

Teoría de Autómatas e Linguaxes Formais: Introducción

Curso 2019-2020



Senén Barro Ameneiro, CiTIUS

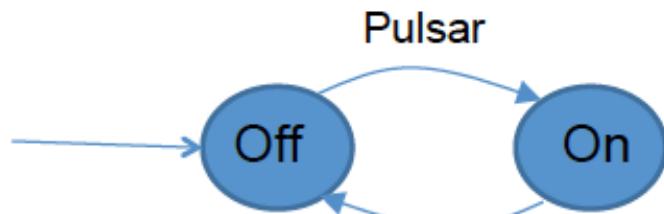
@SenenBarro

Material elaborado fundamentalmente polo
profesor Manuel Mucientes Molina

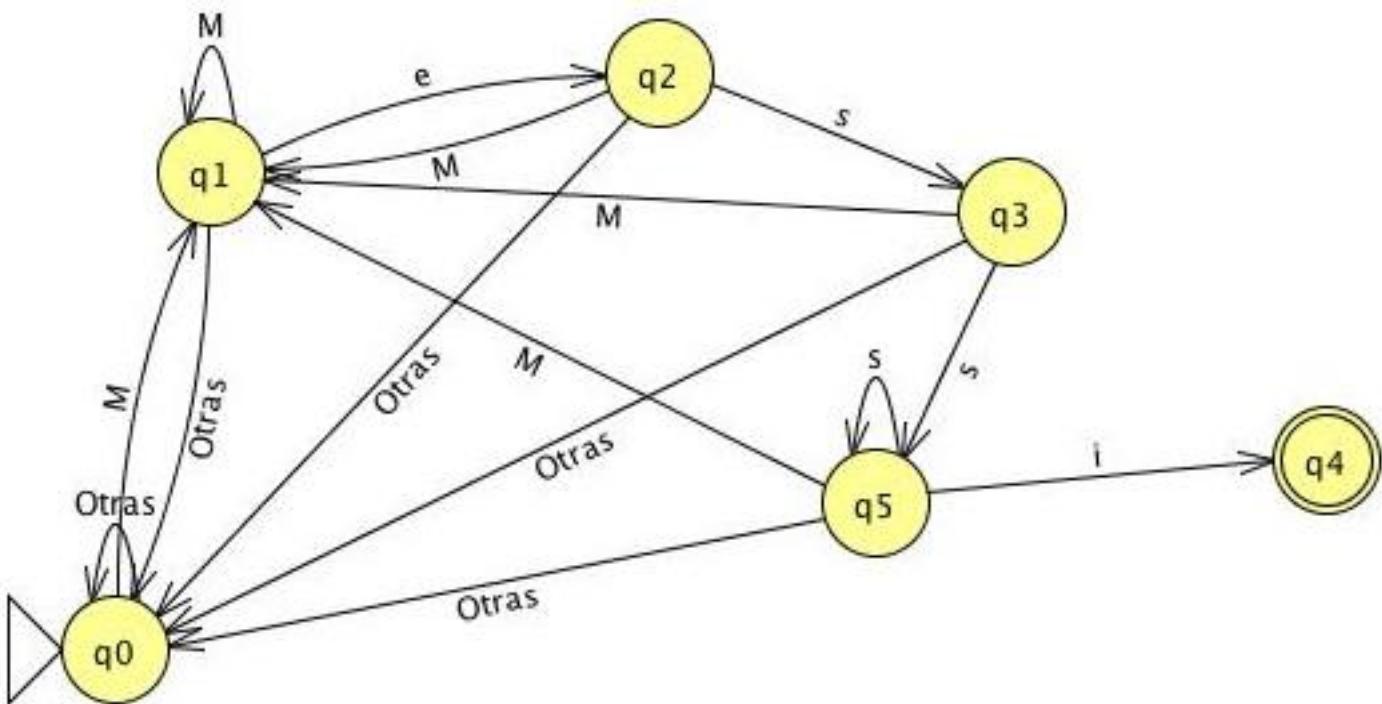
Bibliografía

- J.E. Hopcroft, R. Motwani, y J.D. Ullman. "Teoría de Autómatas, Lenguajes y Computación". Addison Wesley. 2008.
 - capítulo 1
- P. Linz. "An Introduction to Formal Languages and Automata". Jones and Bartlett Publishers, Inc. 2001.
 - capítulo 1

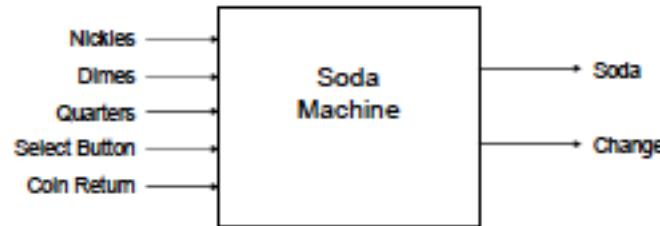
Ejemplos de autómatas MUY simples



Pulsar



Ejemplo: máquina expendedora



Current state	Nickle	Dime	Quarter	Select button	Coin return button
0¢	5¢ ¹	10¢ ¹	25¢ ¹	0¢	0¢ ⁴
5¢	10¢ ¹	15¢ ¹	30¢ ¹	5¢	0¢ ⁴
10¢	15¢ ¹	20¢ ¹	35¢ ¹	10¢	0¢ ⁴
15¢	20¢ ¹	25¢ ¹	40¢ ¹	15¢	0¢ ⁴
20¢	25¢ ¹	30¢ ¹	45¢ ¹	20¢	0¢ ⁴
25¢	25¢ ²	25¢ ²	25¢ ²	0¢ ³	0¢ ⁴
30¢	30¢ ²	30¢ ²	30¢ ²	0¢ ³	0¢ ⁴
35¢	35¢ ²	35¢ ²	35¢ ²	0¢ ³	0¢ ⁴
40¢	40¢ ²	40¢ ²	40¢ ²	0¢ ³	0¢ ⁴
45¢	45¢ ²	45¢ ²	45¢ ²	0¢ ³	0¢ ⁴

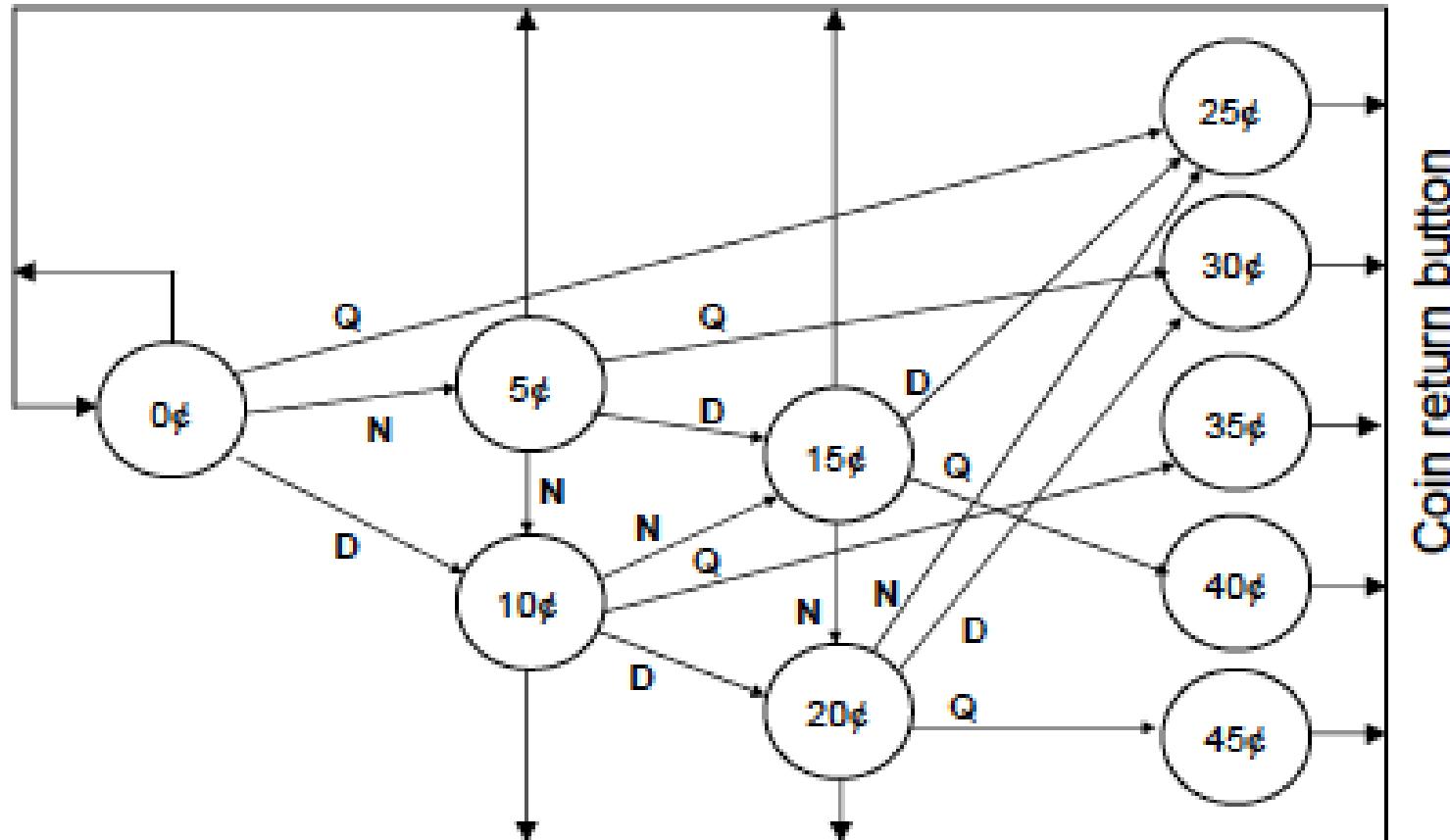
¹ – Accepts Inserted money

² – Any money Inserted is returned

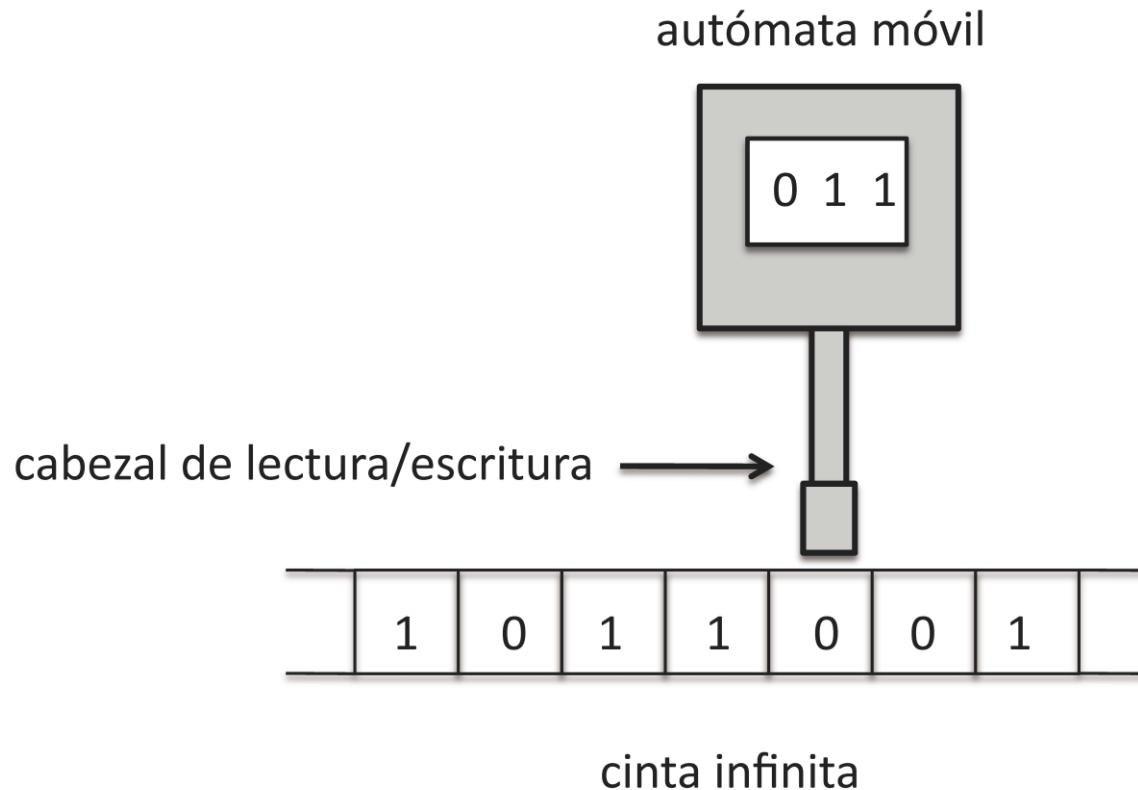
³ – Soda delivered and change returned

⁴ – All Inserted money is returned

Ejemplo: máquina expendedora



Máquina de Turing



¿Qué crees que puede hacer?

Algo de historia

- Teoría de autómatas: estudio de “máquinas” o dispositivos abstractos con capacidad de computación
- Turing (1936): estudio de una máquina abstracta
 - determinar la frontera entre lo que se puede y no se puede hacer con un computador
 - máquina de Turing
- 1940, 1950: autómatas finitos
- Finales de los 50: estudio de las gramáticas formales (Chomsky)
- Cook (1971): separa los problemas que se pueden resolver eficientemente de los que no (intratables)

Utilidad

Autómatas [de número de estados] finito

- Software para el diseño y verificación del comportamiento de circuitos digitales
- Analizador léxico de un compilador
- Software para explorar textos buscando la aparición de ciertos patrones
- Software para comprobar el funcionamiento de un sistema con un número finito de estados diferentes (protocolos de comunicación, protocolos de intercambio seguro de información...)

Utilidad

- Gramáticas
 - Descripción de analizadores sintácticos (*parser*)
- Expresiones regulares
 - Especificación de patrones de cadenas
 - Diseño del software de verificación del formato del texto en formularios web (asignatura de Desarrollo de Aplicaciones Web --3^{er} curso, 1^{er} cuatrimestre--)
- Autómatas y complejidad
 - ¿Qué puede hacer una computadora? Decidibilidad y computabilidad
 - ¿Qué puede hacer una computadora eficientemente? Complejidad

¿Recuerdas la teoría de conjuntos?

- $x \in X, x \notin X$
- $X = \{1, 2, 3\}, X = \{x \mid x \in N \text{ y } x \leq 3\}$
- $X \subseteq N$
- $X \cup Y = \{z \mid z \in X \text{ o } z \in Y\}$
- $X \cap Y = \{z \mid z \in X \text{ y } z \in Y\}$
- $X - Y = \{z \mid z \in X \text{ y } z \notin Y\}$
- Complemento de X respecto a U : $\overline{X} = U - X$
- Leyes de DeMorgan: $\overline{(X \cup Y)} = \overline{X} \cap \overline{Y}$ $\overline{(X \cap Y)} = \overline{X} \cup \overline{Y}$
- Cardinalidad: tamaño de un conjunto, $\text{card}(X)$
 - finito
 - infinito
 - contable o numerable: correspondencia uno a uno con los números naturales
 - incontable

Alfabetos y palabras

- Alfabeto: conjunto finito no vacío de símbolos, Σ
- Cadena o palabra: secuencia finita de símbolos pertenecientes a un alfabeto
 - cadena vacía: λ, ϵ
 - Longitud de la cadena: $|w|$
- Potencias de un alfabeto: conjunto de todas las cadenas de una cierta longitud que se pueden formar con un alfabeto
 - Σ^k
 - $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots$
 - $\Sigma^* = \Sigma^+ \cup \{\lambda\}$

Operaciones con palabras

- **Concatenación de cadenas:** sean x e y dos cadenas. xy es la concatenación de x e y
- **Potencia:** la potencia i -ésima de una palabra x , x^i , se forma por la concatenación i veces de x
- **Reflexión:** si la palabra x está formada por los símbolos $A_1 \dots A_n$, entonces la palabra inversa de x , x^{-1} , se forma invirtiendo el orden de los símbolos en la palabra. $x^{-1} = A_n \dots A_2 A_1$

Lenguajes

- Dado un alfabeto Σ , cualquier $L \subseteq \Sigma^*$ será un lenguaje de Σ
- El alfabeto sobre el que se define el lenguaje debe ser finito, aunque el lenguaje puede tener un número infinito de cadenas
- Ejemplos:
 - Lenguaje vacío: \emptyset
 - Lenguaje que contiene únicamente la cadena vacía: $\{\lambda\}$
 - $L = \{w \mid w \text{ contiene el mismo número de ceros y unos}\}$
 - $L = \{0^n 1^n \mid n \geq 1\}$
 - $L = \{a^i b^j c^k \mid j = i+k \text{ e } i, j, k > 0\}$

Operaciones con lenguajes

- **Unión:** $L_1 \cup L_2$, contendrá todas las palabras que pertenezcan a cualquiera de ellos
- **Intersección:** $L_1 \cap L_2$, contendrá todas las palabras que pertenezcan a ambos
- **Resta:** $L_1 - L_2$, contendrá todas las palabras que pertenezcan a L_1 y no a L_2
- **Concatenación:** $L_1 . L_2$, contendrá todas las palabras que se puedan formar por la concatenación de una palabra de L_1 y otra de L_2

Operaciones con lenguajes

- **Potencia:** la potencia i -ésima de un lenguaje es la concatenación i veces del lenguaje consigo mismo
 - Σ^* : cierre o clausura
 - Σ^+ : clausura positiva
- **Reflexión:** está formada por la aplicación de la reflexión a cada una de las palabras del lenguaje. Se representa por L^-

Gramáticas y autómatas

- Una **gramática** establece la estructura de un lenguaje, es decir, las sentencias que lo forman, proporcionando las formas válidas en que se pueden combinar los símbolos del alfabeto
- Chomsky: clasificación de las gramáticas
 - G0 o de Tipo 0: gramáticas sin restricciones
 - G1 o de Tipo 1: gramáticas sensibles al contexto
 - G2 o de Tipo 2: gramáticas independientes del contexto
 - G3 o de Tipo 3: gramáticas regulares
- $G3 \subseteq G2 \subseteq G1 \subseteq G0$

Gramáticas



Noam Chomsky
(Filadelfia, 7 de diciembre de 1928), lingüista, filósofo, politólogo y activista estadounidense

“En solo cinco años, de 1953 a 1957, un licenciado de la Universidad de Pensilvania, un estudiante de doctorado de poco más de 20 años, se había apoderado de todo un ámbito de estudio, la lingüística, transformándola de arriba abajo, endureciendo esa presunta ciencia social tan esponjosa y convirtiéndola en una ciencia de verdad, una ciencia dura, a la que puso su nombre: Noam Chomsky”.

Tom Wolfe, 2018

Tom Wolfe



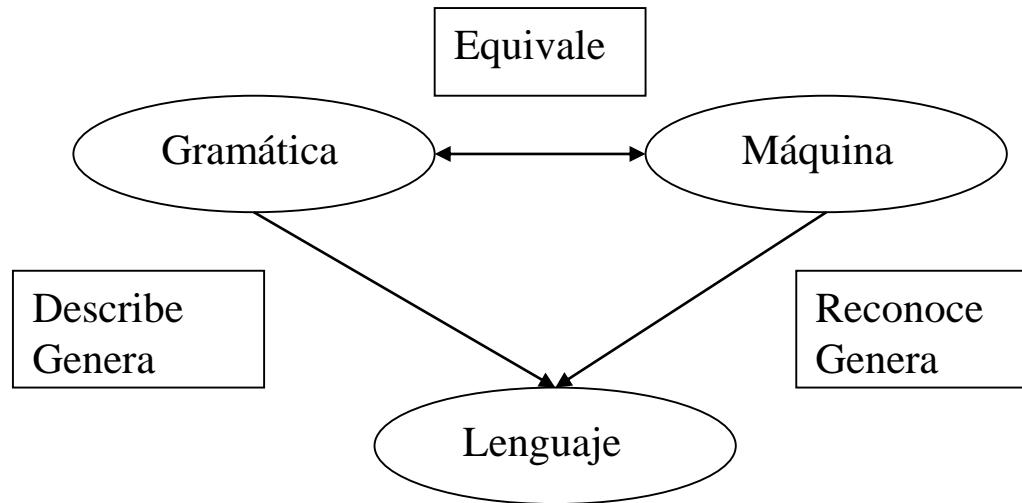
*El reino
del lenguaje*


ANAGRAMA
Colección Argumentos

Gramáticas y autómatas

- Cada gramática es capaz de generar un tipo de lenguaje
 - El lenguaje L será de tipo “ i ” ($i=0,1,2,3$) si existe una gramática de tipo “ i ” capaz de generar o describir dicho lenguaje
- **Máquina abstracta o autómata:** dispositivo teórico capaz de recibir y transmitir información. Dada una cadena de símbolos presentados a su entrada, produce una cadena de símbolos a la salida, en función de dichas entradas y los estados internos por los que transita la máquina.

Lenguajes, gramáticas y autómatas



Gramática	Lenguaje	Máquina
Tipo 0: sin restricciones	Recursivamente enumerable	Máquina de Turing (MT)
Tipo 1: sensible al contexto	Sensible al contexto	Autómata Linealmente Acotado (ALA)
Tipo 2: contexto libre o independiente del contexto	Contexto libre o independiente del contexto	Autómata con Pila (AP)
Tipo 3: regular	Regular	Autómata Finito (AF)

Autómatas finitos

Curso 2019-2020



Senén Barro Ameneiro, CiTIUS

@SenenBarro

Material elaborado fundamentalmente por
el profesor Manuel Mucientes Molina

Bibliografía

- J.E. Hopcroft, R. Motwani y J.D. Ullman,
"Teoría de Autómatas, Lenguajes y
Computación", Addison Wesley, 2008.
 - Capítulo 2 y capítulo 4 (sección 4)
- P. Linz, "An Introduction to Formal Languages
and Automata", Jones and Bartlett Publishers,
Inc, 2001.
 - Capítulo 2

Introducción

- Máquinas secuenciales
 - Mealy
 - Moore
- Autómatas de [número de] estados finito: son máquinas secuenciales
- Reconocen los lenguajes regulares
- Clasificación:
 - Determinista (AFD): el autómata no puede estar en más de un estado simultáneamente
 - No determinista (AFN): puede estar en varios estados al mismo tiempo
- No determinismo
 - no añade ningún lenguaje a los ya definidos por los AFD
 - aumento de la eficiencia en la descripción de una aplicación

Autómatas finitos deterministas (AFD)

- Determinista: para cada entrada, existe un único estado al que el autómata puede llegar partiendo del estado actual
- Consta de:
 - un conjunto finito de **estados**, Q
 - un conjunto finito de **símbolos de entrada**, S
 - una **función de transición** (δ) que, dados un estado y una entrada, devuelve un estado. $\delta(q, a) = p$
 - un **estado inicial** (uno de los estados de Q), q_0
 - un conjunto de **estados finales o de aceptación** (subconjunto de Q), F
- $A = (Q, S, \delta, q_0, F)$

Funcionamiento de un AFD

- **Lenguaje del AFD:** conjunto de las cadenas que acepta
- **Ejemplo:** AFD que acepta todas las cadenas de ceros y unos que contienen la secuencia 01 en algún lugar de la cadena
 - $\{w \mid w \text{ tiene la forma } x01y, \text{ donde } x \text{ e } y \text{ son cualesquiera cadenas de símbolos } 0 \text{ y } 1\}$
 - Si se ha leído la secuencia 01, independientemente de las futuras entradas, el estado debe ser de aceptación
 - Si no se ha leído la secuencia 01, pero la entrada más reciente es 0 y se lee un 1, se pasa a estado de aceptación
 - Si no se ha leído la secuencia 01 y la entrada más reciente es un 1 o no existe, se aceptará la cadena cuando se lea 01
 - $A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\})$

Diagrama de transiciones

- Es un grafo
 - un nodo por cada estado de Q
 - un arco de q a p etiquetado con a para cada $\delta(q, a) = p$
 - una flecha dirigida al estado inicial
 - los estados finales están marcados por un doble círculo

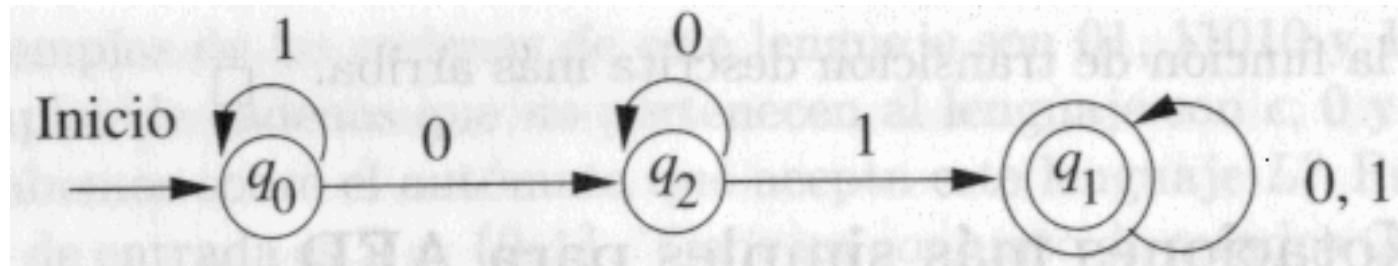


Tabla de transiciones

- Representación tabular de la función δ
- filas: estados
- columnas: entradas
- estado inicial: indicado por una flecha
- estados finales: indicado por: *

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

Extensión a cadenas

- Función de transición: δ
- Función de transición extendida: $\hat{\delta}$
 - dado un estado q y una cadena w , devuelve un estado p
- Definición por inducción de la función de transición extendida
 - Base: $\hat{\delta}(q, \lambda) = q$

$$\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$$

- Paso inductivo: $w = xa$
- Lenguaje de un AFD: lenguaje regular

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \text{ pertenece a } F\}$$

Problemas

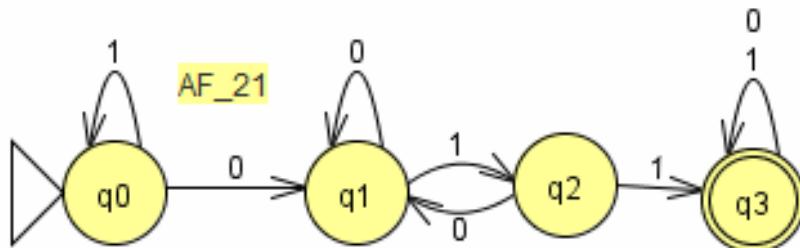
Construir los AFD que acepten los siguientes lenguajes sobre el alfabeto $\{0, 1\}$:

1. El conjunto de cadenas con 011 como subcadena
2. El conjunto de cadenas terminadas en 00
3. El conjunto de cadenas cuyo tercer símbolo desde el extremo derecho sea un 1
4. El conjunto de cadenas que empiezan o terminan por 01
5. El conjunto de palabras que **no** contienen las subcadenas “100”
6. El conjunto de cadenas que contengan un número par (se incluye 0) de subcadenas con un número par de ceros consecutivos. Por ejemplo, el autómata deberá reconocer la cadena **001100010000**, pero no la cadena 00010001**00**

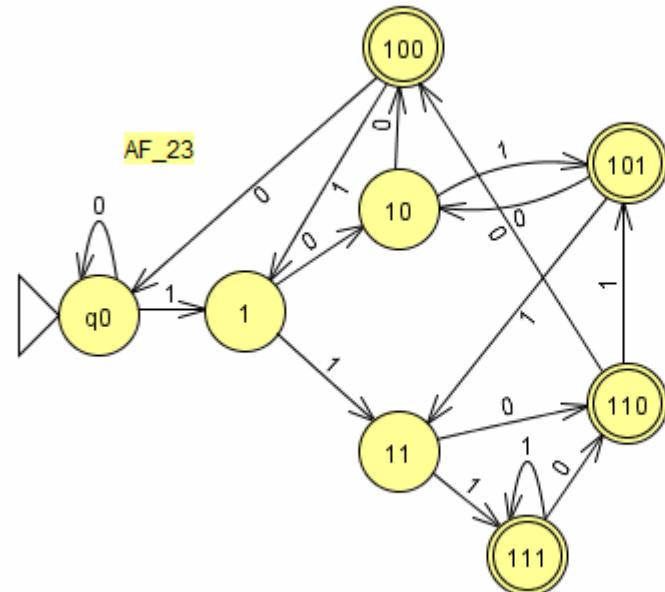
Problemas

Construir los AFD que acepten los siguientes lenguajes sobre el alfabeto $\{0, 1\}$:

