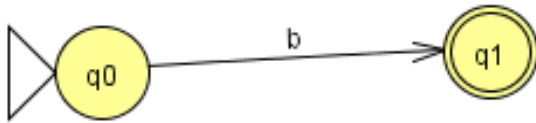
Siguiendo la pautas de diseño, primero se dibuja la condición de la regla sintáctica: la palabra más corta que cumpla con la regla es 'b'



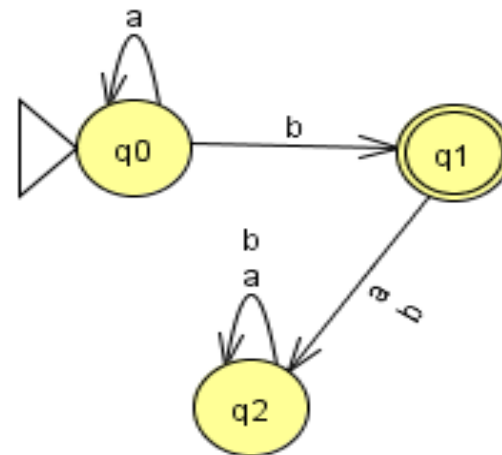
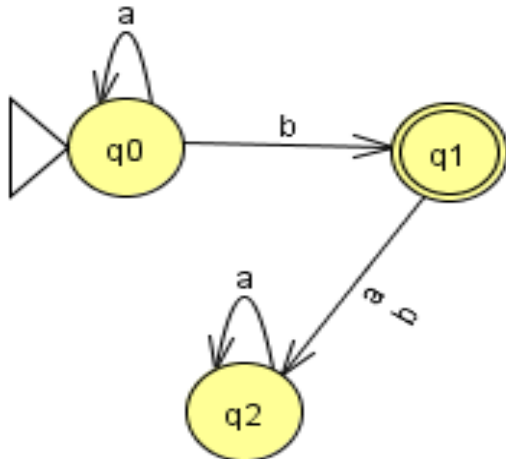
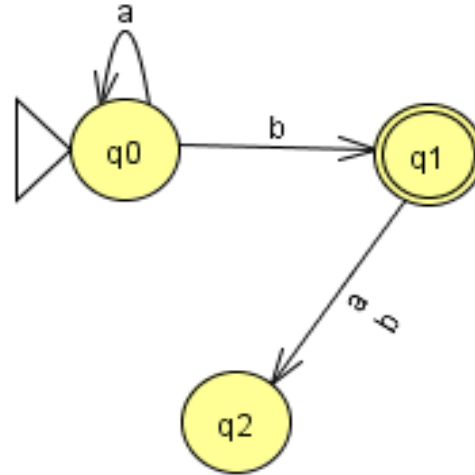
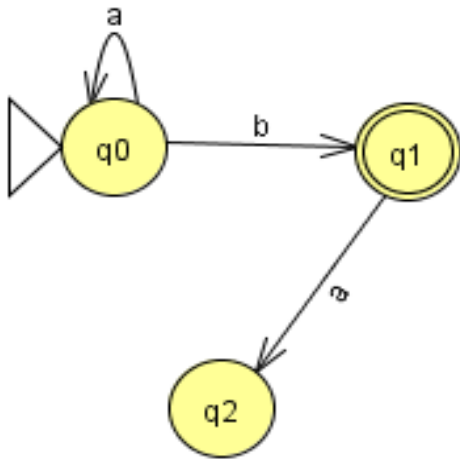
- El estado q1 es final ya que representa la construcción de la palabra más simple de L.
- El estado inicial representa la cadena vacía, que está esperando que se ingrese algún carácter de la palabra a validar.

AFD: EJERCICIOS DE REPASO

- Ahora empezamos a completar cada estado con las flechas que falten de acuerdo a los caracteres de Σ



AFD: EJERCICIOS DE REPASO



Así el AFD queda completo

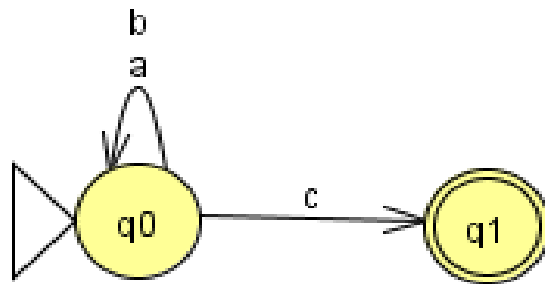


AFD: EJERCICIOS

- Sea $\Sigma = \{a, b, c\}$ definir un AFD que verifique al lenguaje $L = \{w / w \text{ contiene una sola letra 'c'}\}$



Dibujamos la restricción que dice la regla.

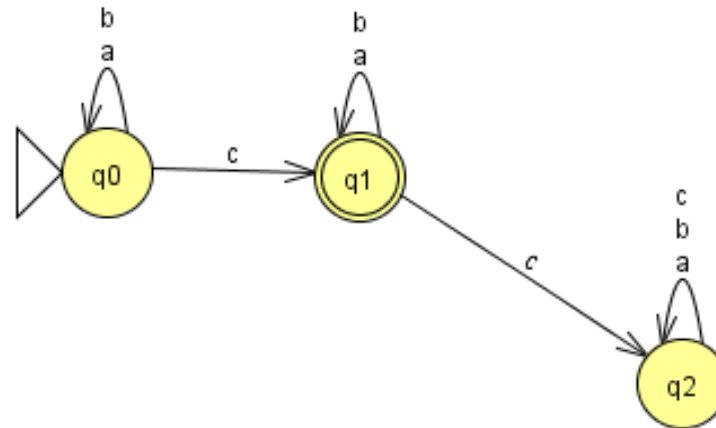
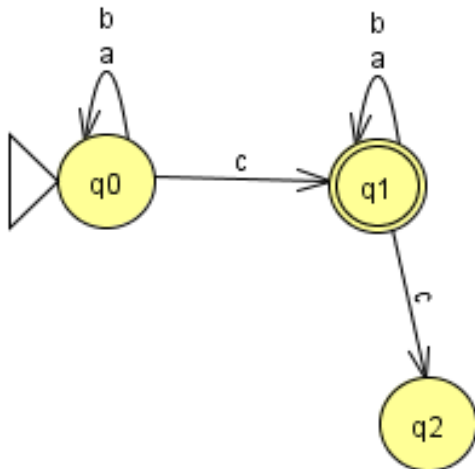
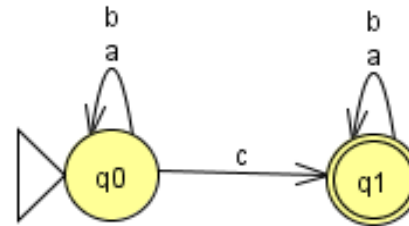
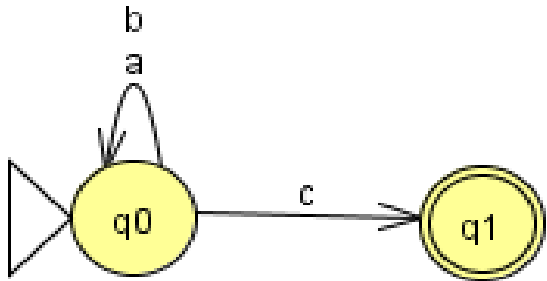


Completamos con las flechas que faltan en q0.



AFD: EJERCICIOS

Seguimos completando el nodo q1



Y por último completamos el nodo q2