1. El conjunto de cadenas con 011 como subcadena

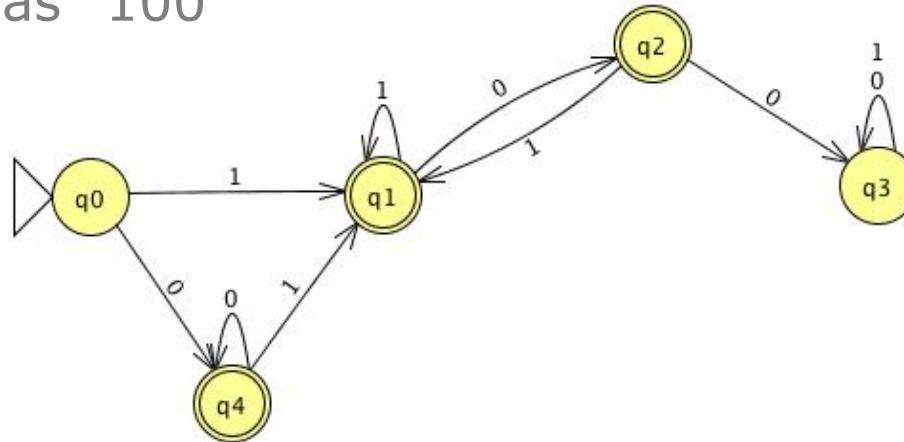


2. El conjunto de cadenas cuyo tercer símbolo desde el extremo derecho sea un 1

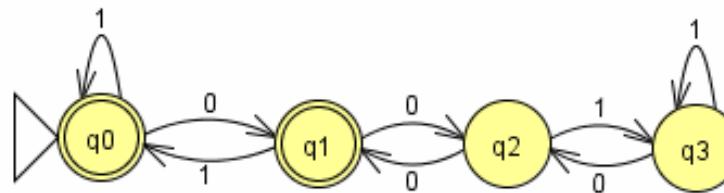


Problemas

5. Lenguaje formado por el conjunto de palabras que no contienen las subcadenas “100”

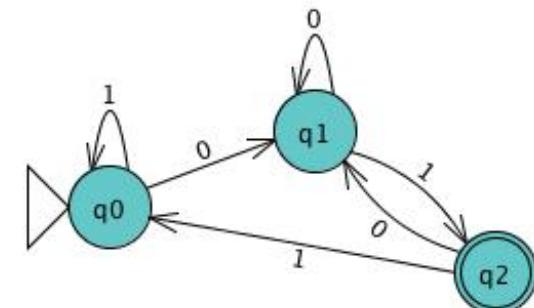
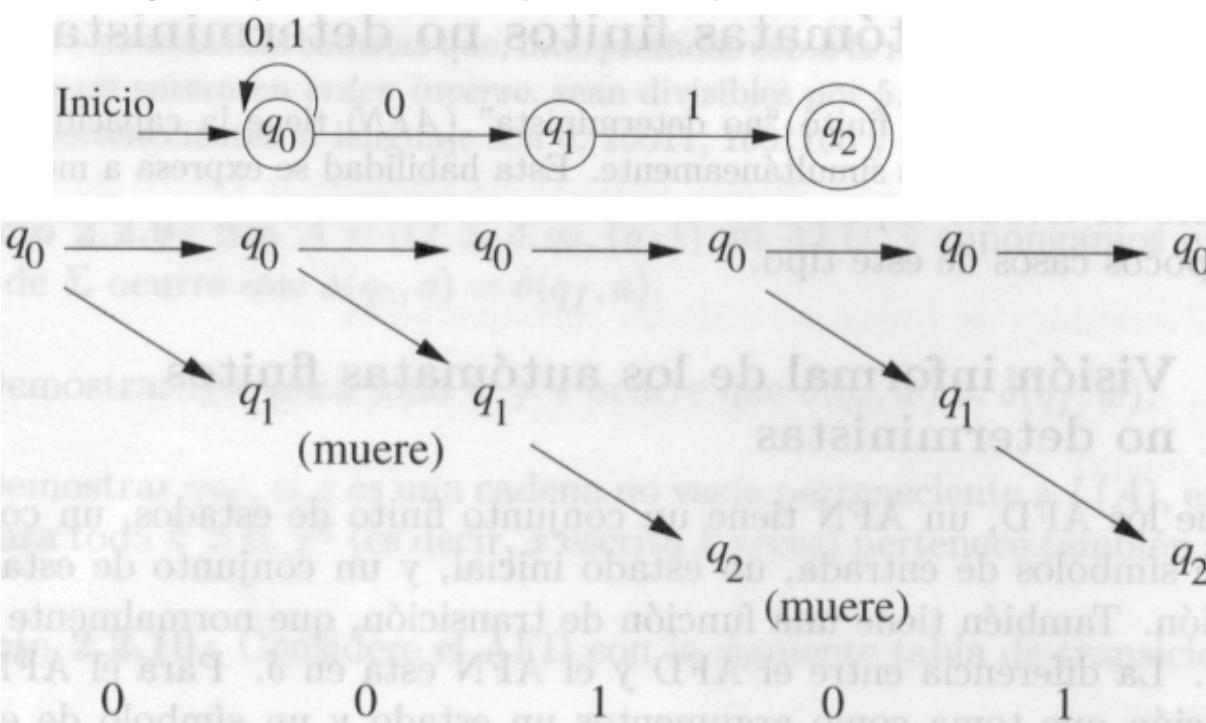


6. Conjunto de cadenas que contengan un número par (se incluye 0) de subcadenas con un número par de ceros consecutivos. Por ejemplo, el autómata deberá reconocer la cadena **001100010000**, pero no la cadena **0001000100**



Autómatas finitos no deterministas

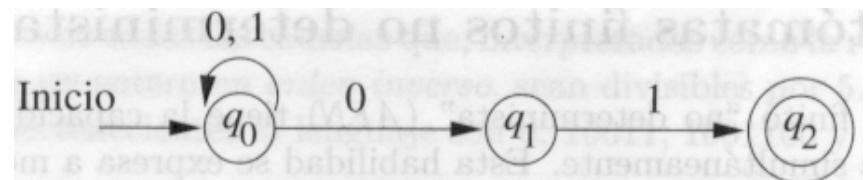
- AFN: capacidad de estar en varios estados simultáneamente
 - Aceptan los lenguajes regulares, igual que los AFD
 - Más compactos y fáciles de diseñar que los AFD
 - Siempre es posible convertir un AFN a un AFD
- Ejemplo: AFN que acepta las cadenas terminadas en 01



Definición de los AFN

- $A = (Q, \Sigma, \delta, q_0, F)$
- δ devuelve en este caso un conjunto de estados y no un sólo estado
- Ejemplo: AFN que acepta las cadenas terminadas en 01: $A=(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	\emptyset	$\{q_2\}$
$*q_2$	\emptyset	\emptyset



Función de transición extendida

- Base: $\hat{\delta}(q, \lambda) = \{q\}$
- Paso inductivo:

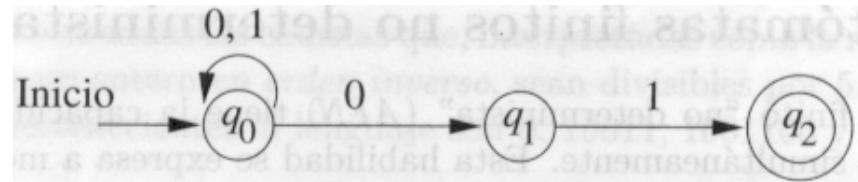
- $w = xa,$

$$\hat{\delta}(q, x) = \{p_1, p_2, \dots, p_k\}$$

$$\bigcup_{i=1}^k \delta(p_i, a) = \{r_1, r_2, \dots, r_m\}$$

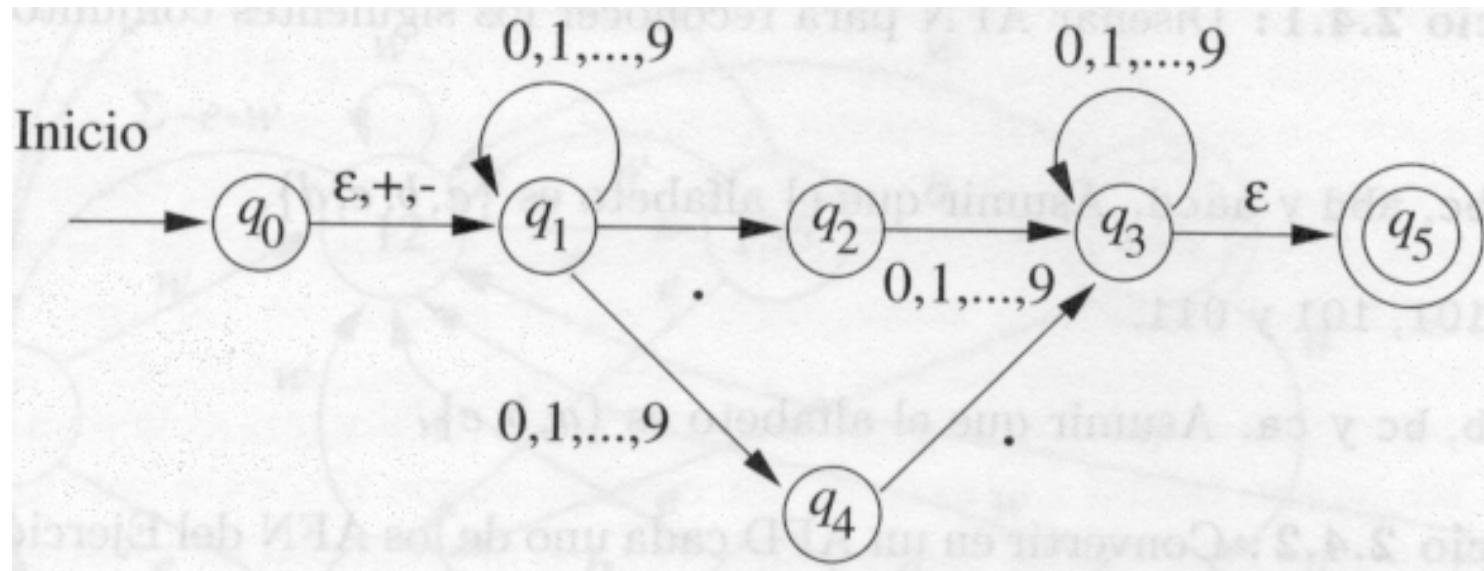
$$\hat{\delta}(q, w) = \{r_1, r_2, \dots, r_m\}$$

- Ejemplo: describir el proceso de la entrada 1001 para el autómata:
- Lenguaje de un AFN: $L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$

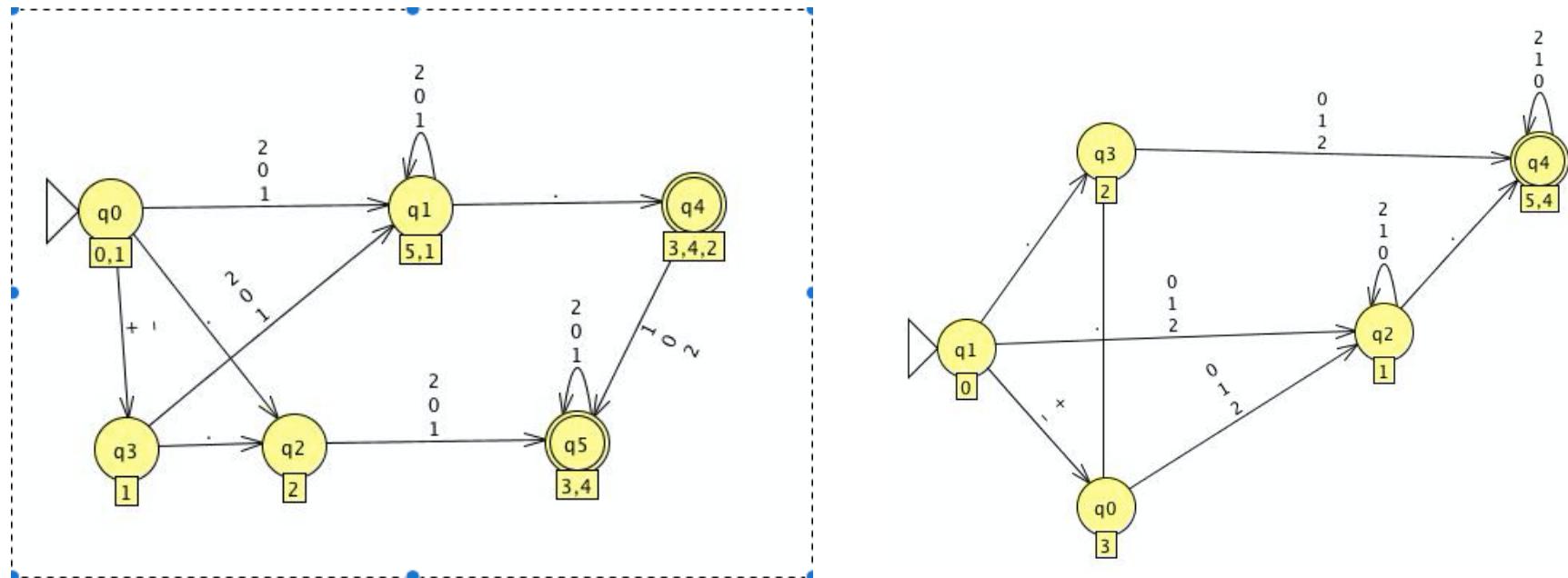
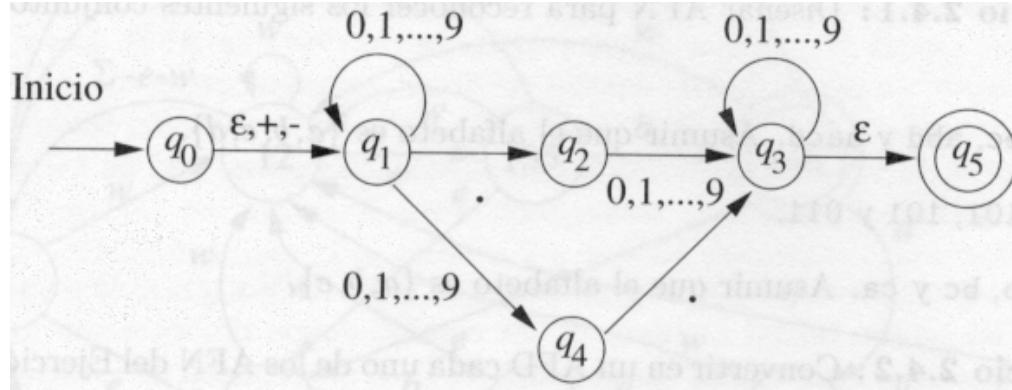


AFN con transiciones ϵ (AFN- ϵ)

- Transiciones para la cadena vacía, ϵ
- No expande la clase de lenguajes que aceptan los AF
- Proporciona “facilidades de programación”
- Ejemplo: AFN- ϵ que acepta números decimales

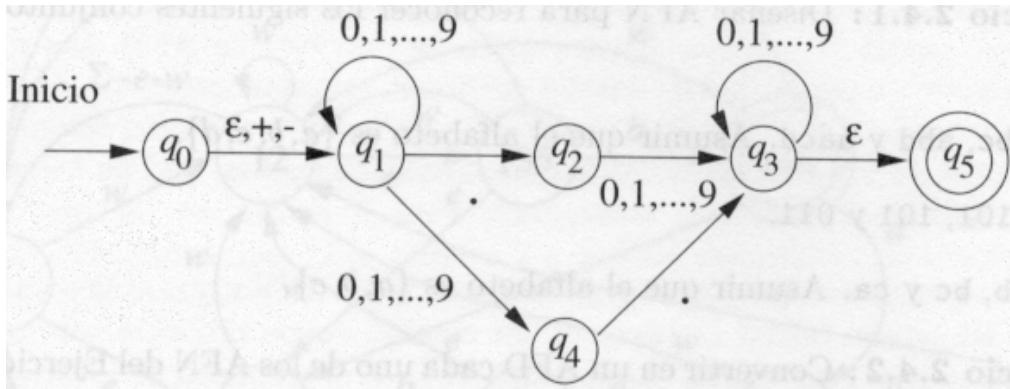


AFN y su AFD equivalente



Notación

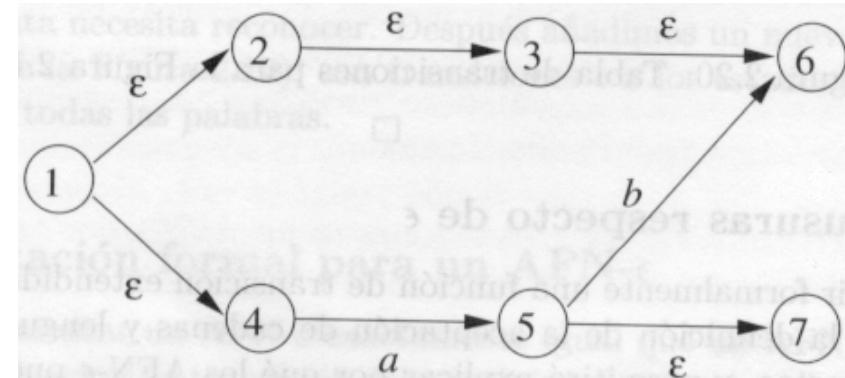
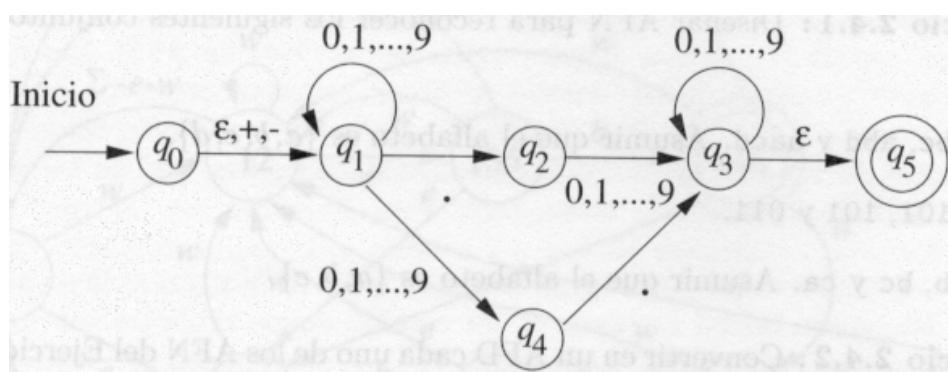
- $A = (Q, \Sigma, \delta, q_0, F)$, pero ahora δ es una función de:
 - un estado de Q
 - un elemento de $\Sigma \cup \{\epsilon\}$
- El símbolo ϵ no puede formar parte del alfabeto
- Ejemplo:



	ϵ	$+, -$	$.$	$0, 1, \dots, 9$
$\rightarrow q_0$	$\{q_1\}$	$\{q_1\}$	\emptyset	\emptyset
q_1	\emptyset	\emptyset	$\{q_2\}$	$\{q_1, q_4\}$
q_2	\emptyset	\emptyset	\emptyset	$\{q_3\}$
q_3	$\{q_5\}$	\emptyset	\emptyset	$\{q_3\}$
q_4	\emptyset	\emptyset	$\{q_3\}$	\emptyset
$*q_5$	\emptyset	\emptyset	\emptyset	\emptyset

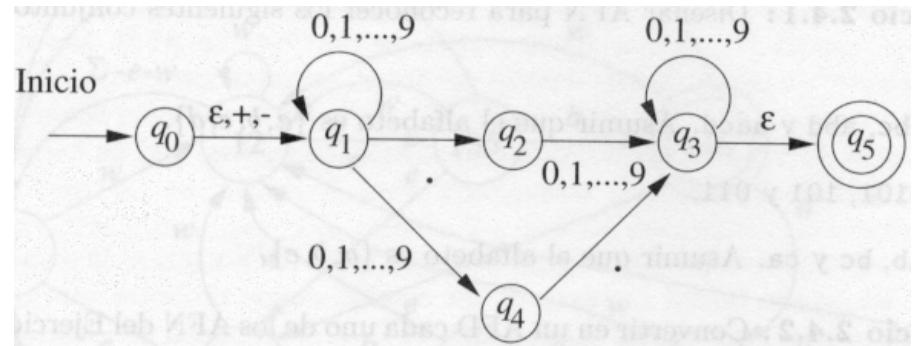
Clausuras respecto de ϵ

- Clausura del estado q respecto de ϵ , $CLAUS_{\epsilon}(q)$
 - Base: el estado q está en $CLAUS_{\epsilon}(q)$
 - Paso inductivo: si δ es la función de transición del AFN- ϵ , y el estado p está en $CLAUS_{\epsilon}(q)$, entonces $CLAUS_{\epsilon}(q)$ contiene todos los estados de $\delta(p, \epsilon)$
- Ejemplos: determinar las clausuras de los estados de los autómatas de las siguientes figuras



Transiciones y lenguajes extendidos

- Definición recursiva de $\hat{\delta}$
 - Base: $\hat{\delta}(q, \varepsilon) = CLAUS_\varepsilon(q)$
 - Paso inductivo: sea $w = xa$, donde a es un elemento de Σ
 - Sea $\hat{\delta}(q, x) = \{p_1, p_2, \dots, p_k\}$
 - Sea $\bigcup_{i=1}^k \delta(p_i, a) = \{r_1, r_2, \dots, r_m\}$
 - $\hat{\delta}(q, w) = \bigcup_{j=1}^m CLAUS_\varepsilon(r_j)$
- Lenguaje de un AFN- ε , $E=(Q, \Sigma, \delta, q_0, F)$:
$$L(E) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$
- Ejemplo: calcular
 $\hat{\delta}(q_0, 5.6)$
para este autómata:



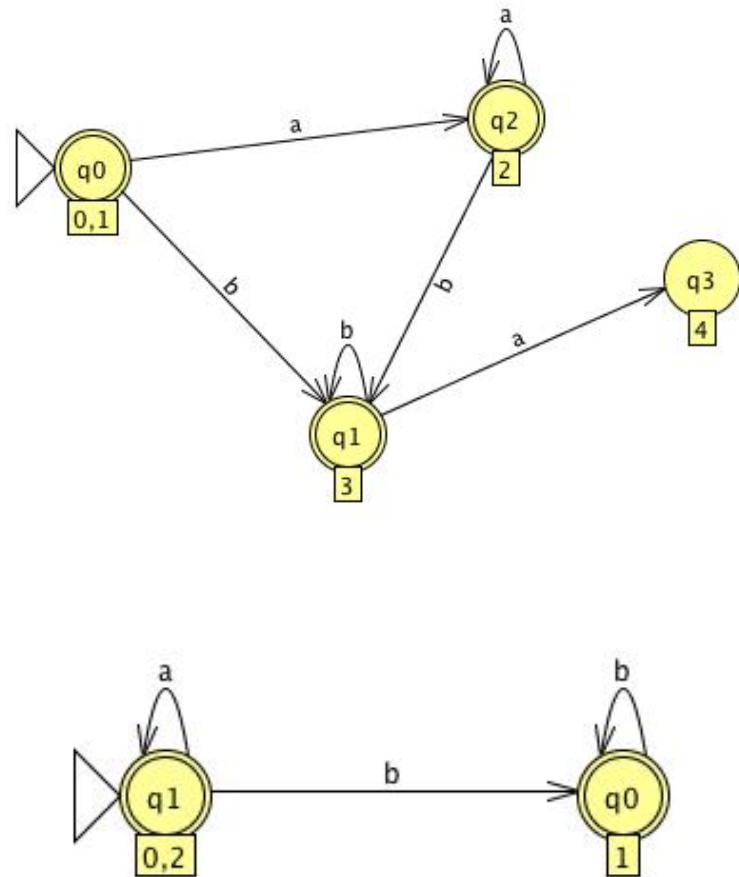
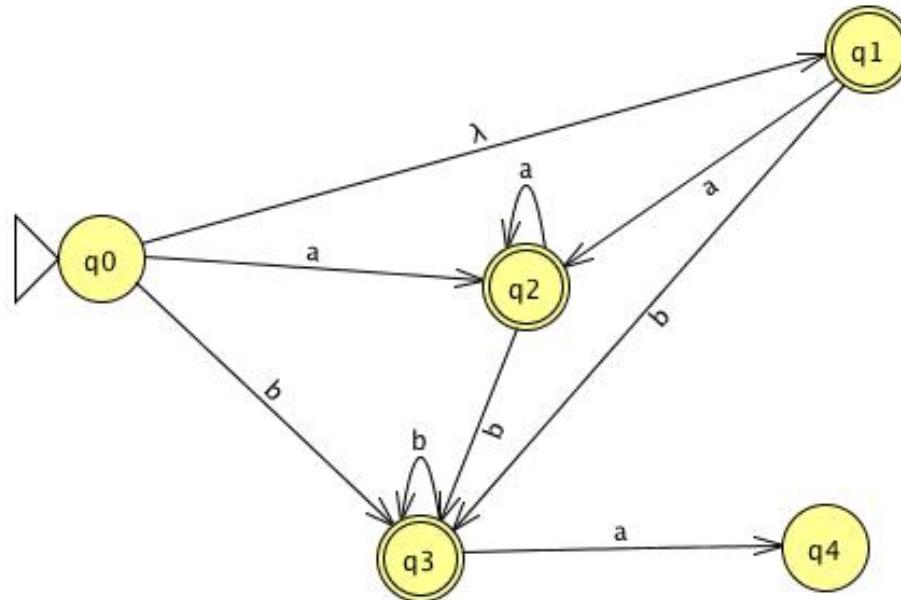
Problemas

Diseñar el AFN- ϵ para los siguientes lenguajes:

1. Conjunto de cadenas con cero o más letras a seguidas de cero o más letras b
2. El conjunto de cadenas formadas por 01 repetido una o más veces o por 010 repetido una o más veces
3. AF que sobre el alfabeto $\{X, Y, Z, 0, 1, 2\}$ reconoce matrículas de vehículos válidas, para las provincias “X”, “YY” y “ZX”. El formato de las matrículas podrá ser:
 - Provincia, 4 números y 0, 1 o 2 letras
 - 4 números y tres letras

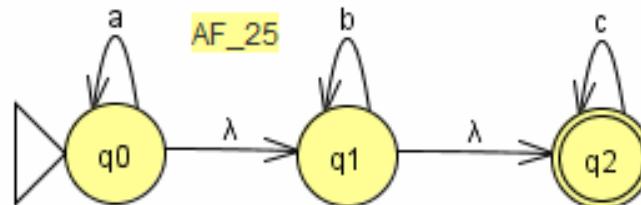
Problemas

Conjunto de cadenas con cero o más letras *a* seguidas de cero o más letras *b*

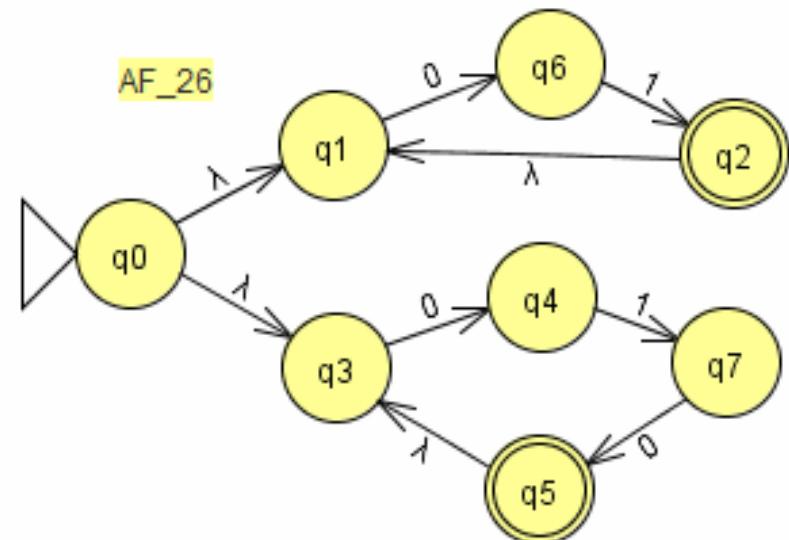


Problemas

1. Conjunto de cadenas con cero o más letras *a* seguidas de cero o más letras *b* seguidas de cero o más letras *c*



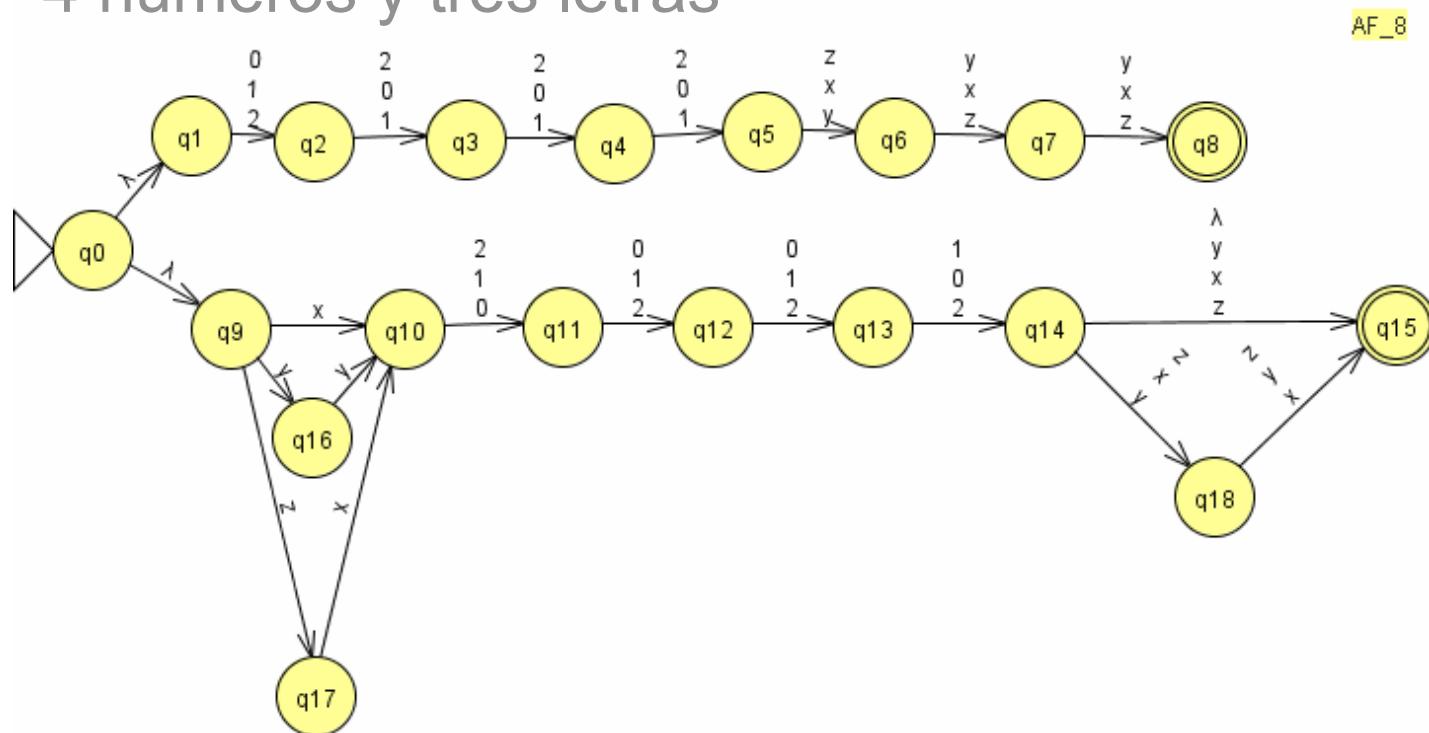
2. El conjunto de cadenas formadas por 01 repetido una o más veces o por 010 repetido una o más veces



Problemas

AF que sobre el alfabeto $\{X, Y, Z, 0, 1, 2\}$, que reconoce matrículas de vehículos válidas, para las provincias “X”, “YY” y “ZX”. El formato de las matrículas podrá ser:

- Provincia, 4 números y 0, 1 o 2 letras
- 4 números y tres letras

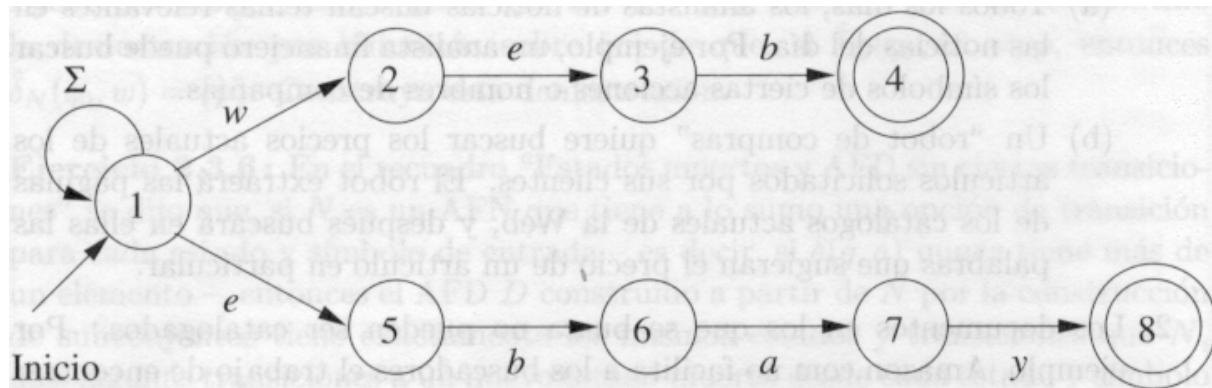


Ejemplo en la búsqueda de texto

- Dado un conjunto de palabras, encontrar todos los documentos que contienen una o más de esas palabras.
- Características que hacen apropiado el uso de autómatas en una aplicación:
 - El repositorio a analizar cambia rápidamente
 - noticias del día
 - robot de compras
 - Los documentos no pueden ser catalogados: Amazon (páginas generadas a partir de consultas)

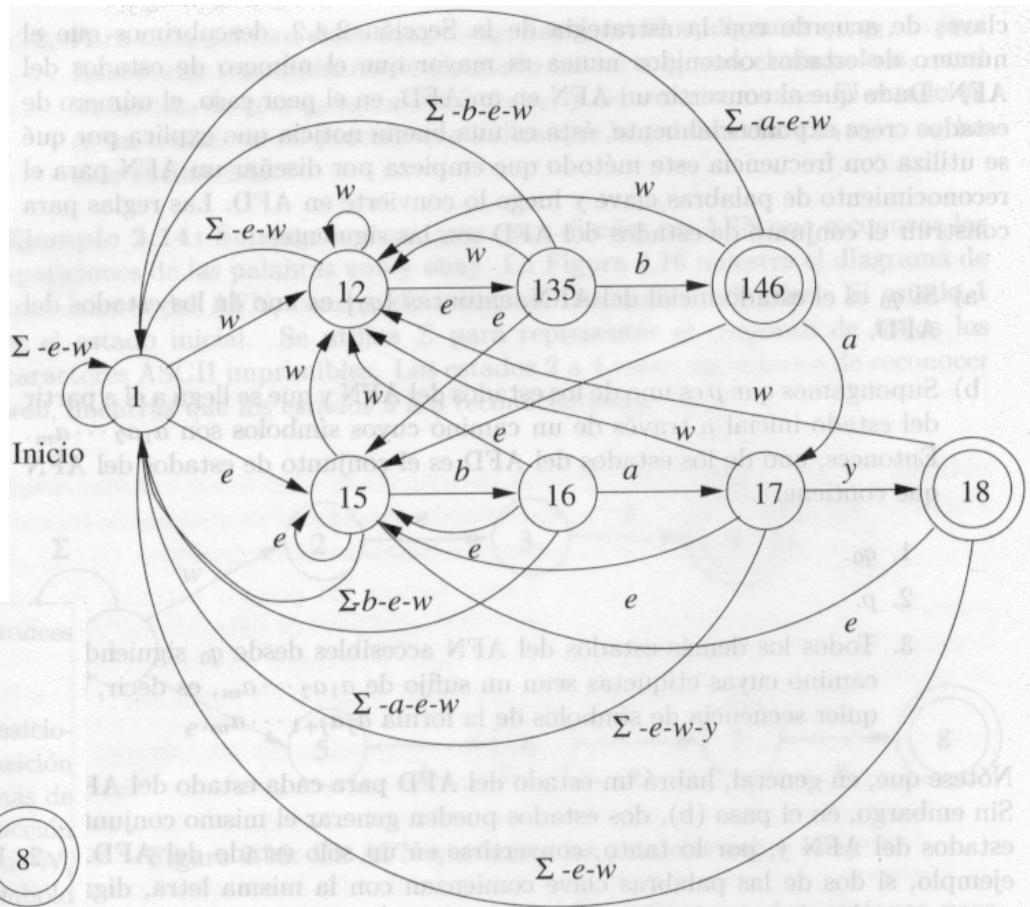
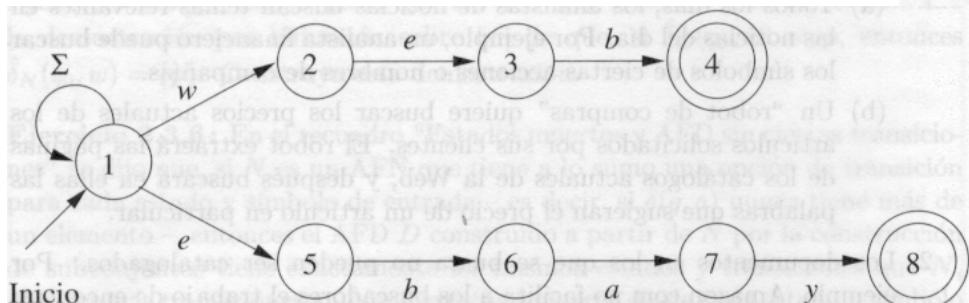
AFN para búsqueda de texto

- Estado inicial con transición a sí mismo para cada símbolo de entrada (conjetura)
- Para cada palabra clave $a_1a_2\dots a_k$, hay k estados, q_1, q_2, \dots, q_k
- Transición del estado inicial a q_1 con a_1 , de q_1 a q_2 con a_2 , etc. El estado q_k indicará que la palabra $a_1a_2\dots a_k$ ha sido aceptada
- Ejemplo: AFN que reconoce las palabras *web* y *ebay*



Ejemplo

- Conversión del siguiente AFN en AFD



Equivalecia entre AFD y AFN

- Todo lenguaje descrito por un AFN también puede ser descrito por un AFD
 - Para un AFN de n estados, el AFD equivalente tendrá como máximo 2^n estados
- Demostración de que un AFD puede hacer lo mismo que un AFN: construcción de subconjuntos
 - construcción de todos los subconjuntos del conjunto de estados del AFN

Construcción de subconjuntos

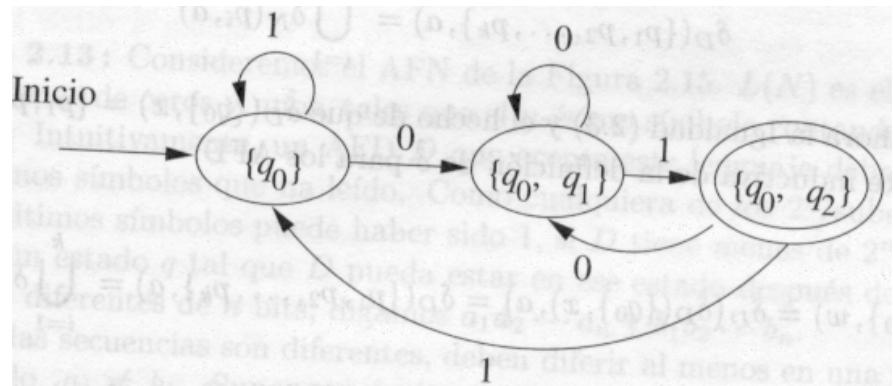
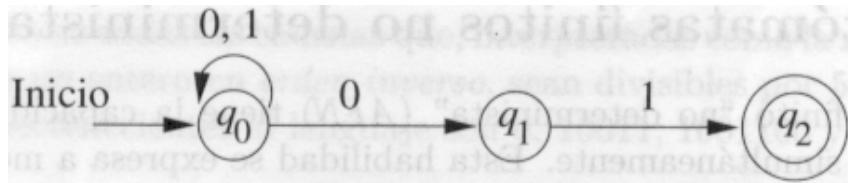
Dado el AFN $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$, obtener el AFD
 $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ tal que $L(D) = L(N)$

- Alfabetos iguales
- Estado inicial: conjunto con el estado inicial de N
- Q_D : conjunto de subconjuntos de Q_N . Si Q_N tiene n estados, Q_D tendrá 2^n . Eliminación de estados no accesibles
- F_D : conjunto de subconjuntos S de Q_N tales que $S \cap F_N \neq \emptyset$
- Para cada $S \subseteq Q_N$ y cada símbolo de entrada a :

$$\delta_D(S, a) = \bigcup_{p \text{ en } S} \delta_N(p, a)$$

Evaluación “perezosa”

- Base: el conjunto de un elemento que contiene el estado inicial de N es accesible
- Paso inductivo: hemos determinado que el conjunto S de estados es accesible. Entonces, para cada símbolo de entrada a , se calcula el conjunto de estados $\delta_D(S, a)$, que también serán accesibles
- Ejemplo: aplicar el proceso al siguiente autómata:



Equivalencia entre AFD y AFN

- Teorema: si $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ es el AFD construido a partir del AFN $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ mediante la construcción de subconjuntos, entonces $L(D) = L(N)$
- Teorema: un lenguaje L es aceptado por algún AFN si y sólo si L es aceptado por algún AFD

Problemas

- Verificar si este AFN y AFD son equivalentes

	0	1
->p	{p, q}	{p}
q	{r, s}	{t}
r	{p, r}	{t}
*s	Ø	Ø
*t	Ø	Ø



A = {p}
B = {p, q}
C = {p, q, r, s}
D = {p, t}

	0	1
->A	B	A
B	C	D
*C	C	D
*D	B	A

Problemas

Completar el AFD equivalente al AFN dado

	0	1
->p	{p, q}	{p}
q	{r}	{r}
r	{s}	\emptyset
*s	{s}	{s}



A = {p}
B = {p, q}
C = {p, r}
D = {p, q, r}
E = {p, q, s}
F = {p, q, r, s}
G = {p, r, s}
H = {p, s}

	0	1
->A	B	A
B		C
C	E	
D	F	C
*E	F	
*F	F	G
*G	E	
*H	E	H

Eliminación de transiciones ϵ

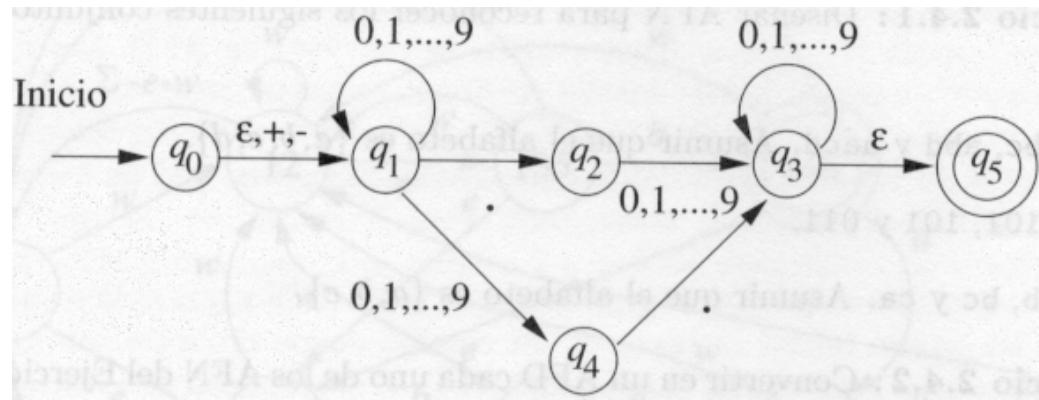
- Dado un AFN- ϵ $E = (Q_E, \Sigma, \delta_E, q_0, F_E)$, encontrar un AFD $D=(Q_D, \Sigma, \delta_D, q_D, F_D)$ que acepte el mismo lenguaje
 - Q_D es el conjunto de subconjuntos de Q_E
 - $q_D = \text{CLAUS}_\epsilon (q_0)$
 - $F_D = \{S \mid S \text{ pertenece a } Q_D \text{ y } S \cap F_E \neq \emptyset\}$
 - $\delta_D(S, a): S = \{p_1, p_2, \dots, p_k\}$

$$\bigcup_{i=1}^k \delta(p_i, a) = \{r_1, r_2, \dots, r_m\}$$

$$\delta_D(S, a) = \bigcup_{j=1}^m \text{CLAUS} \in (r_j)$$

Eliminación de transiciones ϵ

- Ejemplo: construir el AFD equivalente al de la figura:



	Punto	Signo	Núm.
$\rightarrow A$	B	C	D
B			E
C	B		D
D	F		D
*E			E
*F			E

$A = \{q_0, q_1\}$, $B = \{q_2\}$, $C = \{q_1\}$,
 $D = \{q_1, q_4\}$, $E = \{q_3, q_5\}$, $F = \{q_2, q_3, q_5\}$

- Teorema: un lenguaje L es aceptado por algún AFN- ϵ $E=(Q_E, \Sigma, \delta_E, q_0, F_E)$ si y sólo si es aceptado por algún AFD $D=(Q_D, \Sigma, \delta_D, q_D, F_D)$

Problemas

Dados los siguientes AFN- ε , calcular la clausura respecto de ε para cada estado y convertir los autómatas en AFD

	ε	a	b	c
->p	\emptyset	{p}	{q}	{r}
q	{p}	{q}	{r}	\emptyset
*r	{q}	{r}	\emptyset	{p}

	ε	a	b	c
->p	{q, r}	\emptyset	{q}	{r}
q	\emptyset	{p}	{r}	{p, q}
*r	\emptyset	\emptyset	\emptyset	\emptyset

Problemas

Dados los siguientes AFN- ϵ , calcular la clausura respecto de ϵ para cada estado y convertir los autómatas en AFD

	ϵ	a	b	c
->p	\emptyset	{p}	{q}	{r}
q	{p}	{q}	{r}	\emptyset
*r	{q}	{r}	\emptyset	{p}



	a	b	c
->A	A	B	C
B	B	C	C
*C	C	C	C

$$\begin{aligned} A &= \{p\} \\ B &= \{p, q\} \\ C &= \{p, q, r\} \end{aligned}$$

	ϵ	a	b	c
->p	{q, r}	\emptyset	{q}	{r}
q	\emptyset	{p}	{r}	{p, q}
*r	\emptyset	\emptyset	\emptyset	\emptyset



	a	b	c
->*A	A	B	A
*B	A	C	A
*C	D	D	D
D	D	D	D

$$\begin{aligned} A &= \{p, q, r\} \\ B &= \{q, r\} \\ C &= \{r\} \\ D &= \{\emptyset\} \end{aligned}$$

Equivalencia de estados

- Dos estados p y q de un AFD son equivalentes si para toda cadena de entrada w , $\hat{\delta}(p, w)$ es un estado de aceptación si y sólo si $\hat{\delta}(q, w)$ es un estado de aceptación
 - En caso contrario, los estados serán distinguibles
- Teorema: la equivalencia de estados es transitiva. Es decir, si para un AFD dos estados p y q son equivalentes y q y r son equivalentes, entonces p y r son equivalentes
- Teorema: si para cada estado q de un AFD se crea un bloque que contiene a q y a todos los estados equivalentes a q , los bloques de estados formarán una partición del conjunto de estados.
 - cada estado estará en un único bloque
 - todos los miembros de un bloque son equivalentes
 - dos estados de bloques diferentes no pueden ser equivalentes

Equivalencia de estados

Conjunto cociente, Q/E : partición del conjunto de estados en clases. Cada clase contiene estados equivalentes entre sí

Función Conjunto-Cociente (A): Q_E

A : entrada a la función, AF definido por (Σ, Q, f, q_0, F)

Q_E : salida de la función, conjunto cociente del AF

$Q/E_0 := \{F, \bar{F}\}$;

i:=-1;

Repetir

i:=i+1;

Si pE_iq Y

$\forall a \in \Sigma, f(p, a) \in c_k, f(q, a) \in c_k, (c_k \in Q/E_i)$

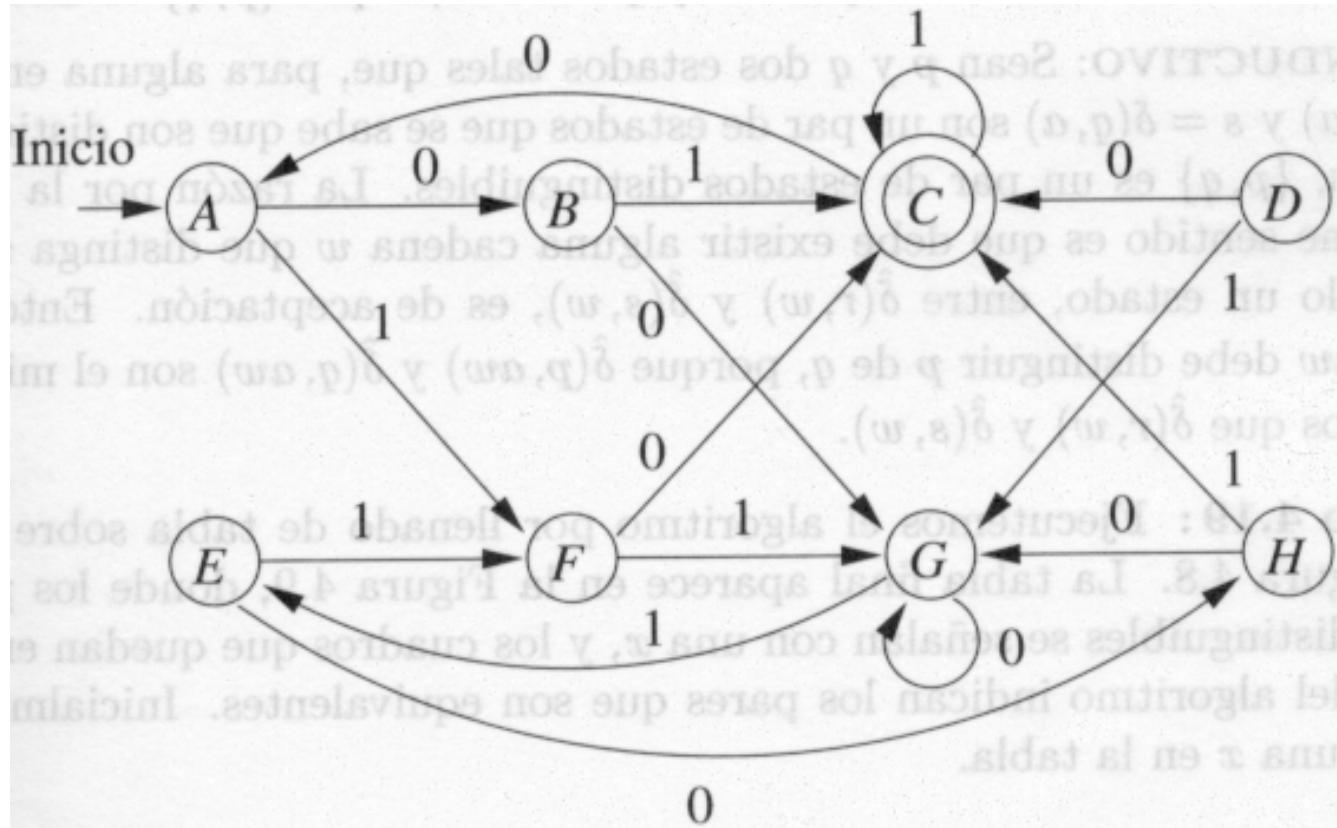
Entonces $pE_{i+1}q$

Hasta que $Q/E_{i+1} = Q/E_i$

Devolver $Q/E = Q/E_i$

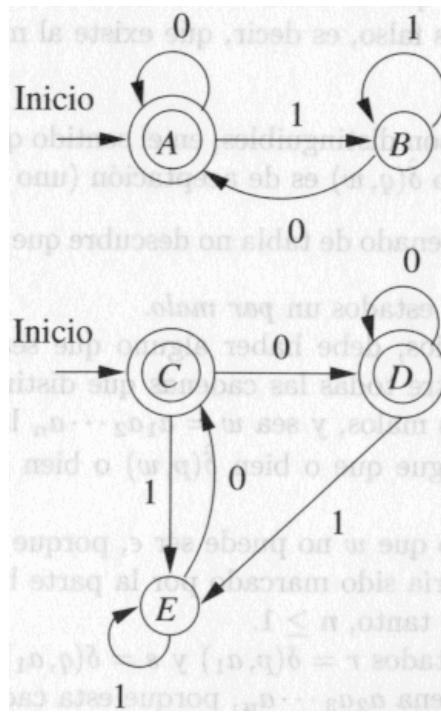
Equivalencia de estados

En el ejemplo de la figura, E es equivalente a A, B es equivalente a H y F es equivalente a D



Equivalencia de lenguajes regulares

- Sean L y M dos lenguajes representados de alguna forma
 - se convierten las representaciones en AFD
 - se comprueba si los estados iniciales de ambos AFD son equivalentes
 - si son equivalentes, entonces $L = M$ y si no, serán diferentes
- Ejemplo:



Minimización de un AFD

Para todo AFD es posible encontrar un AFD equivalente con igual o menor número de estados que cualquier AFD que acepte el mismo lenguaje. Dicho AFD es único.

1. Se eliminan los estados no accesibles desde el inicial
2. Se divide el conjunto de estados Q en bloques de estados mutuamente equivalentes
3. Se construye el AFD mínimo B equivalente a A , utilizando los bloques de estados resultantes como estados del nuevo AFD
 1. La función de transición de B es $\gamma(S, a) = T$, donde S y T son bloques de estados. Cualquier estado del bloque S debe llevar con entrada a a un estado del bloque T . Si no fuese así, esos estados serían distinguibles, lo que no puede ser cierto por construcción de B
 2. el estado inicial de B es el bloque [único] que contiene el estado inicial de A
 3. el conjunto de estados de aceptación de B es el conjunto de bloques que contienen los estados de aceptación de A

Minimización de un AFD

Función Autómata-Mínimo (A): AFD_m

A : entrada a la función, AF definido por (Σ, Q, f, q_0, F)

AFD_m : salida de la función, autómata mínimo equivalente a A .

Eliminar de A todos los estados inaccesibles desde q_0 ;

Construir el $AFD_m = (\Sigma, Q', f', q'_0, F')$ donde

$Q' = \text{Conjunto-Cociente}(A);$

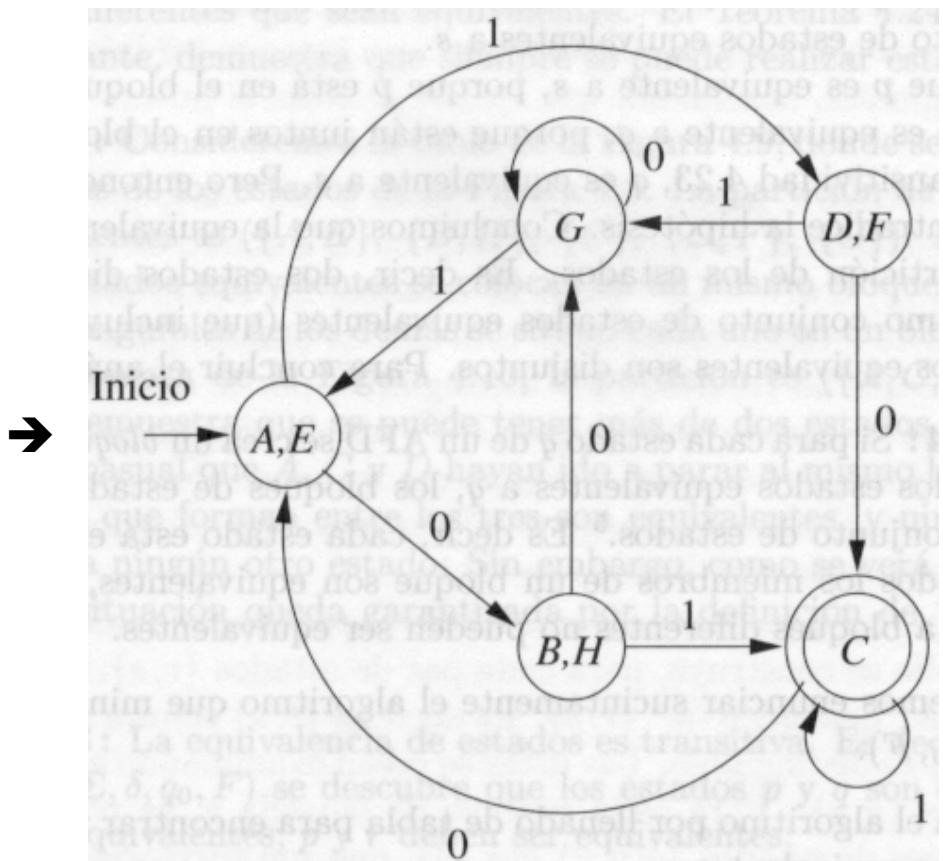
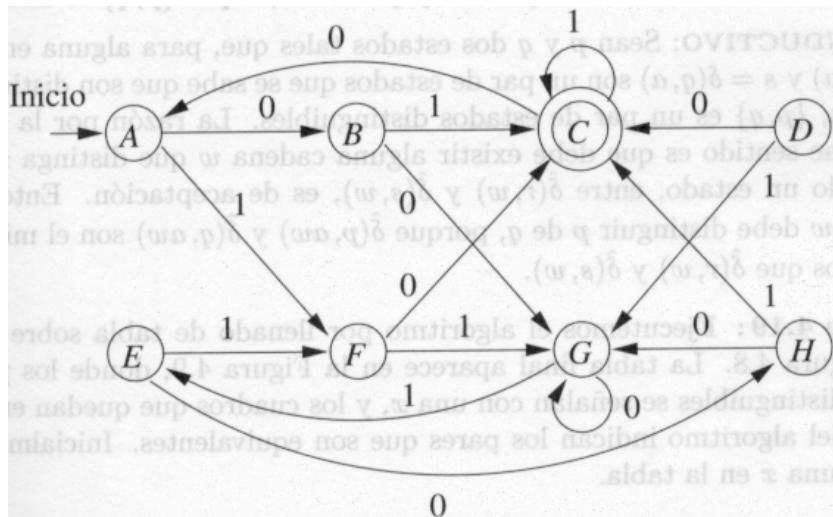
$q'_0 = c_0$ si $q_0 \in c_0, c_0 \in Q'$;

$F' = \{c \mid \exists q \in c, q \in F\}$; y

$\forall a \in \Sigma, f'(c_i, a) = c_j$ si $\exists p \in c_j, q \in c_i \mid f(q, a) = p$

Problema

- Minimizar el siguiente AFD



Problemas

Dados los siguientes AFD, completar los AFD equivalentes mínimos

	0	1
->A	B	A
B	A	C
C	D	F
*D	D	A
E	D	F
F	G	E
G	F	G
H	G	D

H es no accesible
A={A, G}
B={B, F}
C={C, E}
D= {D}



	0	1
->A	B	
B	A	C
C	D	
*D		A

	0	1
->A	B	E
B	C	F
*C	D	H
D	E	H
E	F	I
*F	G	B
G	H	B
H	I	C
*I	A	E

A={A, D, G}
B={B, E, H}
C={C, F, I}



	0	1
->A		B
B	C	
*C		B

Lenguajes regulares

Curso 2019-20



Senén Barro Ameneiro, CiTIUS

@SenenBarro

Material elaborado fundamentalmente por
el profesor Manuel Mucientes Molina

Bibliografía

- J.E. Hopcroft, R. Motwani y J.D. Ullman, "Teoría de Autómatas, Lenguajes y Computación", Addison Wesley, 2008.
 - Capítulos 3 y 4
- P. Linz, "An Introduction to Formal Languages and Automata", Jones and Bartlett Publishers, Inc., 2001.
 - Capítulo 3 y 4

Expresiones regulares (ER)

- Descripción algebraica de los lenguajes regulares
- Forma declarativa de expresar las cadenas que queremos aceptar
- Se usan como lenguaje de entrada en muchos sistemas de proceso de cadenas
 - Especificación de caracteres en el comando *grep* de UNIX
 - Diseño de analizadores léxicos mediante *Lex* o *Flex*
 - Acepta expresiones regulares (formas de las unidades sintácticas) y produce un AFD que reconoce la unidad sintáctica que aparece a continuación en la entrada
 - Diseño de verificadores del formato del texto en formularios web (JavaScript, Perl, etc.)

Operadores de las ER

- Las expresiones regulares denotan lenguajes: $01^* + 10^*$
- Unión de dos lenguajes L y M , $L \cup M$: conjunto de cadenas que pertenecen a L , a M o a ambos
- Concatenación de dos lenguajes L y M , $L.M$ (o LM): conjunto de cadenas formadas por la concatenación de una cadena de L y otra de M
- Clausura, estrella o clausura de Kleene de un lenguaje L , L^* : conjunto de cadenas formado por la concatenación de cualquier número de cadenas de L (se admiten repeticiones). Formalmente:

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

- Sólo existen dos lenguajes con clausura no infinita:
 - $L = \{\emptyset\}$. $L^* = \{\epsilon\}$, ya que $\emptyset^0 = \{\epsilon\}$ y $\emptyset^i = \emptyset$ para $i > 0$
 - $L = \{\epsilon\}$. $L^* = \{\epsilon\}$

Precedencia de los operadores

- El operador * tiene la mayor precedencia
- El operador de concatenación (.) es el segundo en orden de precedencia
- Finalmente se aplican los operadores de unión con sus operandos
- Utilizando los paréntesis se modifican las reglas de precedencia

Álgebra de ER

- Propiedad conmutativa de la unión: $L + M = M + L$
- Propiedad asociativa de la unión: $(L + M) + N = L + (M + N)$
- Propiedad asociativa de la concatenación: $(LM)N = L(MN)$
 - la concatenación no es conmutativa: $LM \neq ML$
- \emptyset es el elemento identidad de la unión: $\emptyset + L = L + \emptyset = L$
- ε es el elemento identidad de la concatenación: $\varepsilon L = L\varepsilon = L$
- \emptyset es el elemento nulo de la concatenación: $\emptyset L = L\emptyset = \emptyset$
- Propiedad distributiva por la izquierda de la concatenación respecto de la unión: $L(M + N) = LM + LN$
- Propiedad distributiva por la derecha de la concatenación respecto de la unión: $(M + N)L = ML + NL$

Álgebra de ER

- Operador idempotente: el resultado de aplicarlo a dos valores iguales es el mismo valor
 - Propiedad de idempotencia de la unión: $L + L = L$
- Propiedades relativas a las clausuras
 - $(L^*)^* = L^*$
 - $\emptyset^* = \epsilon$
 - $\epsilon^* = \epsilon$
 - $L^+ = LL^* = L^*L$
 - $L^* = L^+ + \epsilon$
 - $L? = L + \epsilon$

Construcción de ER

- Base:
 - Las constantes ε y \emptyset son ER. $L(\varepsilon) = \{\varepsilon\}$ y $L(\emptyset) = \{\emptyset\}$
 - Si a es un símbolo, a es la ER del lenguaje $L(a) = \{a\}$
- Paso inductivo:
 - si E y F son ER, $E + F$ es una ER, y $L(E + F) = L(E) \cup L(F)$
 - si E y F son ER, EF es una ER, y $L(EF) = L(E)L(F)$
 - si E es ER, E^* es una ER, y $L(E^*) = (L(E))^*$
 - si E es ER, (E) es una ER, y $L((E)) = L(E)$
- Ejemplo: escribir la ER para el conjunto de cadenas que constan de ceros y unos alternos

Problemas

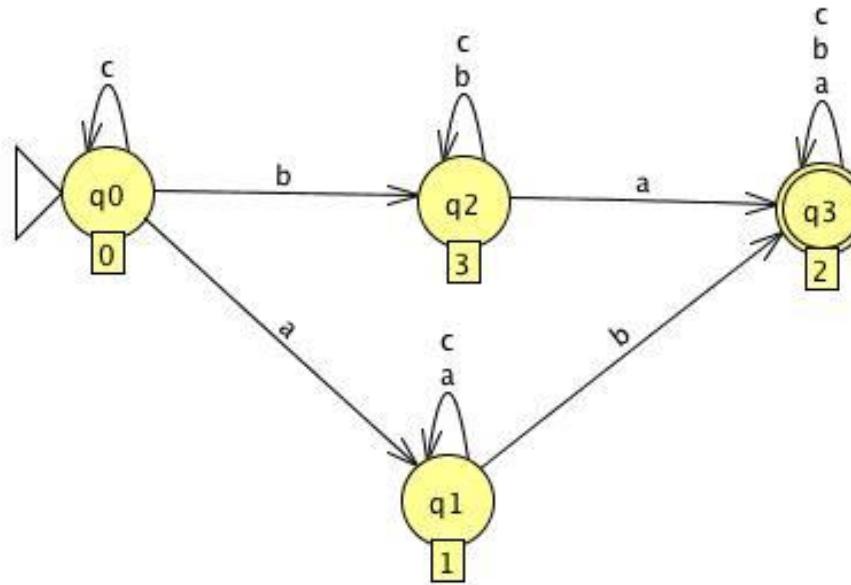
Escribir las ER para los siguientes lenguajes:

1. El conjunto de cadenas del alfabeto $\{a, b, c\}$ que contengan al menos una a y una b.
2. El conjunto de cadenas del alfabeto $\{0, 1\}$ cuyo cuarto símbolo empezando por la izquierda sea un 1.
3. El conjunto de cadenas del alfabeto $\{0, 1\}$, tales que todos los pares de ceros adyacentes aparecen antes que cualquier par de unos adyacentes.

Problemas

El conjunto de cadenas del alfabeto $\{a, b, c\}$ que contengan al menos una a y una b.

Solución:



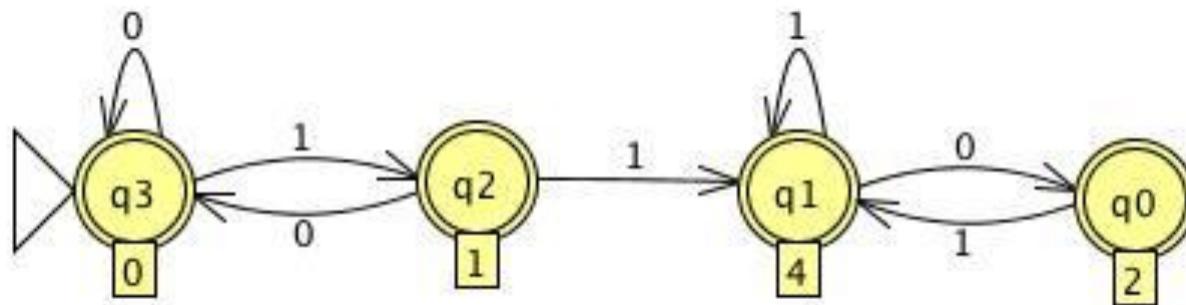
$$c^*a(c+a)^*b(a+b+c)^* + c^*b(c+b)^*a(a+b+c)^*$$

$$c^*(b(c+b)^*a + a(c+a)^*b)(a+b+c)^*$$

Problemas

Escribir las ER para el conjunto de cadenas del alfabeto $\{0, 1\}$, tales que todos los pares de ceros adyacentes aparecen antes que cualquier par de unos adyacentes.

Solución, diseñando primero el AFD:



$$(0+10)^* + (0+10)^*1 + (0+10)^*11(1+01)^* + (0+10)^*11(1+01)^*0$$

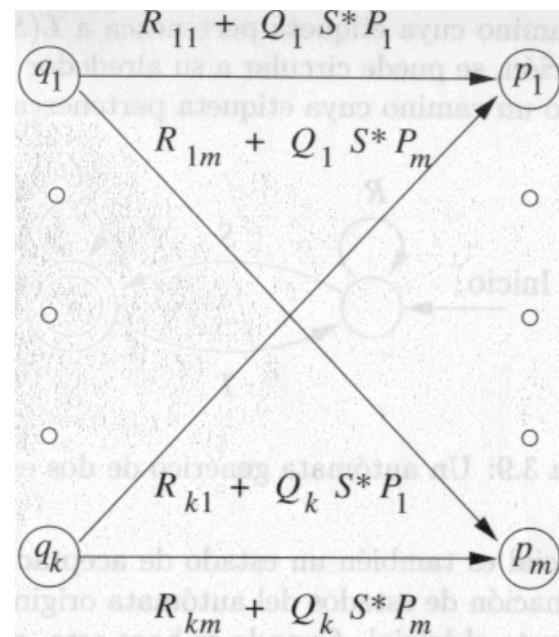
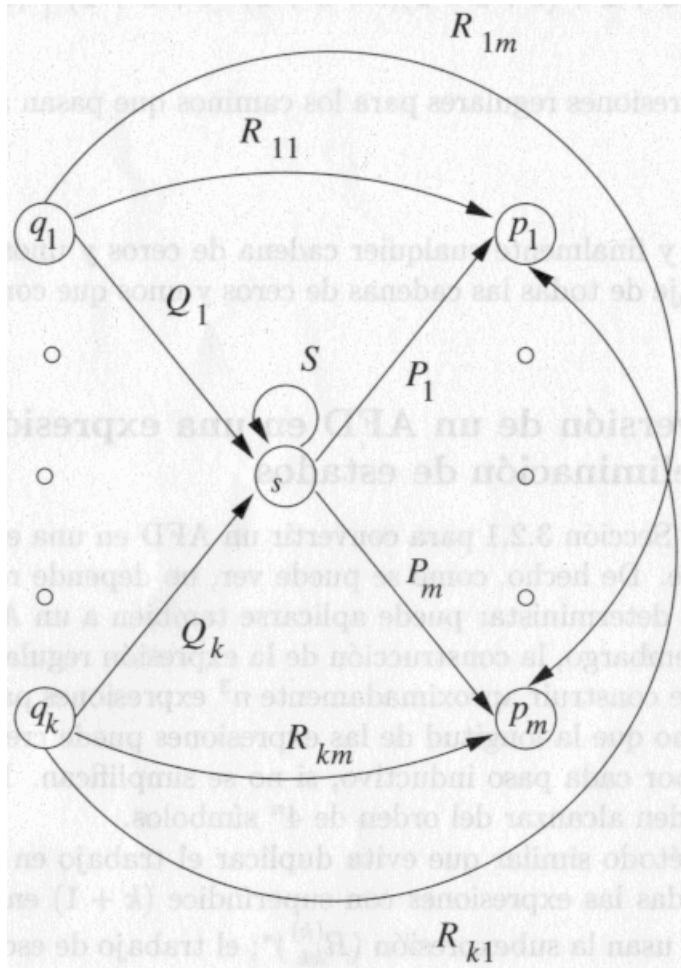
Autómatas finitos y ER

Las ER definen los lenguajes regulares exactamente igual que los autómatas finitos:

- Todo lenguaje definido por un AF (AFD, AFN, AFN- ϵ) también puede definirse mediante una ER
- Todo lenguaje definido por una ER puede definirse mediante un AF

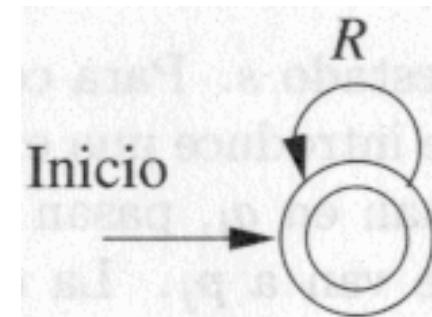
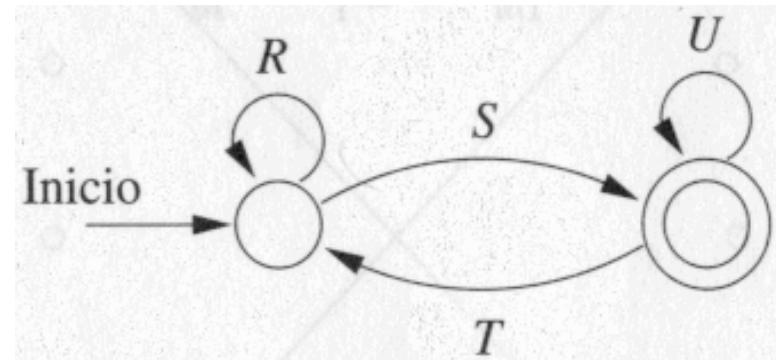
Autómatas finitos a ER

Eliminación de estados: en los arcos aparecen ER



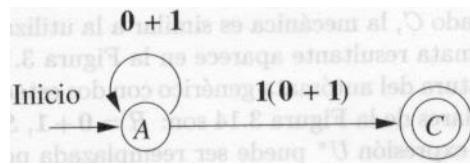
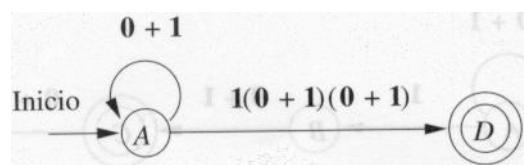
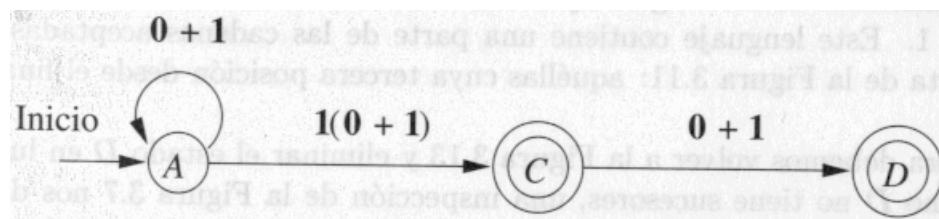
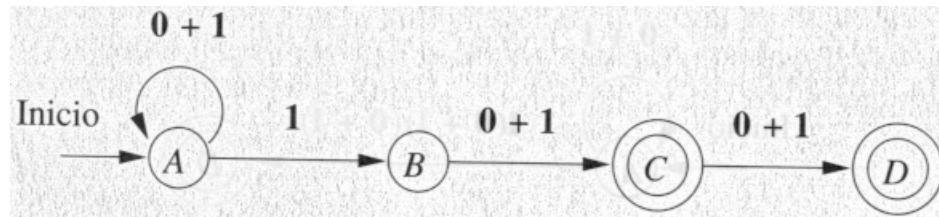
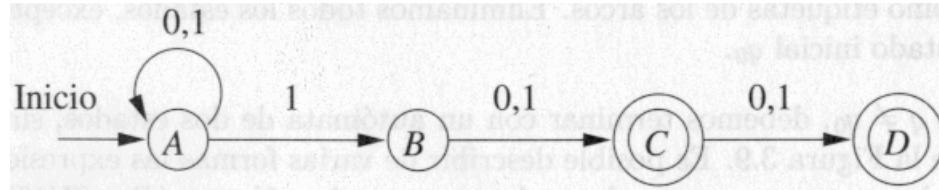
Autómatas finitos a ER

- Para cada estado de aceptación q , se aplica el proceso de reducción. Se eliminan todos los estados excepto q y el estado inicial q_0
- Si $q \neq q_0$, $L = (R^* + SU^*T)^*SU^*$
- Si el estado inicial es estado de aceptación, habrá que eliminar todos los estados excepto el inicial:
$$L = R^*$$
- La ER deseada es la unión de las cadenas obtenidas del autómata para cada estado de aceptación



Autómatas finitos a ER

Ejemplo:



Problemas

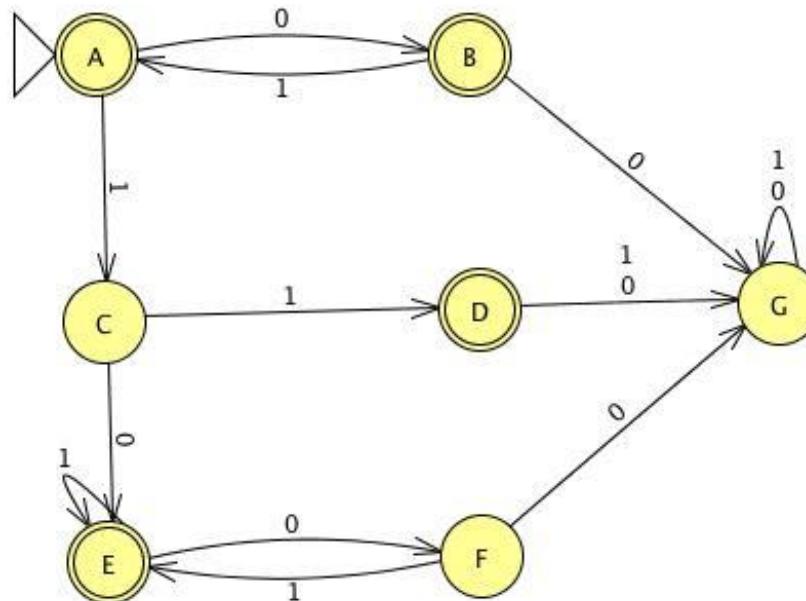
Obtener la expresión regular del AFD definido en la siguiente tabla de transiciones:

	0	1
->*A	B	C
*B	G	A
C	E	D
*D	G	G
*E	F	E
F	G	E
G	G	G

Problemas

Primero obtendremos el AFD asociado y de ahí es fácil obtener la ER

Solución:

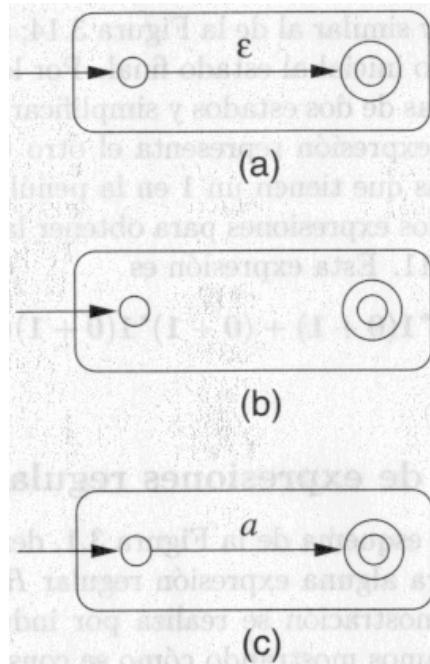


$$(01)^* + (01)^*0 + (01)^*11 + (01)^*10(1+01)^*$$

Conversión de ER en autómatas

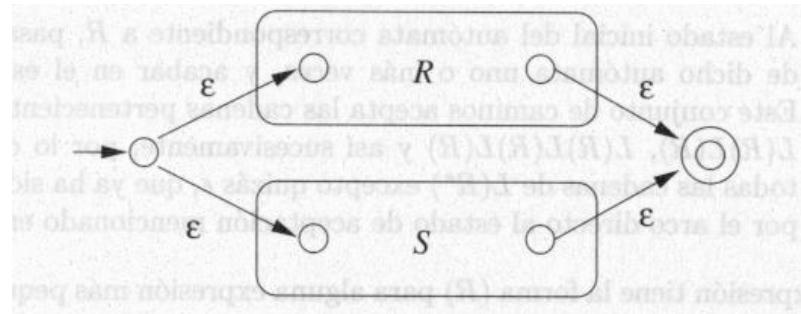
- Teorema: todo lenguaje definido por una ER puede definirse también mediante un AF
- Prueba: sea $L = L(R)$ para la ER R . Se demostrará que $L = L(E)$ para algún AFN- ϵ E

- Base:

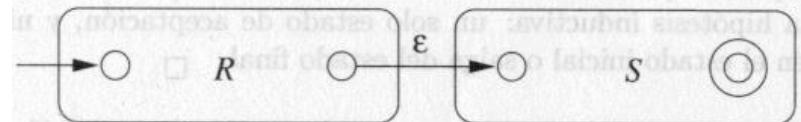


Conversión de ER en autómatas

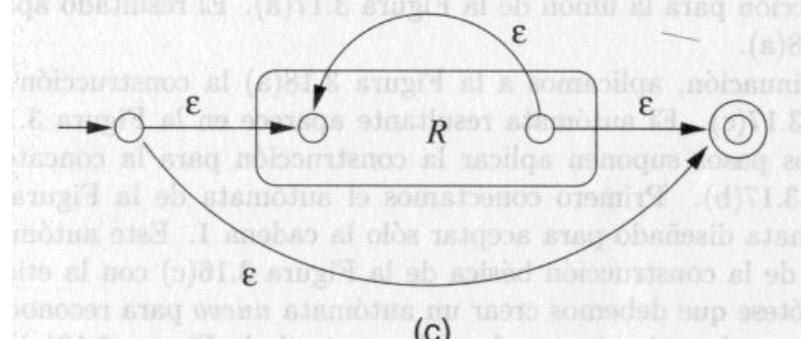
- Paso inductivo:
 - $R + S: L(R) \cup L(S)$
 - $RS: L(R)L(S)$
 - $R^*: L(R^*)$



(a)



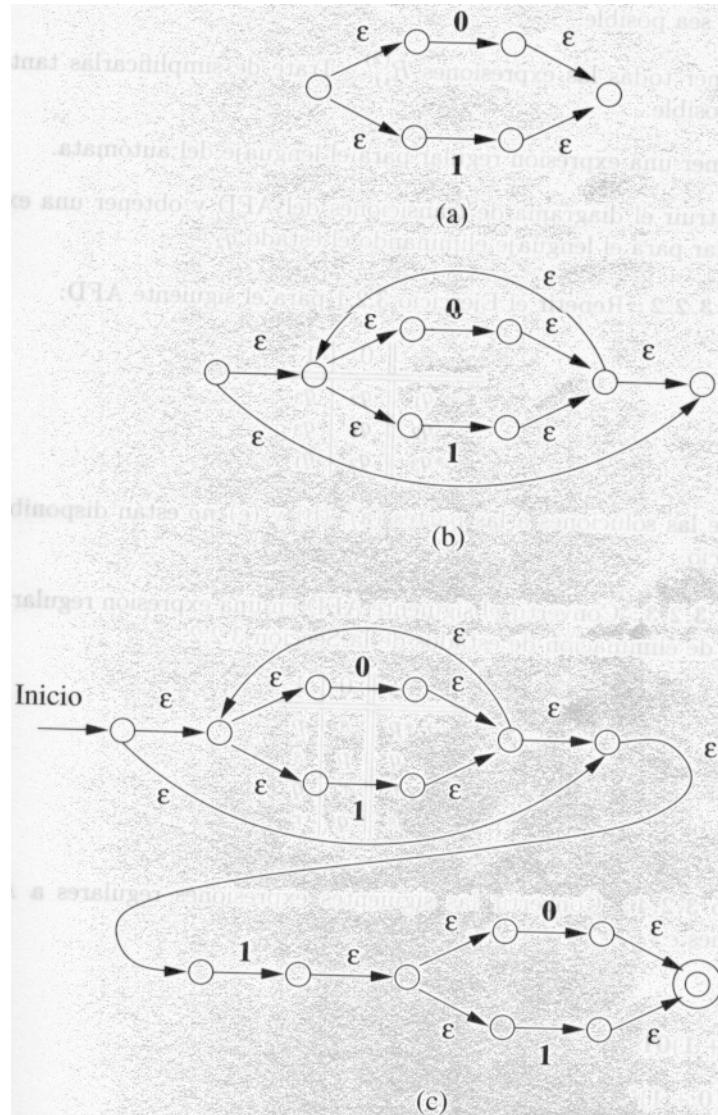
(b)



(c)

Conversión de ER en autómatas

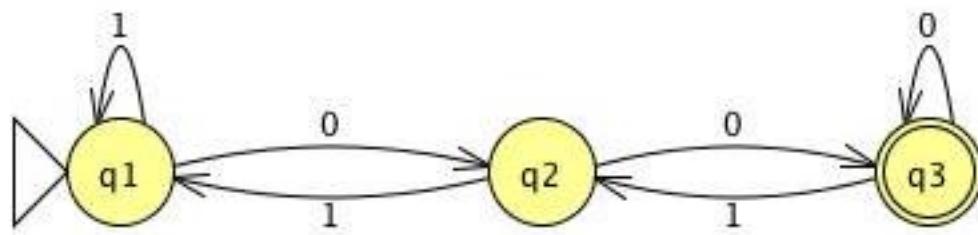
Ejemplo: $(0 + 1)^*1(0 + 1)$



Problemas

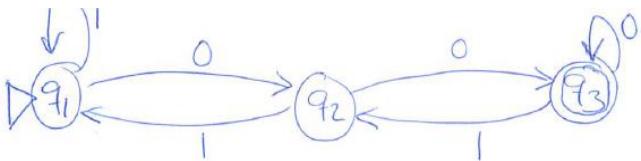
- Dado el siguiente AFD, obtener la ER para el lenguaje del autómata

	0	1
->q ₁	q ₂	q ₁
q ₂	q ₃	q ₁
*q ₃	q ₃	q ₂



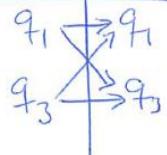
- Convertir las siguientes ER en AFN

- 01^*
- $0^* 1^*$



Eliminamos q_2

Hacemos q_2 desde q_2



Escribimos las transiciones

$$\overline{q_1 q_1}, \overline{q_1 q_3}, \overline{q_3 q_1}, \overline{q_3 q_3}$$

$$\overline{q_1 q_1} = 1 + 0\phi^* 1 = 1 + 01$$

indica que en q_2 no hay
redistribución a si misma

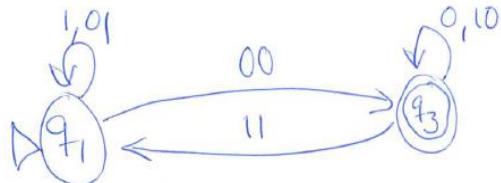
$$\overline{q_1 q_3} = \phi + 0\phi^* 0 = 00$$

no hay transición directa entre q_1 y q_3

$$\overline{q_3 q_1} = \phi + 1\phi^* 1 = 11$$

$$\overline{q_3 q_3} = 0 + 1\phi^* 0 = 0 + 10$$

Luego:



La expresión regular será:

$$((1+01) + 00(0+10)^* 11)^* 00(0+10)^*$$

Recordemos:



$$(R+SU^*T) SU^*$$

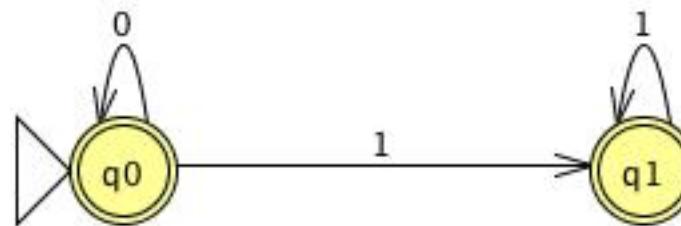
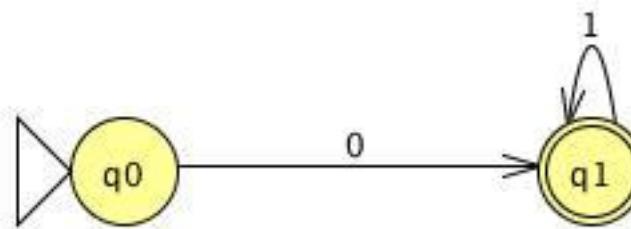
Problemas

Solución al primer ejercicio:

$$((1+01)+00(0+10)*11)*00(0+10)*$$

Solución a la conversión de las siguientes ER en AFN

- 01^*
- $0^* 1^*$



Aplicaciones de las ER

- Búsqueda de patrones de texto mediante ER que dan una “imagen” del patrón que se quiere reconocer
- Aplicaciones:
 - analizadores léxicos
 - búsqueda de textos
 - Software de verificación del formato del texto en formularios web (asignatura de Desarrollo de Aplicaciones Web, 3^{er} curso, 1^{er} cuatrimestre)

Búsqueda de patrones en textos

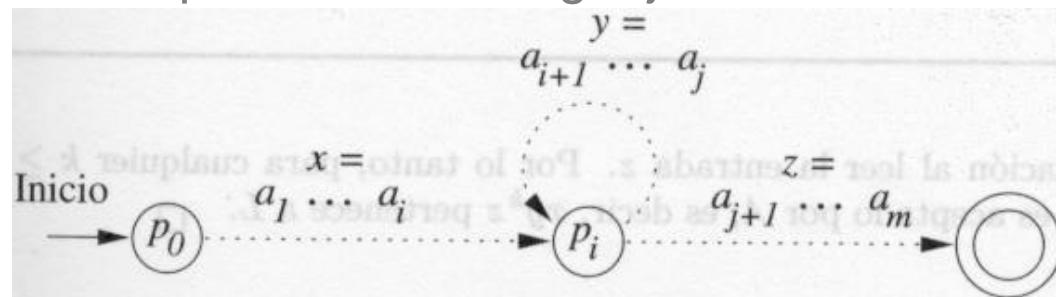
- Búsqueda eficiente de palabras en un gran repositorio de texto, como la Web
- La notación de las ER es valiosa para describir patrones de búsqueda interesantes
- Posibilidad de pasar de ER a una implementación eficiente (autómatas)
- Descripción de patrones de texto vagamente definidos: resulta útil emplear ER, ya que es posible modificarlas con poco esfuerzo
- Ejemplo: direcciones de calles en páginas web
 - $(\text{Calle}|\text{c/}|\text{Avenida}|\text{Avda}\.|\text{Plaza}|\text{Pza}\.) \ [A-Z][a-z]^* (\ [A-Z][a-z]^*)^* [0-9]^+[A-Z]?$

Propiedades de los lenguajes regulares

- Descripción de los lenguajes regulares
 - AFD
 - AFN
 - AFN- ϵ
 - Expresiones regulares
- No todos los lenguajes son regulares
 - $L_{01} = \{0^n1^n \mid n \geq 1\}$
 - si fuese regular existiría un AF con k estados que lo reconocería

Lema del bombeo para lenguajes regulares

- Para un lenguaje regular [infinito], el cumplimiento del lema de bombeo (LB) es una condición **necesaria**, pero **no suficiente**
- Teorema: sea L un lenguaje regular. Entonces existe una constante n (que depende de L), tal que, para toda cadena w perteneciente a L , con $|w| \geq n$, podemos dividir w en tres cadenas, $w = xyz$, de modo que:
 - $y \neq \epsilon$
 - $|xy| \leq n$
 - para todo $k \geq 0$, la cadena xy^kz también pertenece a L
- Siempre es posible encontrar una cadena no vacía y , no demasiado lejos del comienzo de w , que se puede “bombar”
- Si se repite y cualquier número de veces o se borra ($k = 0$), la cadena resultante también pertenece al lenguaje L



Aplicación del lema del bombeo

1. Elegimos un lenguaje L para el que tratamos de demostrar que no es regular
2. El valor de n es desconocido, por lo que debemos considerar cualquier posible valor
3. Elegimos w (podemos usar n como parámetro)
 - Si para un n suficientemente grande no podemos escoger w , L será regular
4. Repetir para todas las descomposiciones
 1. Escoger una descomposición de w en xyz , sujeta a las restricciones:
 1. $y \neq \varepsilon$
 2. $|xy| \leq n$
 2. Si xy^kz pertenece a L para todo valor de k
 1. Se verifica el LB
 2. No se puede afirmar que el lenguaje sea regular
 3. No es necesario probar con otras descomposiciones (finaliza el algoritmo)
5. Si 4.2 no se ha cumplido para ninguna descomposición, no se verifica el LB y, por tanto, el lenguaje no es regular

Problemas

Demostrar si se verifica el lema de bombeo para el lenguaje descrito por la siguiente ER: 0^m1^m

Demostración:

Recordemos que debe existir una constante n tal que, para toda cadena w perteneciente al lenguaje, con $|w| \geq n$, podamos dividir w en tres cadenas, $w = xyz$, de modo que:

1. $y \neq \varepsilon$
2. $|xy| \leq n$
3. para todo $k \geq 0$, la cadena xy^kz también pertenece al lenguaje

Si tomamos $n=5$ y $w=0000011111$, podremos considerar: $x=0000$; $y=0$; $z=11111$, verificándose los puntos 1 y 2 previos, pero no el punto 3, ya que si $k=0$, por ejemplo, resultará $xy^0z=000011111$, que no pertenece al lenguaje.

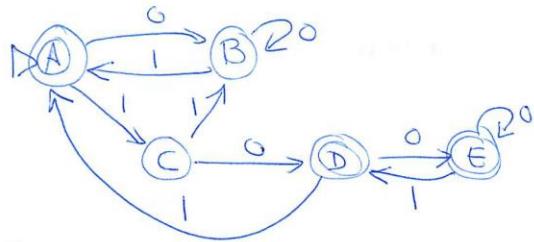
Problemas

Dados los siguientes lenguajes:

1. Razonar si son regulares, es decir, si es posible reconocerlos con un autómata finito.

Lenguajes:

- a. El que consta de todas las cadenas con el mismo número de 0 y 1 (sin ningún orden particular)
- b. El conjunto vacío
- c. $\{00, 11\}$
- d. $(00 + 11)^*$
- e. $L = \{h \in \{a, b\}^* : N_a(h) < N_b(h)\}$
- f. $L = \{a^i b^j c^k / k=i-j; i, j, k \geq 0\}$

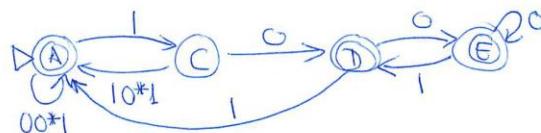


Eliminando B

	$\xrightarrow{B} \xleftarrow{B}$
A	
C	

$$\overline{AA} = 00^*1$$

$$\overline{CA} = 10^*1$$

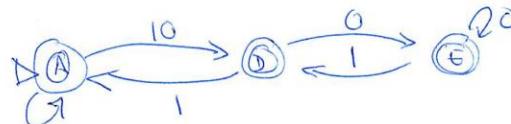


Eliminando C

	$\xrightarrow{C} \xleftarrow{C}$
A	
D	

$$\overline{AB} = 00^*1 + 110^*1$$

$$\overline{AD} = 10$$

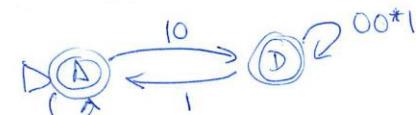


$$00^*1 + 110^*1$$

Eliminando E

	$\xrightarrow{E} \xleftarrow{E}$
D	

$$\overline{DD} = 00^*1$$



$$00^*1 + 110^*1$$

$$L_D = \{(00^*1 + 110^*1) + 10(00^*1)^*\}^* 10 (00^*1)^*$$

Eliminando D

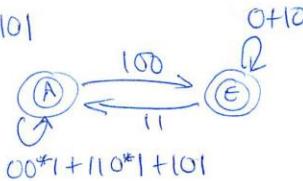
	$\xrightarrow{D} \xleftarrow{D}$
A	
E	

$$\overline{AA} = 00^*1 + 110^*1 + 101$$

$$\overline{AE} = 100$$

$$\overline{EA} = 11$$

$$\overline{EE} = 0+10$$



$$00^*1 + 110^*1 + 101$$

$$L_E = \{(00^*1 + 110^*1 + 101) + 100(0+10)^*\}^* 11 \{^*. 100(0+10)^*\}$$

Gramáticas independientes del contexto



Senén Barro Ameneiro, CiTIUS

@SenenBarro

Material elaborado fundamentalmente por
el profesor Manuel Mucientes Molina

Bibliografía

- J.E. Hopcroft, R. Motwani y J.D. Ullman.
"Teoría de Autómatas, Lenguajes y
Computación", Addison Wesley, 2008.
 - Capítulo 5
- P. Linz, "An Introduction to Formal Languages
and Automata", Jones and Bartlett Publishers,
Inc., 2001.
 - Capítulo 5

GIC: lo que vamos a ver

- Lenguajes independientes del contexto (LIC)
- Gramáticas independientes del contexto (GIC)
 - implementación de analizadores sintácticos
 - descripción de formatos de documentos: XML
- Árboles sintácticos: representan gráficamente la estructura de la gramática
- Autómatas con pila

Definición de GIC

Las GIC están formadas por cuatro componentes:

- el conjunto finito de **símbolos no terminales (V)** o variables, que permiten representar subconjuntos del lenguaje o estados intermedios en la generación de las palabras del lenguaje
- el alfabeto de **símbolos terminales (T)**, que son los símbolos finales del lenguaje
- un conjunto finito de **producciones o reglas (P)**, que indican las transformaciones posibles desde los símbolos no terminales a las palabras del lenguaje
- el **símbolo inicial o axioma (S)** de la gramática (una de las variables), a partir del cual se obtiene cualquier palabra del lenguaje

Definición de GIC

Las **reglas de la** gramática están formadas por:

- una variable, cabeza de la producción,
- el símbolo de producción \rightarrow ,
- una cadena de cero o más símbolos terminales y no terminales, que son el cuerpo de la producción

Es decir, las producciones son de la forma $B \rightarrow x$, con $B \in V$ y $x \in (V \cup T)^*$

$$G = (V, T, P, S)$$

- ejemplo: $G_{\text{palíndromo}} = (\{S\}, \{0, 1\}, P, S)$, donde P son las producciones o reglas:

$$S \rightarrow \epsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$$

Distintas gramáticas

Las **distintas gramáticas** admiten **formas distintas para las producciones**:

Tipo 0, no restringida o recursivamente enumerables

$$\begin{array}{l} x \rightarrow y \\ x \in (NT/T)^+ \\ y \in (NT/T)^* \end{array}$$

Tipo 1, sensible al contexto

$$\begin{array}{l} \alpha \rightarrow \beta ; |\alpha| \leq |\beta| \\ \alpha = z_1 x z_2 \\ \beta = z_1 y z_2 \\ z_1, z_2 \in T^* \\ x \in NT \\ y \in (NT/T)^+ \end{array}$$

Tipo 2, libre de contexto

$$\begin{array}{l} x \rightarrow y \\ x \in NT \\ y \in (NT/T)^* \end{array}$$

Notemos que se está usando NT (No Terminal) como V

Tipo 3, regular

$$\begin{array}{l} \alpha \rightarrow \beta \\ \alpha \in NT \\ \beta \in \begin{cases} aB \\ Ba \\ b \end{cases} \\ B \in NT \\ a \in T^+ \\ b \in T^* \end{array}$$

Gramáticas regulares

Los lenguajes regulares, estudiados en capítulos anteriores, pueden asociarse a una **gramática de tipo 3 o regular**

- Gramáticas regulares:
 - lineales por la derecha
 - lineales por la izquierda
- Una gramática $G = (V, T, P, S)$ es lineal por la derecha si todas sus producciones son de la forma:
 - $A \rightarrow xB$
 - $A \rightarrow x$donde A y B pertenecen a V y x pertenece a T^*
- Una gramática $G = (V, T, P, S)$ es lineal por la izquierda si todas sus producciones son de la forma:
 - $A \rightarrow Bx$
 - $A \rightarrow x$

Ejemplo de gramática (GIC)

$G = (\{E, I\}, \{+, *, (,), a, b, 0, 1\}, P, E)$,
donde P es el conjunto de producciones:

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$

Derivaciones de una gramática

Sea $G = (V, T, P, S)$ y $aA\beta$ una cadena de terminales y no terminales (variables), donde A está en V y a y β están en $(V \cup T)^*$.

Sea $A \rightarrow \gamma$ una producción de G . Entonces:

$$\alpha A \beta \xrightarrow{G} \alpha \gamma \beta$$

Una **derivación** de una sentencia ω es la **secuencia** de sustituciones de no terminales que, partiendo del símbolo inicial S , produce como resultado ω .

Derivaciones de una gramática

Ejemplo de derivación de: “a * (a + b00)”

- Derivación más a la izquierda:

$$E \Rightarrow E * E \Rightarrow I * E \Rightarrow a * E \Rightarrow$$

$$a * (E) \Rightarrow a * (E + E) \Rightarrow a * (I + E) \Rightarrow a * (a + E) \Rightarrow$$

$$a * (a + I) \Rightarrow a * (a + I0) \Rightarrow a * (a + I00) \Rightarrow a * (a + b00)$$

- Derivación más a la derecha:

$$E \xrightarrow{md} E * E \xrightarrow{md} E * (E) \xrightarrow{md} E * (E + E) \xrightarrow{md}$$

$$E * (E + I) \xrightarrow{md} E * (E + I0) \xrightarrow{md} E * (E + I00) \xrightarrow{md} E * (E + b00) \xrightarrow{md}$$

$$E * (I + b00) \xrightarrow{md} E * (a + b00) \xrightarrow{md} I * (a + b00) \xrightarrow{md} a * (a + b00)$$

Lenguaje de una gramática

Si $G = (V, T, P, S)$ es una GIC, el **lenguaje de G** será:

$$L(G) = \{w \text{ que están en } T^* \mid S \xrightarrow[G]{*} w\}$$

- Si $G = (V, T, P, S)$ es una GIC, cualquier cadena $\alpha \in (V \cup T)^*$, tal que $S \xrightarrow[G]{*} \alpha$ es una **forma sentencial**
- El lenguaje $L(G)$ está formado por las formas sentenciales que están en T^* y se denominan **sentencias**

$$E \Rightarrow E * E \Rightarrow E * (E) \Rightarrow E * (E + E) \Rightarrow E * (I + E)$$

Ejemplos de formas sentenciales:

$$\begin{array}{ccccccc} E & \Rightarrow & E * E & \Rightarrow & I * E & \Rightarrow & a * E \\ & & mi & & mi & & mi \end{array}$$

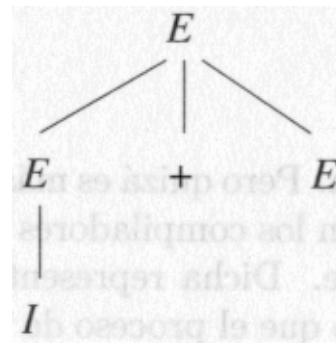
$$\begin{array}{ccccccc} E & \Rightarrow & E * E & \Rightarrow & E * (E) & \Rightarrow & E * (E + E) \\ & & md & & md & & md \end{array}$$

Árboles de derivación

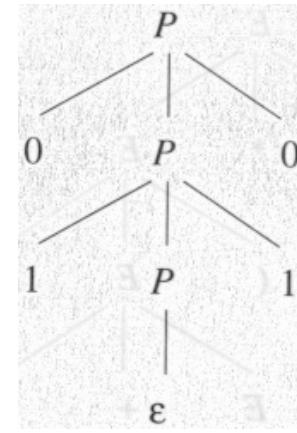
Representación de las derivaciones en forma de árbol

- Sea $G = (V, T, P, S)$. El árbol de derivación para G tendrá las siguientes características:
 - cada nodo interior está etiquetado con una variable
 - cada hoja está etiquetada con una variable, un terminal o ϵ . Si es ϵ , tiene que ser el único hijo de su nodo progenitor
 - si un nodo interior está etiquetado con A y sus hijos están etiquetados con X_1, X_2, \dots, X_k (de izquierda a derecha), entonces $A \rightarrow X_1X_2\dots X_k$ es una producción de P
- Ejemplos:

$$E \xrightarrow{*} I + E$$



$$P \xrightarrow{*} 0110$$



Resultado de un árbol de derivación

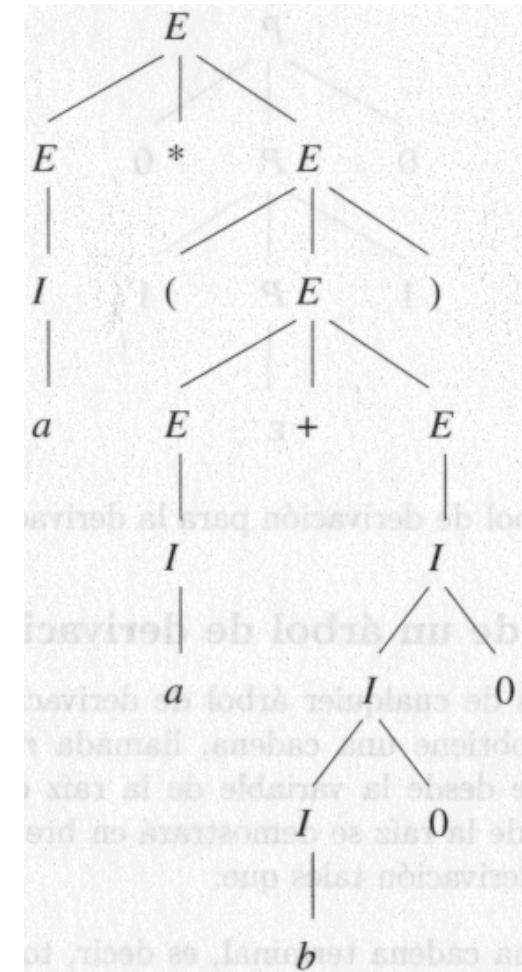
Concatenando la hojas de un árbol desde la izquierda se obtiene la cadena resultado del árbol, que se deriva desde la raíz

Cadenas del lenguaje:

- los nodos hoja son terminales
- la raíz está etiquetada con el símbolo inicial

$$E \xrightarrow{*} a^*(a+b00)$$

1. $E \rightarrow I$
2. $I \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow I0$
10. $I \rightarrow I1$



Aplicaciones de las GIC

Descripción de lenguajes
de programación

- Análisis sintáctico

Lenguajes de marcado

- HTML (HyperText
Markup Language)

1.	<i>Car</i>	\rightarrow	<i>a</i> <i>A</i> ...
2.	<i>Texto</i>	\rightarrow	ϵ <i>Car Texto</i>
3.	<i>Doc</i>	\rightarrow	ϵ <i>Elemento Doc</i>
4.	<i>Elemento</i>	\rightarrow	<i>Texto</i> $<\text{EM}>$ <i>Doc</i> $</\text{EM}>$ $<\text{P}>$ <i>Doc</i> $<\text{OL}>$ <i>Lista</i> $</\text{OL}>$...
5.	<i>ListItem</i>	\rightarrow	$<\text{LI}>$ <i>Doc</i>
6.	<i>Lista</i>	\rightarrow	ϵ <i>ListItem Lista</i>

Figura 5.13. Parte de una gramática de HTML.

Aplicaciones de las GIC

Lenguajes de marcado: XML (eXtensible Markup Language)

- El formato se especifica con un DTD (Document Type Definition)
 - Uso de GIC para describir las etiquetas permitidas y su forma de anidarse

```
<!DOCTYPE PcSpecs [  
  <!ELEMENT PCS (PC*)>  
  <!ELEMENT PC (MODELO, PRECIO, PROCESADOR, RAM, DISCO+)>  
  <!ELEMENT MODELO (#PCDATA)>  
  <!ELEMENT PRECIO (#PCDATA)>  
  <!ELEMENT PROCESADOR (FABRICANTE, MODELO, VELOCIDAD)>  
  <!ELEMENT FABRICANTE (#PCDATA)>  
  <!ELEMENT MODELO (#PCDATA)>  
  <!ELEMENT VELOCIDAD (#PCDATA)>  
  <!ELEMENT RAM (#PCDATA)>  
  <!ELEMENT DISCO (DISCODURO | CD | DVD)>  
  <!ELEMENT DISCODURO (FABRICANTE, MODELO, TAMAÑO)>  
  <!ELEMENT TAMAÑO (#PCDATA)>  
  <!ELEMENT CD (VELOCIDAD)>  
  <!ELEMENT DVD (VELOCIDAD)>  
>  
]
```

Figura 5.14. Una DTD para computadoras personales.

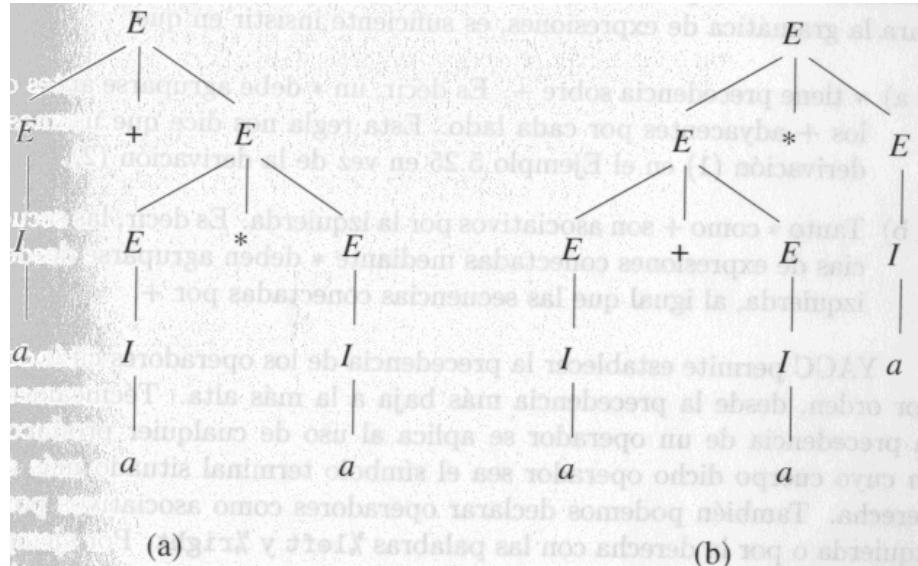
```
<PCS>  
  <PC>  
    <MODELO>4560</MODELO>  
    <PRECIO>$2295</PRECIO>  
    <PROCESADOR>  
      <FABRICANTE>Intel</FABRICANTE>  
      <MODELO>Pentium</MODELO>  
      <VELOCIDAD>800MHz</VELOCIDAD>  
    </PROCESADOR>  
    <RAM>256</RAM>  
    <DISCO><DISCODURO>  
      <FABRICANTE>Maxtor</FABRICANTE>  
      <MODELO>Diamond</MODELO>  
      <TAMAÑO>30.5Gb</TAMAÑO>  
    </DISCODURO></DISCO>  
    <DISCO><CD>  
      <VELOCIDAD>32x</VELOCIDAD>  
    </CD></DISCO>  
  </PC>  
  <PC>  
    ...  
  </PC>  
</PCS>
```

Figura 5.15. Parte de un documento que obedece a la estructura de la DTD

Ambigüedad

Una GIC $G = (V, T, P, S)$ es ambigua si existe al menos una cadena w en T^* para la que podemos encontrar dos árboles de derivación distintos con la raíz etiquetada con S y cuyo resultado es w

1. $E \rightarrow I$
2. $E \rightarrow E + E$
3. $E \rightarrow E * E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $\leftarrow I \rightarrow I0$
10. $I0 \rightarrow I1$



La existencia de derivaciones diferentes para una cadena no supone un defecto en la gramática; *la existencia de árboles de derivación diferentes sí supone un problema*

$$\begin{aligned} E &\rightarrow E + E \rightarrow I + E \rightarrow a + E \rightarrow a + I \rightarrow a + b \\ E &\rightarrow E + E \rightarrow E + I \rightarrow E + b \rightarrow I + b \rightarrow a + b \end{aligned}$$



Problemas

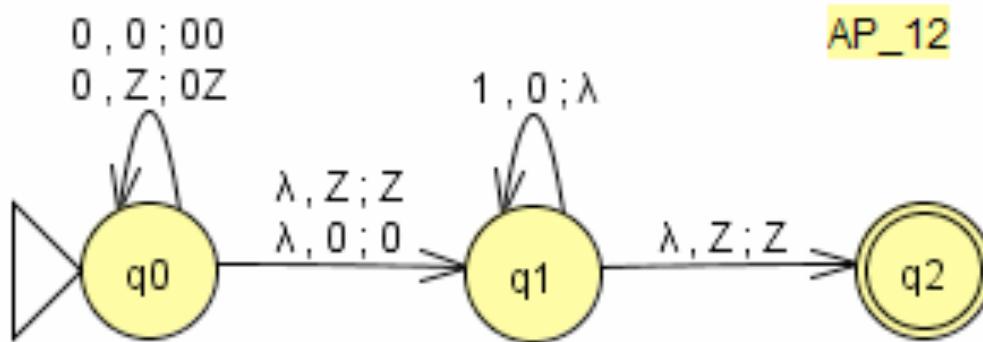
Diseñar la GIC que genere el lenguaje:

$$L = \{0^n 1^n \mid n \geq 0\}$$

$$L = \{\lambda, 01, 0011, 000111\dots\}$$

$$G = (\{S\}, \{0,1\}, P, S)$$

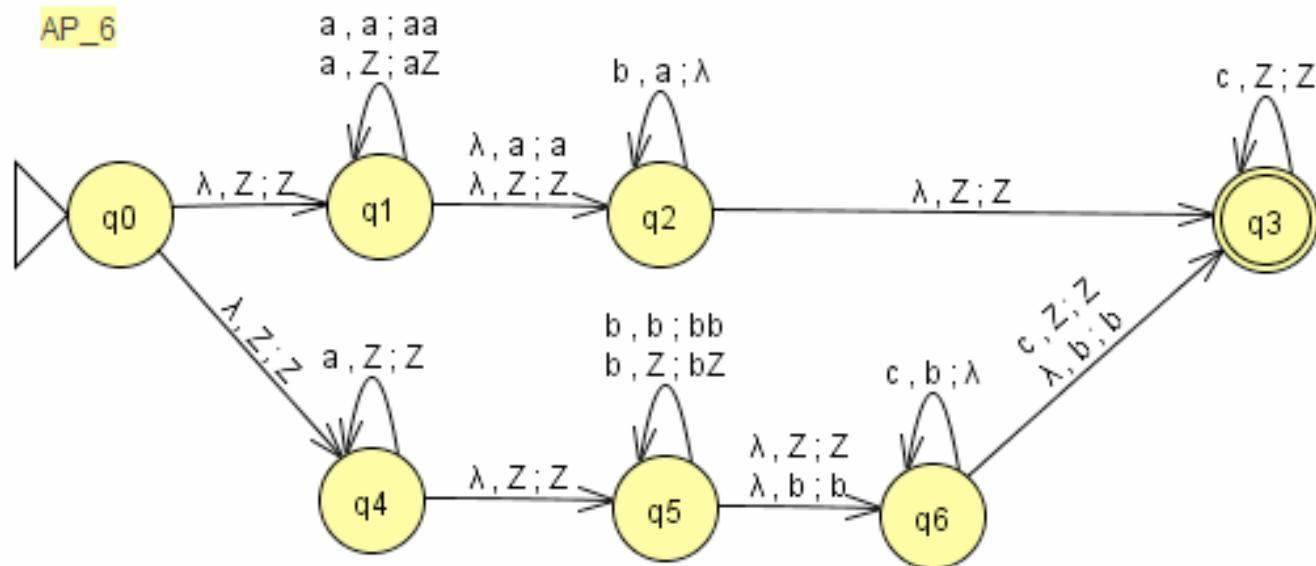
$$P = \{S \rightarrow \lambda \mid 0S1\}$$



Problemas

Diseñar la GIC que genere el lenguaje:

$$L = \{a^i b^j c^k / i = j \text{ ó } j \neq k, \text{ con } i, j, k \geq 0\}$$



Problemas

1. Diseñar la GIC que genere el lenguaje:

$$L = \{0^n 1^n \mid n \geq 1\}$$

1. Diseñar la GIC que genere el lenguaje:

$$L = \{a^i b^j \mid 2i = j; i, j > 0\}$$

1. Diseñar la GIC que genere el lenguaje:

$$L = \{a^i b^j c^k \mid i \neq j \text{ ó } j \neq k\}$$

1. Diseñar la GIC que genere el lenguaje:

$$L = \{(a+b+c)^* \mid N(a) + N(b) > N(c)\}$$

Problemas

Diseñar la GIC que genere el lenguaje:

$$L = \{0^n 1^n \mid n \geq 1\}$$

Table Text Size

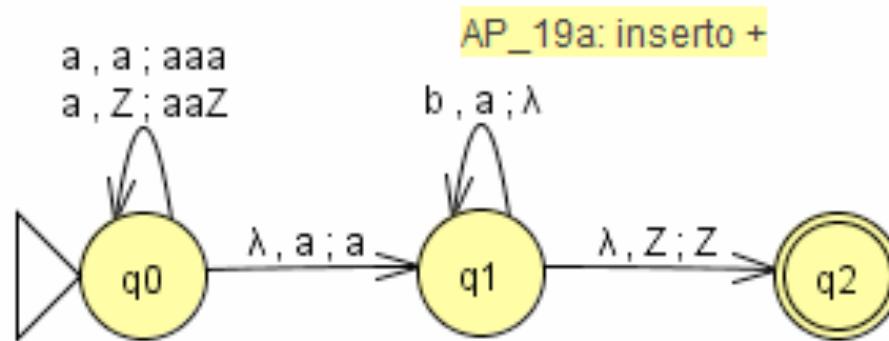


LHS		RHS
S	\rightarrow	0S1
S	\rightarrow	01

Problemas

Diseñar la GIC que genere el lenguaje:

$$L = \{a^i b^j \mid 2i = j; i, j > 0\}$$



Problemas

Diseñar la GIC que genere el lenguaje:

$$L = \{a^i b^j c^k / i \neq j \text{ ó } j \neq k\}$$

Table Text Size		
LHS		RHS
S	\rightarrow	XC
S	\rightarrow	AY
X	\rightarrow	aXb
X	\rightarrow	aA
X	\rightarrow	bB
Y	\rightarrow	bYc
Y	\rightarrow	bB
Y	\rightarrow	cC
A	\rightarrow	aA
A	\rightarrow	λ
B	\rightarrow	bB
B	\rightarrow	λ
C	\rightarrow	cC
C	\rightarrow	λ

Problemas

Diseñar la GIC que genere el lenguaje:

$$L = \{(a+b+c)^* / N(a)+N(b) > N(c)\}$$

Table Text Size

LHS		RHS
Z	\rightarrow	SaS
Z	\rightarrow	SbS
S	\rightarrow	bScS
S	\rightarrow	aScS
S	\rightarrow	cSaS
S	\rightarrow	cSbS
S	\rightarrow	aS
S	\rightarrow	bS
S	\rightarrow	λ

Formas normales para GIC

- Las gramáticas en **formas normales** pueden generar todos los LIC
 - Forma Normal de Chomsky
 - Forma Normal de Greibach
- Las gramáticas en formas normales reducen la complejidad para la obtención de las derivaciones
- Para obtener una gramática en forma normal es necesario realizar una serie de transformaciones que no modifican el lenguaje generado:
 - eliminación de producciones ϵ
 - eliminación de producciones unitarias (reglas de encadenamiento)
 - eliminación de símbolos inútiles

Formas normales para GIC

Eliminación de producciones ε

- Una variable es anulable si $A \Rightarrow^* \varepsilon$
- Algoritmo. Sea $G = (V, T, P, S)$ una GIC.
Encontraremos todos los símbolos anulables de G mediante el siguiente algoritmo:
 - Base: si $A \rightarrow \varepsilon$ es una producción de G , A es anulable
 - Paso inductivo: si existe una producción $B \rightarrow C_1C_2 \dots C_k$ en la que las C_i , $i=1,\dots,k$ son anulables, B es anulable

Formas normales para GIC

Eliminación de producciones ϵ

- Construcción de una gramática sin producciones ϵ :
 - sea $G = (V, T, P, S)$ una GIC
 - se determinan todos los símbolos anulables de G
 - se construye la gramática $G_1 = (V, T, P_1, S)$ donde P_1 se determina como sigue:
 - para cada producción $A \rightarrow X_1X_2 \dots X_k$ donde $k \geq 1$, supongamos que m de los k símbolos son anulables
 - la nueva gramática tendrá 2^m versiones de esta producción (las X_i anulables estarán presentes o ausentes en todas las combinaciones posibles)
 - si, $m = k$ no se incluirá el caso de todas las X_i ausentes
 - además, las producciones de P de la forma $A \rightarrow \epsilon$ no estarán en P_1
 - si ϵ forma parte del lenguaje, se añade la producción $S \rightarrow \epsilon$

Formas normales para GIC

Eliminación de producciones ϵ

Ejemplos:

- Dada la gramática $S \rightarrow ABC, A \rightarrow aAA \mid \epsilon, B \rightarrow bBC \mid \epsilon, C \rightarrow bc \mid \epsilon$, construir la gramática sin producciones ϵ
- Dada la gramática $S \rightarrow aS \mid AB \mid AC, A \rightarrow aA \mid \epsilon, B \rightarrow bB \mid bS, C \rightarrow cC \mid \epsilon$, construir la gramática sin producciones ϵ

Formas normales para GIC

Eliminación de producciones unitarias

- Producción unitaria: $A \rightarrow B$, donde A y B son variables
 - añaden pasos adicionales en las derivaciones
- Pares unitarios: pares de variables A y B tales que $A \Rightarrow^* B$, usando una secuencia que sólo hace uso de producciones unitarias
- Obtención de los pares unitarios (A, B)
 - Base: (A, A) es un par unitario para toda variable A
 - Paso inductivo: sea (A, B) un par unitario, y $B \rightarrow C$ una producción donde C es una variable. Entonces (A, C) es un par unitario

Formas normales para GIC

Eliminación de producciones unitarias

Ejemplo: determinar los pares unitarios de la gramática

$$\begin{array}{ll} I & \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F & \rightarrow I \mid (E) \\ T & \rightarrow F \mid T * F \\ E & \rightarrow T \mid E + T \end{array}$$

Formas normales para GIC

Eliminación de producciones unitarias

- Eliminación de las producciones unitarias: dada la GIC $G = (V, T, P, S)$, se construye la GIC

$G_1 = (V, T, P_1, S)$ como sigue:

1. encontramos los pares unitarios de G
 2. para cada par unitario (A, B) , añadimos a P_1 todas las producciones $A \rightarrow \alpha$, donde $B \rightarrow \alpha$ es una producción no unitaria de P
- Ejemplo: eliminar las producciones unitarias de la siguiente gramática

$$\begin{array}{lcl} I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F & \rightarrow & I \mid (E) \\ T & \rightarrow & F \mid T * F \\ E & \rightarrow & T \mid E + T \end{array}$$

Formas normales para GIC

Eliminación de símbolos inútiles

- Un símbolo X es útil para una gramática $G = (V, T, P, S)$ si existe alguna derivación de la forma $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$, donde w está en T^*
- X es **generador** si $X \Rightarrow^* w$ (w está en T^*)
 - todo símbolo terminal es generador
- X es **alcanzable** si existe una derivación $S \Rightarrow^* \alpha X \beta$ para algún α y β
- **Todo símbolo útil es generador y alcanzable**
- **Eliminación de símbolos inútiles**
 1. eliminamos los símbolos no generadores
 2. eliminamos los símbolos no alcanzables

Cálculo de símbolos generadores

Algoritmo: sea $G = (V, T, P, S)$ una gramática. Para calcular los símbolos generadores de G se lleva a cabo la siguiente inducción:

- Base: todo símbolo de T es generador
- Paso inductivo: dada una producción $A \rightarrow a$, donde todo símbolo de a es generador, entonces A es generador (se incluye el caso $a = \varepsilon$)

Cálculo de símbolos alcanzables

Algoritmo: sea $G = (V, T, P, S)$ una gramática. Para calcular los símbolos alcanzables de G se lleva a cabo la siguiente inducción:

- Base: S es alcanzable
- Paso inductivo: dada una variable A alcanzable, para todas las producciones cuya cabeza es A , todos los símbolos de los cuerpos de dichas producciones serán alcanzables

Problemas

Dada la gramática siguiente, eliminar producciones ϵ y unitarias y símbolos inútiles:

$$S \rightarrow AC \mid BS \mid B$$

$$A \rightarrow aA \mid aF$$

$$B \rightarrow cF \mid b$$

$$C \rightarrow cC \mid D$$

$$D \rightarrow aD \mid BD \mid C$$

$$E \rightarrow aA \mid BSA$$

$$F \rightarrow bB \mid b$$

Problemas

S	→ AC
S	→ BS
S	→ cF
S	→ b
D	→ BD
E	→ BSA
C	→ BD
D	→ cC
C	→ aD
F	→ b
F	→ bB
E	→ aA
D	→ aD
C	→ cC
B	→ b
B	→ cF
A	→ aF
A	→ aA

Hay tres producciones unitarias:
(S,B), (C,D) y (D,C), que han de reescribirse para eliminarlas, quedando como se ve en la figura

Problemas

S	→ BS
S	→ cF
S	→ b
B	→ cF
B	→ b
F	→ bB
F	→ b

Después de eliminar las producciones unitarias y los símbolos inútiles, eliminando los que no son generadores y los que no son alcanzables, queda el conjunto de producciones de la figura

Resumen de las transformaciones

Recordemos: para convertir una GIC G en una GIC equivalente que no tiene símbolos inútiles, producciones ϵ o producciones unitarias, es necesario realizar los siguientes pasos en el orden establecido:

1. eliminar las producciones ϵ
2. eliminar las producciones unitarias
3. eliminar los símbolos inútiles

Formal Normal de Chomsky

Todo LIC no vacío tiene una gramática G en la que todas las producciones tienen una de las formas siguientes:

- $A \rightarrow BC$, donde A, B, C son variables
- $A \rightarrow a$, donde A es una variable y a un símbolo terminal
- $S \rightarrow \epsilon$

Se dice que G está en Forma Normal de Chomsky (FNC)

- Una cadena de longitud n se analiza en $2n-1$ pasos
- El árbol de derivación es binario y su profundidad máxima es n
- Se usa como algoritmo el método CYK

Formal Normal de Chomsky

Ejemplos:

- Gramática: $S \rightarrow AX_1, X_1 \rightarrow BX_2, X_2 \rightarrow CX_3, X_3 \rightarrow DE, A \rightarrow a, B \rightarrow b, C \rightarrow c, D \rightarrow d, E \rightarrow e$. Cadena: abcde
- Gramática: $S \rightarrow SS \mid AB \mid CD, A \rightarrow a, B \rightarrow b, C \rightarrow c, D \rightarrow d$. Cadena: abcdab

Formal Normal de Chomsky

Para transformar una gramática a FNC:

- no puede tener producciones ϵ , producciones unitarias, ni símbolos inútiles
- las producciones de esa gramática tienen la forma $S \rightarrow \epsilon$, $A \rightarrow a$ (ya están en FNC) o bien un cuerpo de longitud dos o más. Habrá que:
 - a) conseguir que en los cuerpos de longitud dos o más sólo aparezcan variables
 - b) descomponer los cuerpos de longitud tres o más en una cascada de producciones en cuyos cuerpos sólo aparezcan dos variables

Formal Normal de Chomsky

Construcción para (a):

- para cada símbolo terminal a que aparece en un cuerpo de longitud dos o más, se crea una nueva variable A
- esta variable sólo tendrá la producción $A \rightarrow a$
- se sustituyen las apariciones de a por A siempre que aparezca en un cuerpo de longitud mayor o igual que dos

Construcción para (b):

- se descomponen las producciones de la forma $A \rightarrow B_1B_2 \dots B_k$ para $k \geq 3$, en un grupo de producciones con dos variables en cada cuerpo
- se introducen $k-2$ variables nuevas, C_1, C_2, \dots, C_{k-2}
- se reemplaza la producción original por las $k-1$ producciones $A \rightarrow B_1C_1, C_1 \rightarrow B_2C_2, \dots, C_{k-3} \rightarrow B_{k-2}C_{k-2}, C_{k-2} \rightarrow B_{k-1}B_k$

Formal Normal de Chomsky

Ejemplo: convertir la gramática en FNC

$$\begin{aligned} E &\rightarrow E + T \mid T * F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ T &\rightarrow T * F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ F &\rightarrow (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\ I &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{aligned}$$
$$\begin{aligned} E &\rightarrow EC_1 \mid TC_2 \mid LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\ T &\rightarrow TC_2 \mid LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\ F &\rightarrow LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\ I &\rightarrow a \mid b \mid IA \mid IB \mid IZ \mid IO \\ A &\rightarrow a \\ B &\rightarrow b \\ Z &\rightarrow 0 \\ O &\rightarrow 1 \\ P &\rightarrow + \\ M &\rightarrow * \\ L &\rightarrow (\\ R &\rightarrow) \\ C_1 &\rightarrow PT \\ C_2 &\rightarrow MF \\ C_3 &\rightarrow ER \end{aligned}$$

Formal Normal de Greibach

Todo LIC no vacío es $L(G)$ para alguna gramática G cuyas producciones tienen la forma: $A \rightarrow a\alpha$, donde a es un símbolo terminal y α una cadena de cero o más variables

El uso de una producción introduce un símbolo terminal en una forma sentencial

- una cadena de longitud n tiene una derivación de n pasos
- Un analizador sintáctico descendente parará a profundidad n
- Nunca habrá recursividad por la izquierda

Problemas

1. Encontrar una gramática equivalente
a: $S \rightarrow AB \mid CA, A \rightarrow a, B \rightarrow BC \mid AB,$
 $C \rightarrow aB \mid b,$ sin símbolos inútiles

2. Dada la gramática $S \rightarrow ASB \mid \epsilon, A \rightarrow aAS \mid a,$ $B \rightarrow SbS \mid A \mid bb,$ obtener la gramática equivalente en FNC

Autómatas con pila



Senén Barro Ameneiro, CiTIUS

@SenenBarro

Material elaborado fundamentalmente por
el profesor Manuel Mucientes Molina

Bibliografía

- J.E. Hopcroft, R. Motwani y J.D. Ullman,
"Teoría de Autómatas, Lenguajes y
Computación", Addison Wesley, 2008.
 - Capítulos 6 y 7
- P. Linz, "An Introduction to Formal Languages
and Automata", Jones and Bartlett Publishers,
Inc., 2001.
 - Capítulo 6-8

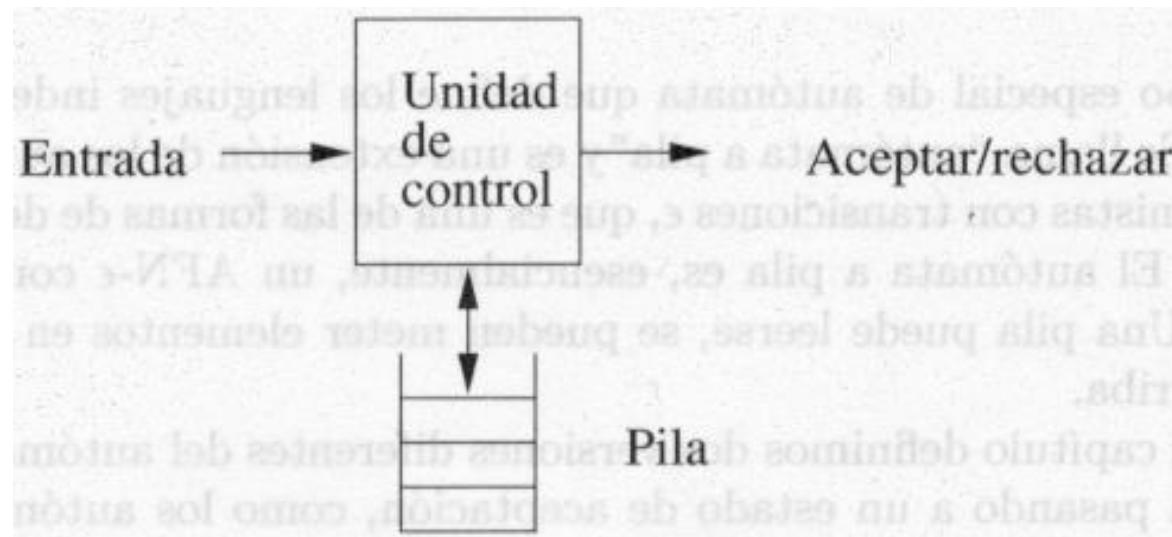
Introducción a los AP

- Un autómata con pila (AP) es un AFN con transiciones ε y con una pila en la que se puede almacenar una cadena de “símbolos de pila”
- El AP puede recordar una cantidad **infinita** de información
 - LIFO
- Los AP reconocen todos los LIC y sólo estos
 - Existen lenguajes que no son LIC.
Ejemplo: $\{0^n1^n2^n \mid n \geq 1\}$

Introducción a los AP

Funcionamiento:

- Se consume de la entrada un símbolo o bien ϵ
- Se pasa a un nuevo estado
- Se reemplaza el símbolo en lo alto de la pila por una cadena (podría ser ϵ)



Introducción a los AP

Ejemplo: diseñar el AP para $L_{WWR} = \{ww^R \mid w \text{ está en } (0+1)^*\}$

- se comienza en el estado q_0
 - suponemos que la cadena w aún no ha finalizado
 - se van almacenando los símbolos de entrada leídos en la pila
- en cualquier momento se puede suponer que w ha finalizado y se ha comenzado a leer w^R
 - el final de w estará en la cima de la pila
 - se transita al estado q_1
 - AP no determinista:
 - podemos suponer que hemos llegado al final de w
 - también podemos continuar en q_0 y seguir almacenando las entradas en la pila

Introducción a los AP

Ejemplo: diseñar el AP para el lenguaje $L_{WWR} = \{ww^R \mid w \text{ está en } (0+1)^*\}$

Sigue de antes:

- en el estado q_1 se compara el símbolo de entrada con el símbolo en la cima de la pila
 - son iguales: se elimina el símbolo de la pila
 - son distintos: no habíamos llegado al final de w . Esa rama muere
- si se vacía la pila, hemos leído ww^R y se acepta la entrada

Definición formal del AP

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

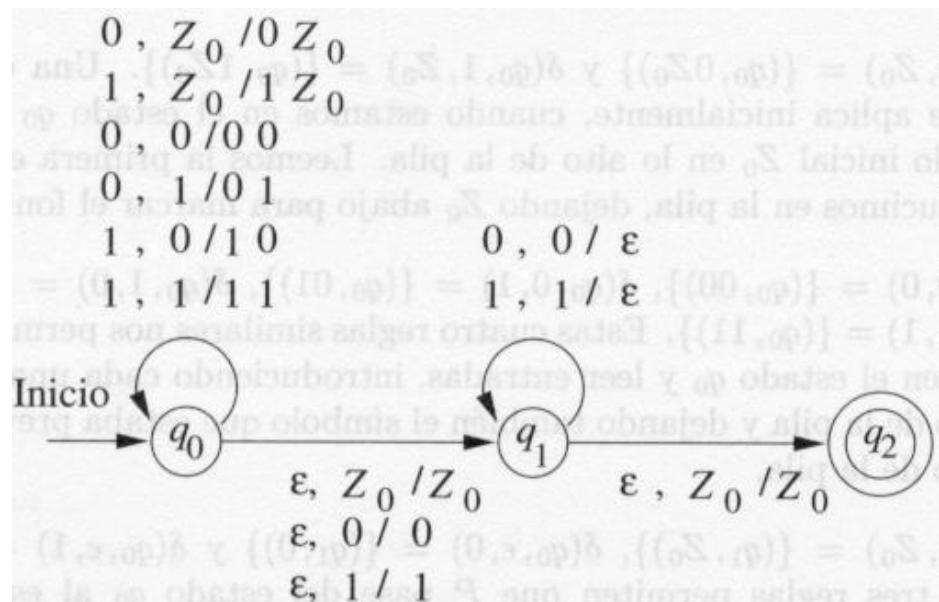
- Q : conjunto finito de estados
- Σ : conjunto finito de símbolos de entrada
- Γ : alfabeto de pila finito
- δ : función de transición, $\delta(q, a, X) = (p, \gamma)$
- q_0 : estado inicial
- Z_0 : símbolo inicial de la pila
- F : conjunto de estados de aceptación

Definición formal del AP

Ejemplo: diseñar un AP que acepte el lenguaje L_{WWR} :

- $P = (\{q_0, q_1, q_2\}, \{0, 1\}, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$, donde δ se define:

Q	S	G	Movimiento
q_0	0	Z_0	$(q_0, 0Z_0)$
q_0	1	Z_0	$(q_0, 1Z_0)$
q_0	0	0	$(q_0, 00)$
q_0	0	1	$(q_0, 01)$
q_0	1	0	$(q_0, 10)$
q_0	1	1	$(q_0, 11)$
q_0	e	Z_0	(q_1, Z_0)
q_0	e	0	$(q_1, 0)$
q_0	e	1	$(q_1, 1)$
q_1	0	0	(q_1, e)
q_1	1	1	(q_1, e)
q_1	e	Z_0	(q_2, Z_0)

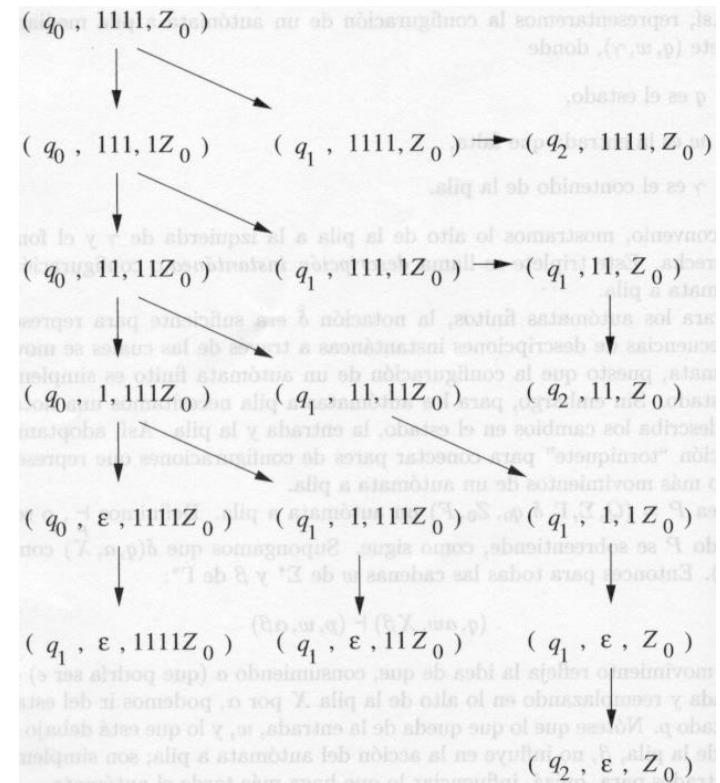
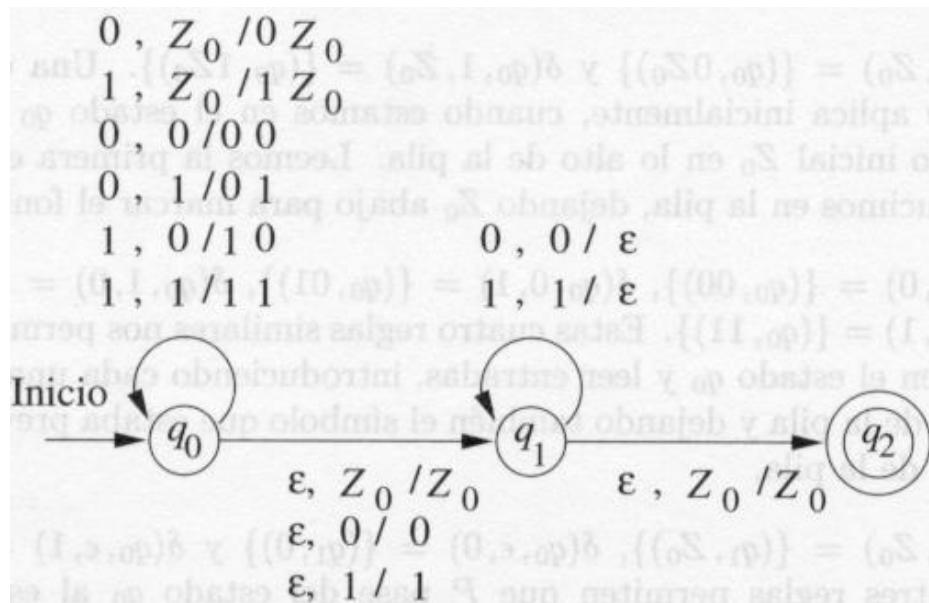


Descripción instantánea de un AP

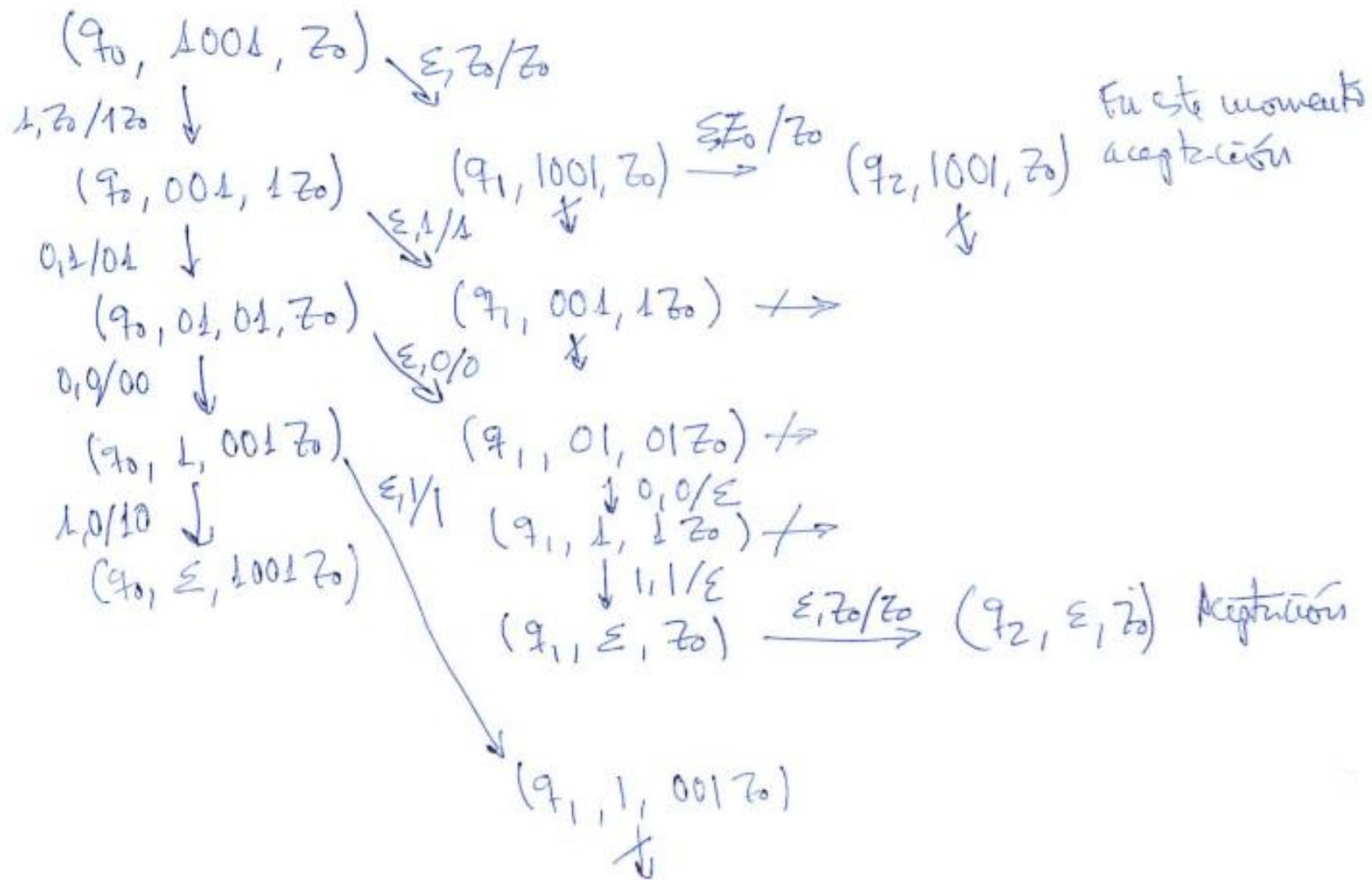
Descripción instantánea: (q, w, γ)

- q es el estado
- w es la entrada que falta por leer
- γ es el contenido de la pila (la cima de la pila se muestra a la izquierda de γ , y el fondo a la derecha)

Ejemplo para la entrada 1111



Ejemplo para la entrada 1001



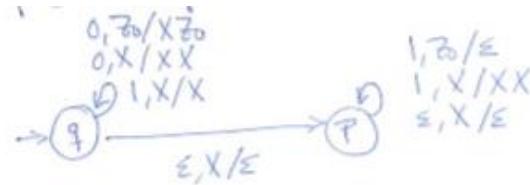
Problemas

Dado el AP $P = (\{q, p\}, \{0, 1\}, \{Z_0, X\}, \delta, q, Z_0, \{p\})$, donde δ se define en la siguiente tabla, mostrar las configuraciones alcanzables a partir de la inicial (q, w, Z_0) , para w igual a 01 y 010

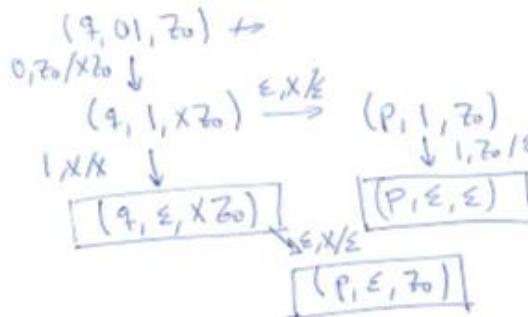
Q	Σ	Γ	Movimiento
q	0	Z_0	(q, XZ_0)
q	0	X	(q, XX)
q	1	X	(q, X)
q	ϵ	X	(p, ϵ)
p	ϵ	X	(p, ϵ)
p	1	X	(p, XX)
p	1	Z_0	(p, ϵ)

Solución

Q	Σ	Γ	Movimiento
q	0	Z_0	(q, XZ_0)
q	0	X	(q, XX)
q	1	X	(q, X)
q	ϵ	X	(p, ϵ)
p	ϵ	X	(p, ϵ)
p	1	X	(p, XX)
p	1	Z_0	(p, ϵ)

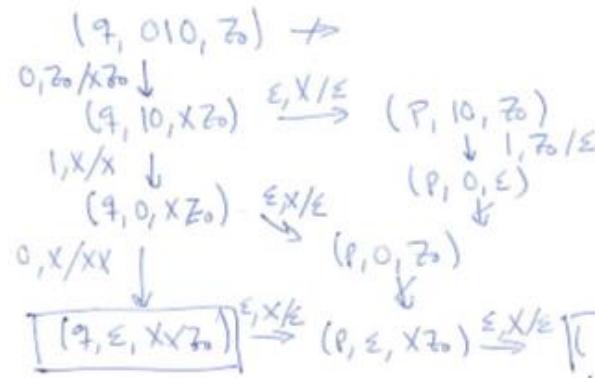


$\omega = 01$



Solución

$\omega = 010$



Solución

Problemas

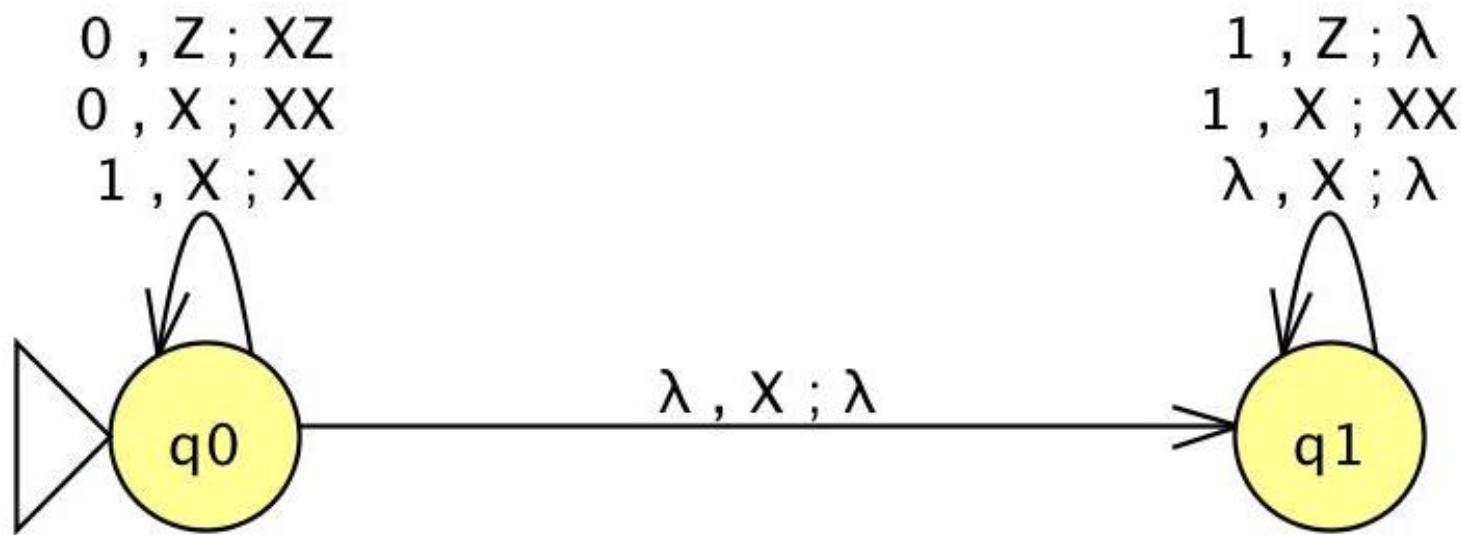
Dado el AP $P = (\{q, p\}, \{0, 1\}, \{Z_0, X\}, \delta, q, Z_0, \{p\})$, donde δ se define en la siguiente tabla, mostrar las configuraciones alcanzables a partir de la inicial (q, w, Z_0) , para w igual a 01 y 010

Q	Σ	Γ	Movimiento
q	0	Z_0	(q, XZ_0)
q	0	X	(q, XX)
q	1	X	(q, X)
q	ε	X	(p, ε)
p	ε	X	(p, ε)
p	1	X	(p, XX)
p	1	Z_0	(p, ε)

Soluciones:

- $(q, 01, Z_0) \xrightarrow{*} (p, \varepsilon, Z_0)$
 $\xrightarrow{*} (p, \varepsilon, \varepsilon)$
 $\xrightarrow{*} (q, \varepsilon, XZ_0)$
- $(q, 010, Z_0) \xrightarrow{*} (p, \varepsilon, Z_0)$
 $\xrightarrow{*} (q, \varepsilon, XXZ_0)$

Autómata de Pila



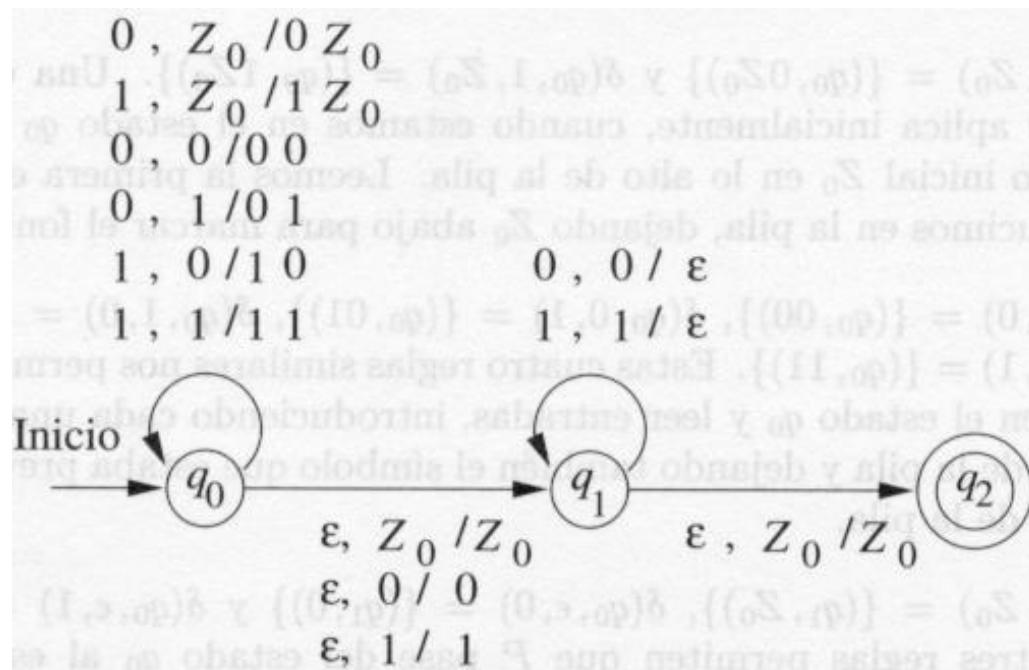
Correspondiente al AP de la página anterior

Lenguajes aceptados por un AP

- Dos tipos de aceptación
 - por estado final
 - por vaciado de pila
- Ambos métodos son equivalentes
- **Aceptación por estado final**
 - sea $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un AP
 - $L(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \alpha)\}$ para algún estado q de F y cualquier cadena de pila α

Lenguajes aceptados por un AP

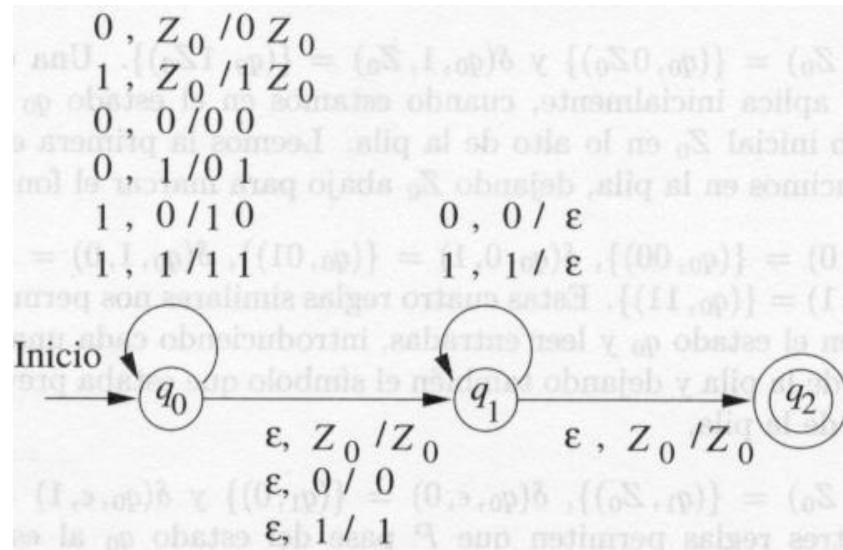
Ejemplo: el AP de la figura acepta cadenas x por estado final si y sólo si x tiene la forma ww^R



Lenguajes aceptados por un AP

Aceptación por pila vacía

- Para todo AP $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ se define el lenguaje que acepta como:
 $N(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)\}$ para cualquier estado q
- Ejemplo: modificar el AP de la figura para que reconozca por vaciado de pila
 - se cambia $\delta(q_1, \varepsilon, Z_0) = \{(q_2, Z_0)\}$ por $\delta(q_1, \varepsilon, Z_0) = \{(q_2, \varepsilon)\}$
 - se transforma q_2 en no final

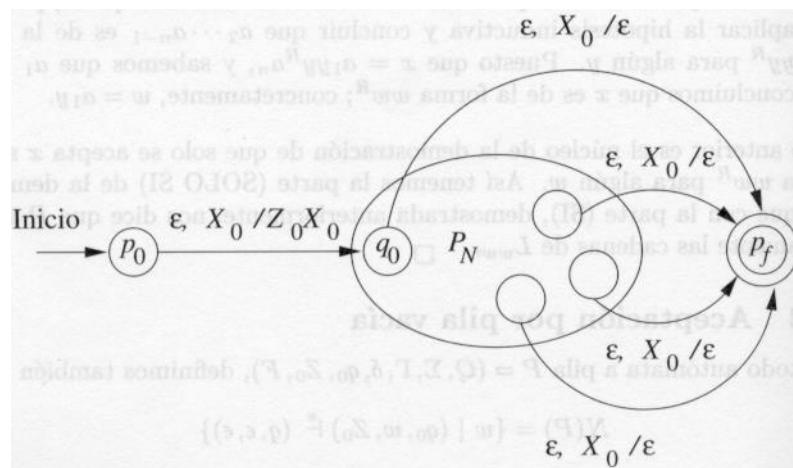


Conversión vaciado de pila a estado final

Teorema: si $L = N(P_N)$ para algún AP $P_N = (Q, \Sigma, \Gamma, \delta_N, q_0, Z_0)$, existe un AP P_F tal que $L = L(P_F)$

Prueba: $P_F = (Q \cup \{p_0, p_F\}, \Sigma, \Gamma \cup \{X_0\}, \delta_F, p_0, X_0, \{p_F\})$, donde δ_F se define:

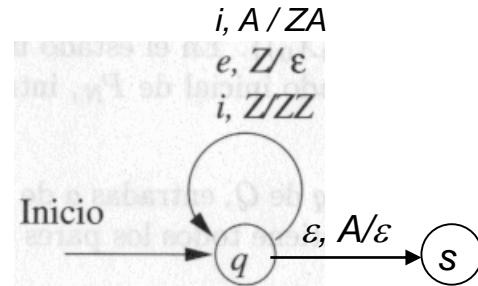
1. $\delta_F(p_0, \varepsilon, X_0) = \{(q_0, Z_0 X_0)\}$
2. para todo estado q de Q , entrada a de Σ o $a = \varepsilon$, y símbolos de pila γ de Γ , $\delta_F(q, a, \gamma)$ contiene todos los pares de $\delta_N(q, a, \gamma)$
3. además, $\delta_F(q, \varepsilon, X_0)$ contiene (p_F, ε) para todo estado q de Q



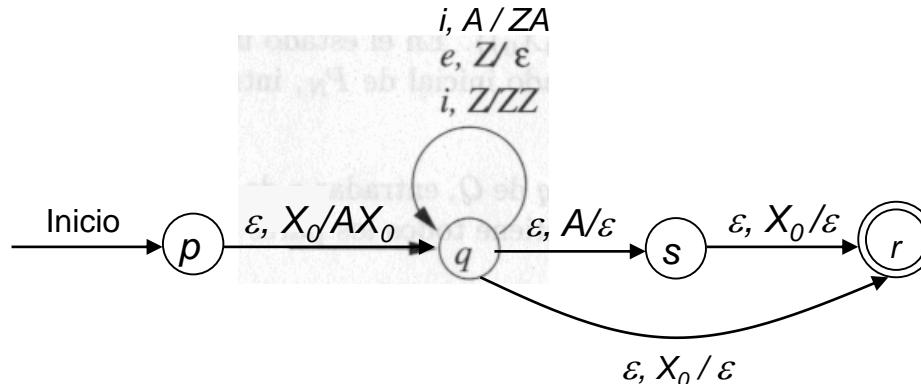
Conversión vaciado de pila a estado final

Ejemplo: diseñar un AP que procese secuencias "if else", detectando cuándo se introducen igual número de *else* que *if*

- $P_N = (\{q, s\}, \{i, e\}, \{Z, A\}, \delta_N, q, A)$



- $P_F = (\{p, q, r, s\}, \{i, e\}, \{Z, A, X_0\}, \delta_F, p, X_0, \{r\})$

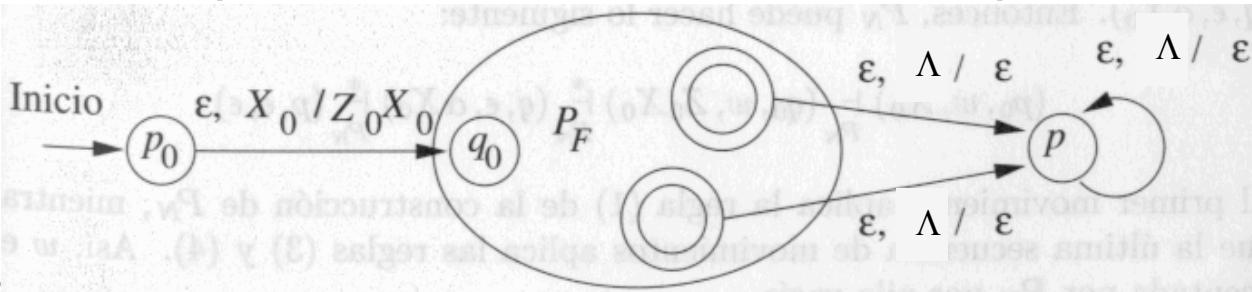


Conversión estado final a vaciado de pila

Teorema: sea L el lenguaje $L(P_F)$ de algún autómata de pila $P_F = (Q, \Sigma, \Gamma, \delta_F, q_0, Z_0, F)$. Entonces existe un AP P_N tal que $L = N(P_N)$

Prueba: $P_N = (Q \cup \{p_0, p\}, \Sigma, \Lambda = \Gamma \cup \{X_0\}, \delta_N, p_0, X_0)$, donde δ_N se define:

1. $\delta_N(p_0, \varepsilon, X_0) = \{(q_0, Z_0 X_0)\}$
2. para todo estado q de Q , entrada a de Σ o $a = \varepsilon$, y símbolos de pila Y en Γ , $\delta_N(q, a, Y)$ contiene todos los pares de $\delta_F(q, a, Y)$
3. para todo estado de aceptación q en F y símbolos de pila Y en Λ , $\delta_N(q, \varepsilon, Y)$ contiene (p, ε)
4. para todos los símbolos de la pila Y en Λ , $\delta_N(p, \varepsilon, Y) = \{(p, \varepsilon)\}$



Necesidad del nuevo símbolo inicial de pila

de pila: si el APF vacía su pila en un estado no final no debería reconocer la secuencia. Si no se añadiese el nuevo símbolo inicial de pila, el APN pasaría a reconocer en esas situaciones.

Equivalencia entre AP y GIC

El objetivo es demostrar que los tres siguientes lenguajes son todos de la misma clase:

1. lenguajes independientes del contexto
 2. lenguajes aceptados por estado final por algún AP
 3. lenguajes aceptados por vaciado de pila por algún AP
-
- La equivalencia de (2) y (3) ya se ha demostrado
 - Demostraremos que de (1) se sigue (3), aunque no que de (3) se sigue (1)

Conversión de gramáticas a AP

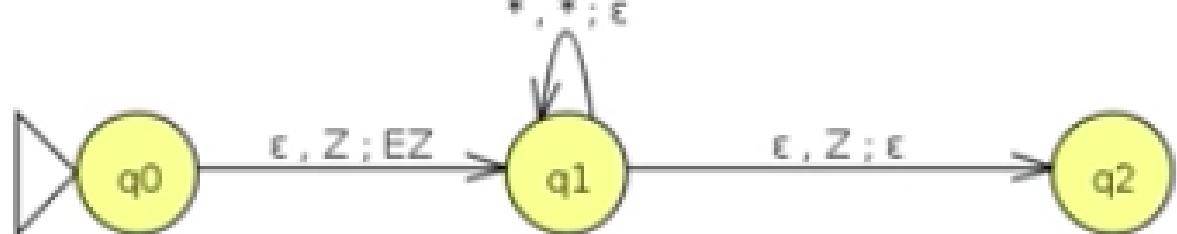
- Sea la GIC $G = (V, T, Q, S)$. El AP que acepta $L(G)$ por pila vacía será:
 - $P = (\{q\}, T, V \cup T, \delta, q, S)$
 - δ se define por:
 1. para cada variable A , $\delta(q, \varepsilon, A) = \{(q, \beta) \mid A \xrightarrow{} \beta \text{ es una producción de } P\}$
 2. para cada símbolo terminal a , $\delta(q, a, a) = (q, \varepsilon)$
- Ejemplo: obtener el AP que reconozca por vaciado de pila la siguiente gramática:

$I \xrightarrow{} a \mid b \mid Ia \mid Ib \mid I0 \mid I1, E \xrightarrow{} I \mid E^*E \mid E+E \mid (E)$

Solución al ejemplo

$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1, E \rightarrow I \mid E^*E \mid E+E \mid (E)$

$\epsilon, E; E+E$
 $\epsilon, I; Ib$
 $\epsilon, I; Ia$
 $\epsilon, E; (E)$
 $\epsilon, E; I$
 $\epsilon, I; a$
 $\epsilon, I; b$
 $\epsilon, I; I0$
 $\epsilon, I; I1$
 $\epsilon, E; E^*E$
 $a, a; \epsilon$
 $b, b; \epsilon$
 $0, 0; \epsilon$
 $1, 1; \epsilon$
 $(, (; \epsilon$
 $),); \epsilon$
 $+, +; \epsilon$
 $*, *; \epsilon$



Autómatas con pila deterministas

Los APD aceptan un conjunto de lenguajes a medio camino entre los lenguajes regulares y las GIC

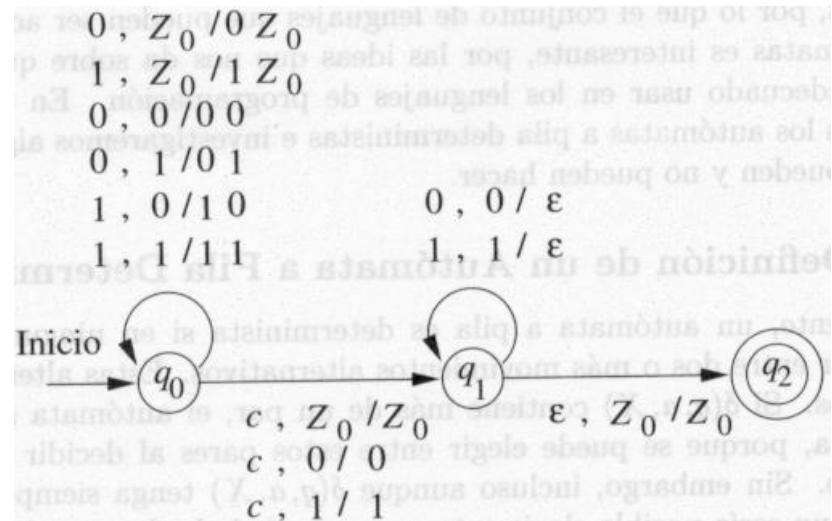
- los analizadores sintácticos se comportan generalmente como APD

Un AP $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ es determinista si:

- $\delta(q, a, X)$ tiene como máximo un elemento para cualquier q en Q , a en Σ o $a = \epsilon$, y X en Γ
- si $\delta(q, a, X)$ no está vacío para algún a en Σ , $\delta(q, \epsilon, X)$ debe estar vacío

Ejemplo:

$$L_{wcwr} = \{wcw^R \mid w \text{ está en } (0 + 1)^*\}$$



Problemas

1. Determinar si el siguiente AP es o no determinista: $P = (\{q, p\}, \{0, 1\}, \{Z_0, X\}, \delta, q, Z_0, \{p\})$, donde δ se define en la siguiente tabla

Q	Σ	Γ	Movimiento
q	0	Z_0	(q, XZ_0)
q	0	X	(q, XX)
q	1	X	(q, X)

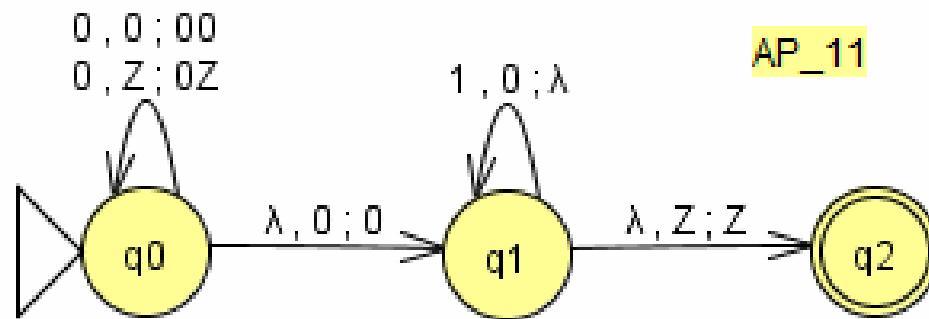
Q	Σ	Γ	Movimiento
q	ϵ	X	(p, ϵ)
p	ϵ	X	(p, ϵ)
p	1	X	(p, XX)
p	1	Z_0	(p, ϵ)

2. Diseñar un APF que acepte el lenguaje $\{0^n 1^n \mid n > 0\}$
- Restricción: el alfabeto de la pila será igual al alfabeto de entrada más el símbolo inicial de pila.
3. Igual que el anterior, pero con $n \geq 0$.

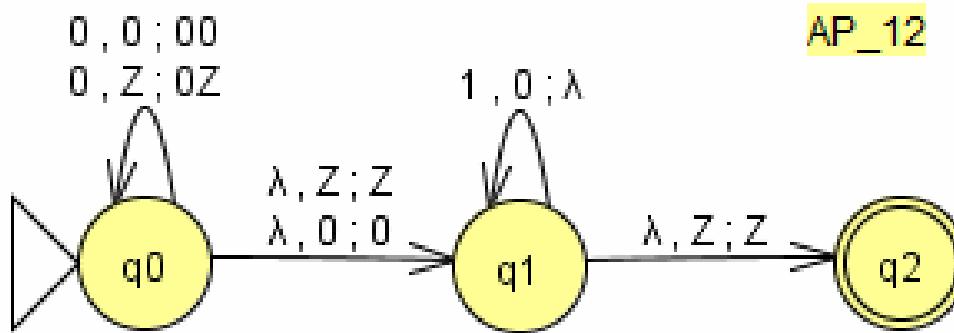
Problemas

2. Diseñar un APF que acepte el lenguaje $\{0^n 1^n \mid n > 0\}$

- Restricción: el alfabeto de la pila será igual al alfabeto de entrada más el símbolo inicial de pila.



3. Igual que el anterior, pero con $n \geq 0$.



Problemas

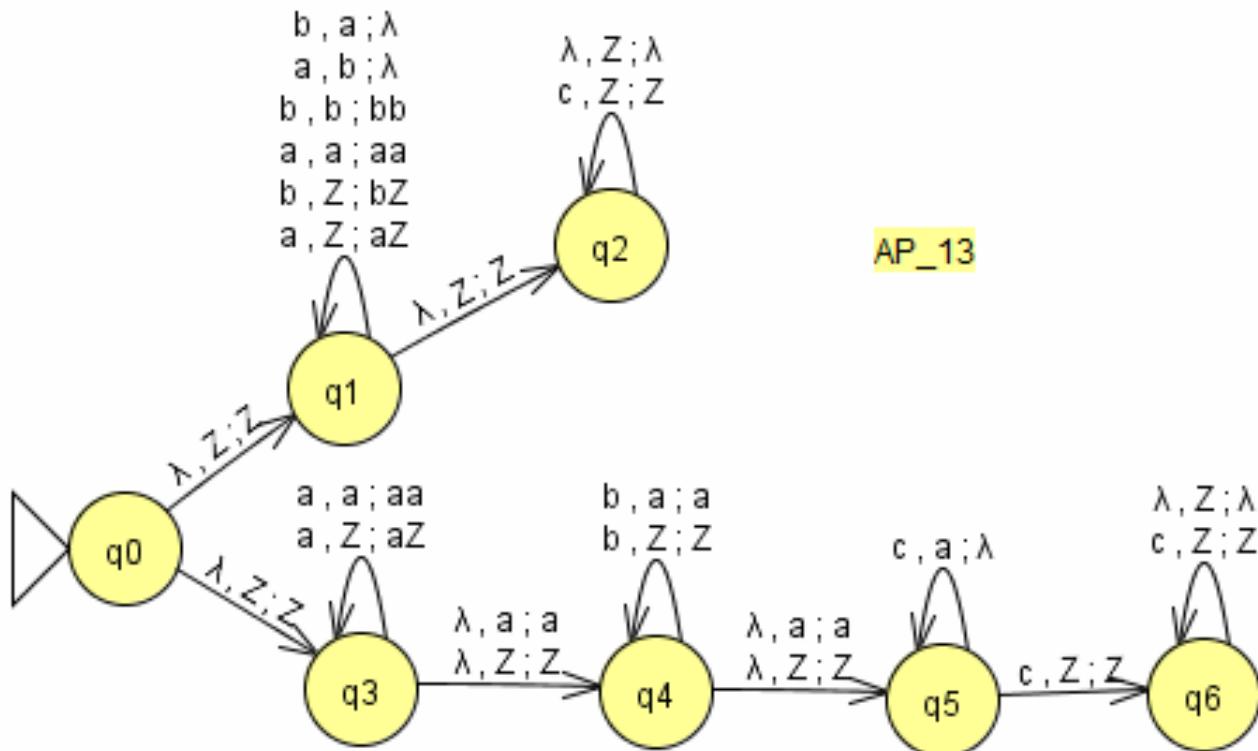
1. Diseñar un **APN** que acepte el lenguaje formado por aquellas cadenas que cumplen alguno de los siguientes criterios:

- contienen igual número de símbolos **a** y **b**, entrando estos en cualquier orden, y finalizan con un número k de símbolos **c**, $k \geq 0$.
- $a^i b^j c^k / k > i$

Restricción: el alfabeto de la pila será igual al alfabeto de entrada más el símbolo inicial de pila.

Problemas

Solución:



Problemas

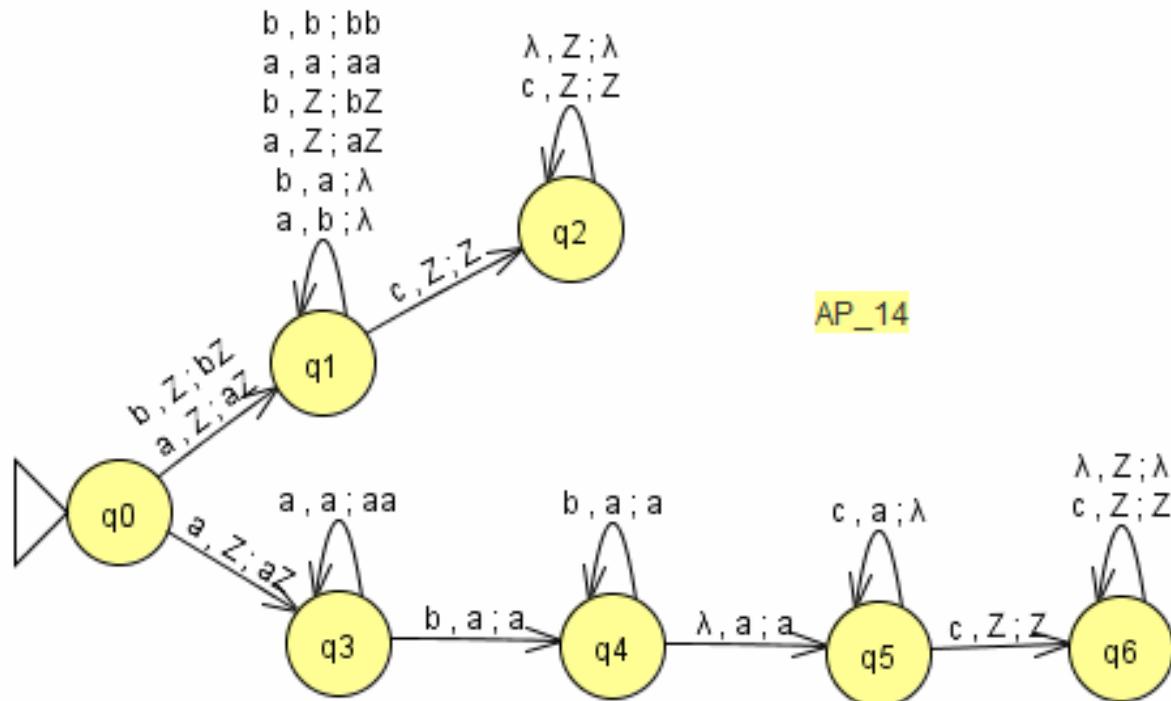
1. Igual que el anterior, pero las cadenas deben tener ahora al menos un símbolo de cada tipo.
2. Diseñar el **APF** sobre el alfabeto $\{a, b\}$ que acepte el lenguaje $\{a^i b^j \mid 2i = j; i, j > 0\}$.

Restricción: el alfabeto de la pila será igual al alfabeto de entrada más el símbolo inicial de pila.

3. Igual que el anterior, pero para la condición $i = 2j$.

Problemas

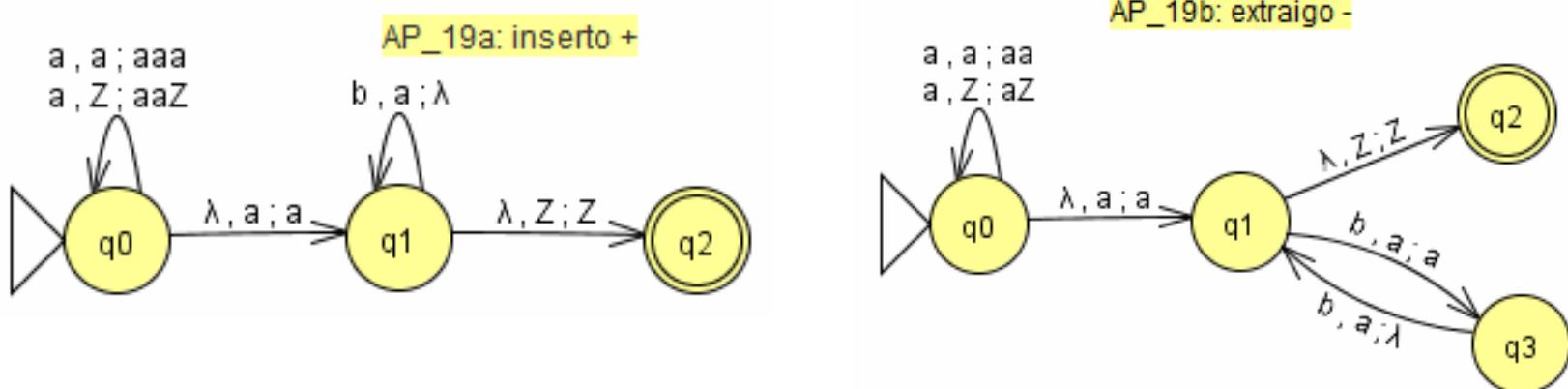
1. Igual que el anterior, pero las cadenas deben tener ahora al menos un símbolo de cada tipo.



Problemas

Diseñar el APF sobre el alfabeto $\{a, b\}$ que acepte el lenguaje $\{a^i b^j \mid 2i = j; i, j > 0\}$.

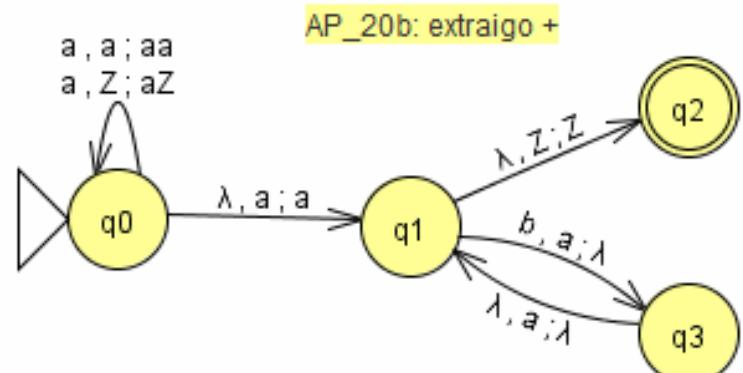
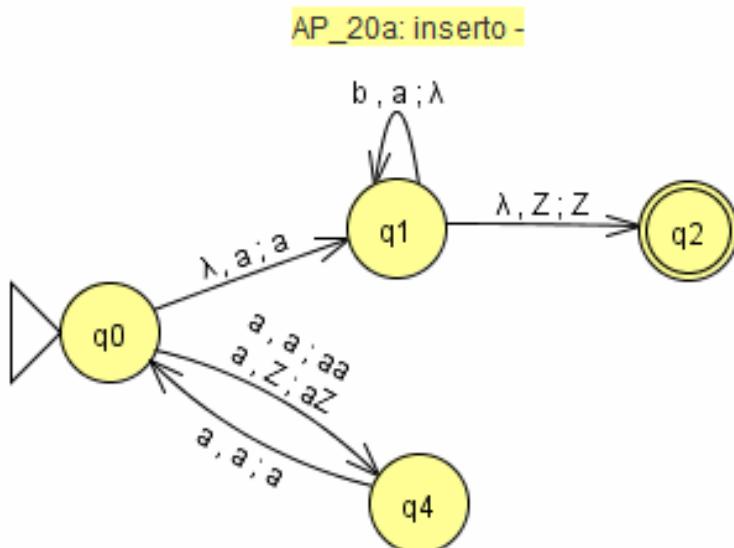
Restricción: el alfabeto de la pila será igual al alfabeto de entrada más el símbolo inicial de pila.



Problemas

Diseñar el APF sobre el alfabeto $\{a, b\}$ que acepte el lenguaje $\{a^i b^j \mid i = 2j; i, j > 0\}$.

Restricción: el alfabeto de la pila será igual al alfabeto de entrada más el símbolo inicial de pila.



Lema de bombeo para LIC

- Para un LIC, el cumplimiento del lema de bombeo (LB) es una condición **necesaria**, pero **no suficiente**
- Teorema: sea L un LIC. Entonces existe una constante n tal que si z es cualquier cadena de L de longitud $|z| \geq n$, podemos escribir $z=uvwxy$, con las siguientes condiciones:
 1. $|vwx| \leq n$
 2. $vx \neq \epsilon$
 3. Para todo $k \geq 0$, uv^kwx^ky está en L

Aplicación del lema de bombeo

1. Elegimos L del que queremos demostrar que no es LIC
2. El valor de n es desconocido, por lo que debemos considerar cualquier posible valor
3. Elegimos z (podemos usar n como parámetro)
4. Repetir para todas las descomposiciones:
 1. Escoger una descomposición de z en $uvwxy$, sujeta a las restricciones:
 1. $vx \neq \epsilon$
 2. $|vwx| \leq n$
 2. Si uv^kwx^ky pertenece a L para todo valor de k
 1. Se verifica el LB
 2. No se puede afirmar que el lenguaje sea independiente del contexto
 3. No es necesario probar con otras descomposiciones (finaliza el algoritmo)
5. Si 4.2 no se ha cumplido para ninguna descomposición, no se verifica el LB y, por tanto, el lenguaje no es un LIC

Aplicación del lema de bombeo

Verificar si se cumple el lema del bombeo para: $L=\{a^n b^n c^n \mid n \geq 1\}$

1. Si L es un LIC, entonces existe una constante n tal que si z es cualquier cadena de L de longitud $|z| \geq n$, podemos escribir $z=uvwxy$, cumpliendo las condiciones antes vistas.
2. Elegimos $z = a^n b^n c^n$
3. Al tener que cumplirse: $|vwx| \leq n$, vwx no pueden contener al mismo tiempo los tres símbolos del alfabeto. Entonces tampoco vx , que al menos, eso sí, contendrá un símbolo.
4. Por tanto, si consideramos $k=0$, resulta $uv^0wx^0y = uwy$, y esta cadena no podrá pertenecer a L ya que le faltarán los elementos de vx para estar equilibrada en número de “aes”, “bes” y “ces”.

Ejemplos de Lenguajes no IC

- $L=\{0^p1^p2^p \mid p \geq 1\}$
 - un LIC no puede emparejar tres grupos de símbolos de acuerdo con su igualdad o desigualdad
- $L=\{0^i1^j2^i3^j \mid i \geq 1 \text{ y } j \geq 1\}$
 - un LIC no puede emparejar dos pares de números iguales de símbolos que se entrelacen
- $L=\{ss \mid s \in (0+1)^*\}$
 - un LIC no puede emparejar dos cadenas de longitud arbitraria si las cadenas se eligen de un alfabeto de más de un símbolo

Problemas

Dados los siguientes lenguajes:

- Razonar si son LIC, es decir, si es posible reconocerlos con un autómata con pila.
 1. $L=\{0^p1^p \mid p \geq 1\}$
 2. $L=\{0^p1^p2^p \mid p \geq 1\}$

Máquinas de Turing



Senén Barro Ameneiro, CiTIUS

@SenenBarro

Material elaborado fundamentalmente por
el profesor Manuel Mucientes Molina

Bibliografía

- J.E. Hopcroft, R. Motwani y J.D. Ullman,
"Teoría de Autómatas, Lenguajes y
Computación", Addison Wesley, 2008.
 - Capítulo 8
- P. Linz, "An Introduction to Formal Languages
and Automata", Jones and Bartlett Publishers,
Inc., 2001.
 - Capítulo 10

Alan Turing



"Máquinas inteligentes", 1948*

.. una ilimitada capacidad de memoria obtenida en la forma de una cinta infinita marcada con cuadrados, en cada uno de los cuales podría imprimirse un símbolo. En cualquier momento hay un símbolo en la máquina; llamado el símbolo leído. La máquina puede alterar el símbolo leído y su comportamiento está en parte determinado por ese símbolo, pero los símbolos en otros lugares de la cinta no afectan el comportamiento de la máquina. Sin embargo, la cinta se puede mover hacia adelante y hacia atrás a través de la máquina, siendo esto una de las operaciones elementales de la máquina. **Por lo tanto cualquier símbolo en la cinta puede tener finalmente una oportunidad.**

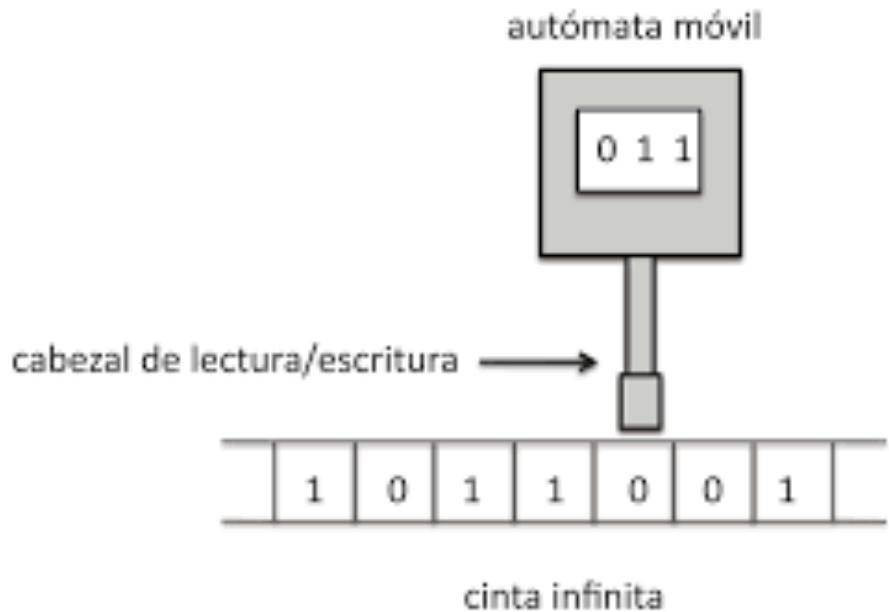
* Explicación de la "máquina de computación lógica", después conocida como: Máquina de Turing

Máquina de Turing

Una Máquina de Turing

(MT) es un autómata que cuenta con un dispositivo de almacenamiento denominado cinta.

Asociada con la cinta, existe una cabeza de lectura/escritura



Máquina de Turing

- En una MT:
 - La entrada está escrita en la cinta al comienzo
 - La salida se escribirá en la cinta durante la operación de la MT
- $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
 - Q : conjunto de estados
 - Σ : alfabeto de entrada
 - Γ : alfabeto de la cinta
 - δ : función de transición
 - $q_0 \in Q$: estado inicial
 - $B \in \Gamma$: espacio en blanco ($B \notin \Sigma$)
 - $F \subseteq Q$: conjunto de estados finales
- $\Sigma \subseteq \Gamma - \{B\}$: el alfabeto de entrada es un subconjunto del alfabeto de cinta sin el espacio en blanco

Máquina de Turing

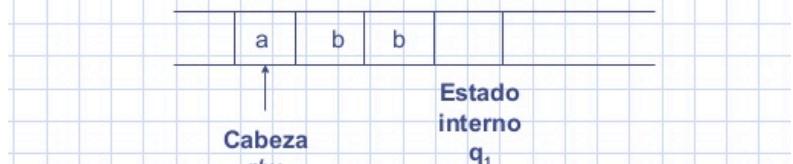
$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{I, D\}$$

- Acciones:
 - Escribir el nuevo símbolo
 - Cambiar de estado
 - Mover la cabeza de lectura/escritura
- Movimiento tras la operación de lectura y escritura
- La cabeza se desplaza una única posición Izda/Dcha

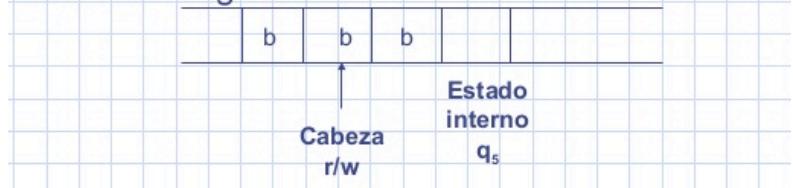
Ejemplo:

$$\delta(q_1, a) = (q_5, b, R)$$

◆ La transición $\delta(q_1, a) = (q_5, b, R)$ provoca que la TM pase de una configuración:



◆ A la configuración:

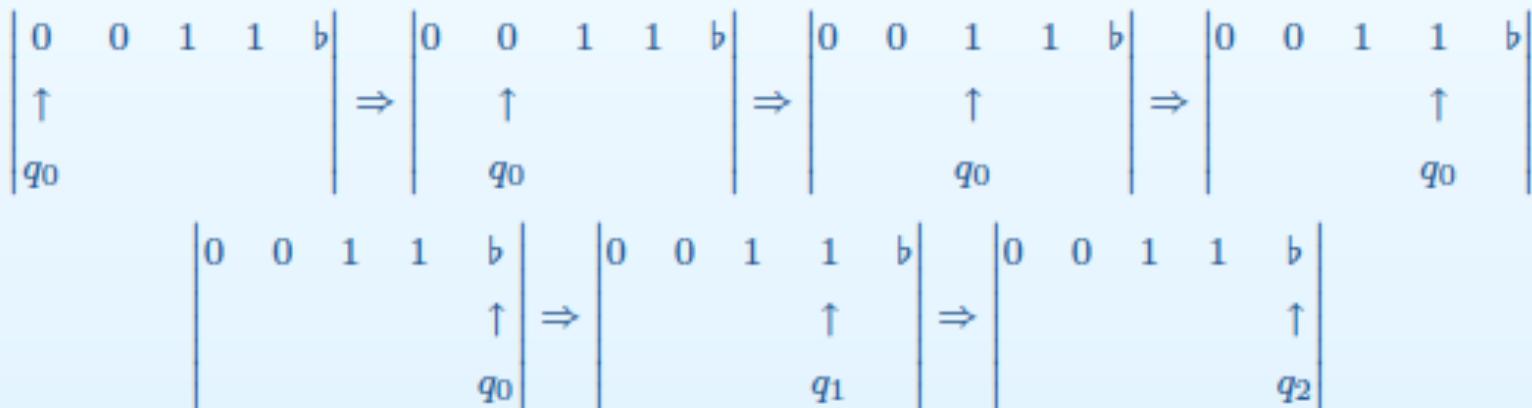


Máquina de Turing: ejemplo

Ejemplo:

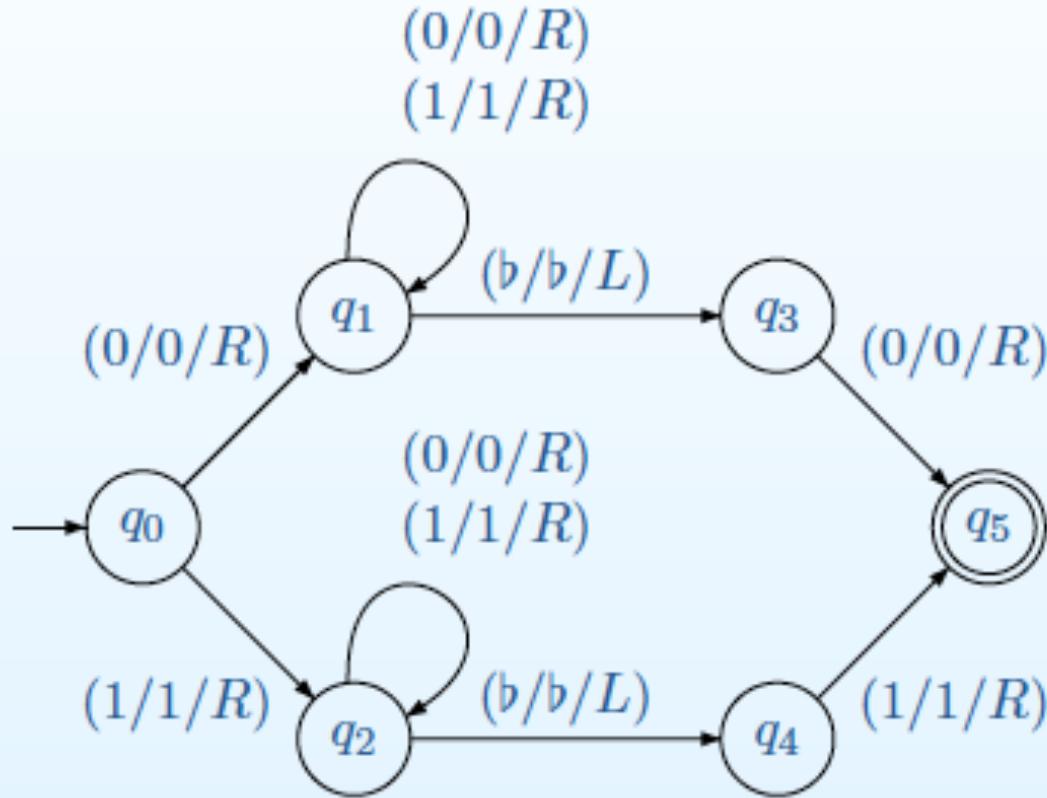
	0	1	\flat	
q_0	$(q_0, 0, R)$	$(q_0, 1, R)$	(q_1, \flat, L)	
q_1	—	$(q_2, 1, R)$	—	

$$F = \{q_2\}$$

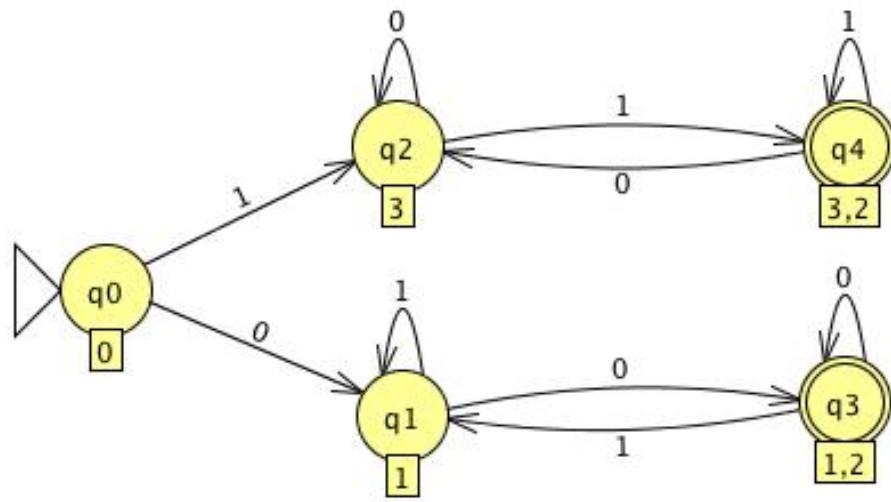
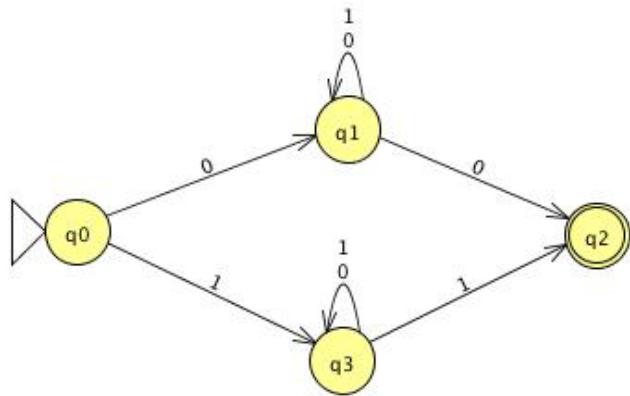


Máquina de Turing como aceptora

Ejemplo 2: Máquina que acepta el lenguaje de palabras sobre $\{0, 1\}$ que comienzan y acaban con el mismo símbolo



Autómata Finito del ejemplo anterior



AFN (Izquierda) y equivalente AFD

Problemas

1. MT que, dados dos enteros x e y , calcule $x+y$:

- Ejemplo de contenido inicial de la cinta: $w(x)0w(y)$
- Ejemplo de contenido final de la cinta: $w(x+y)0$

2. MT que realice la resta de dos números en alfabeto unario:

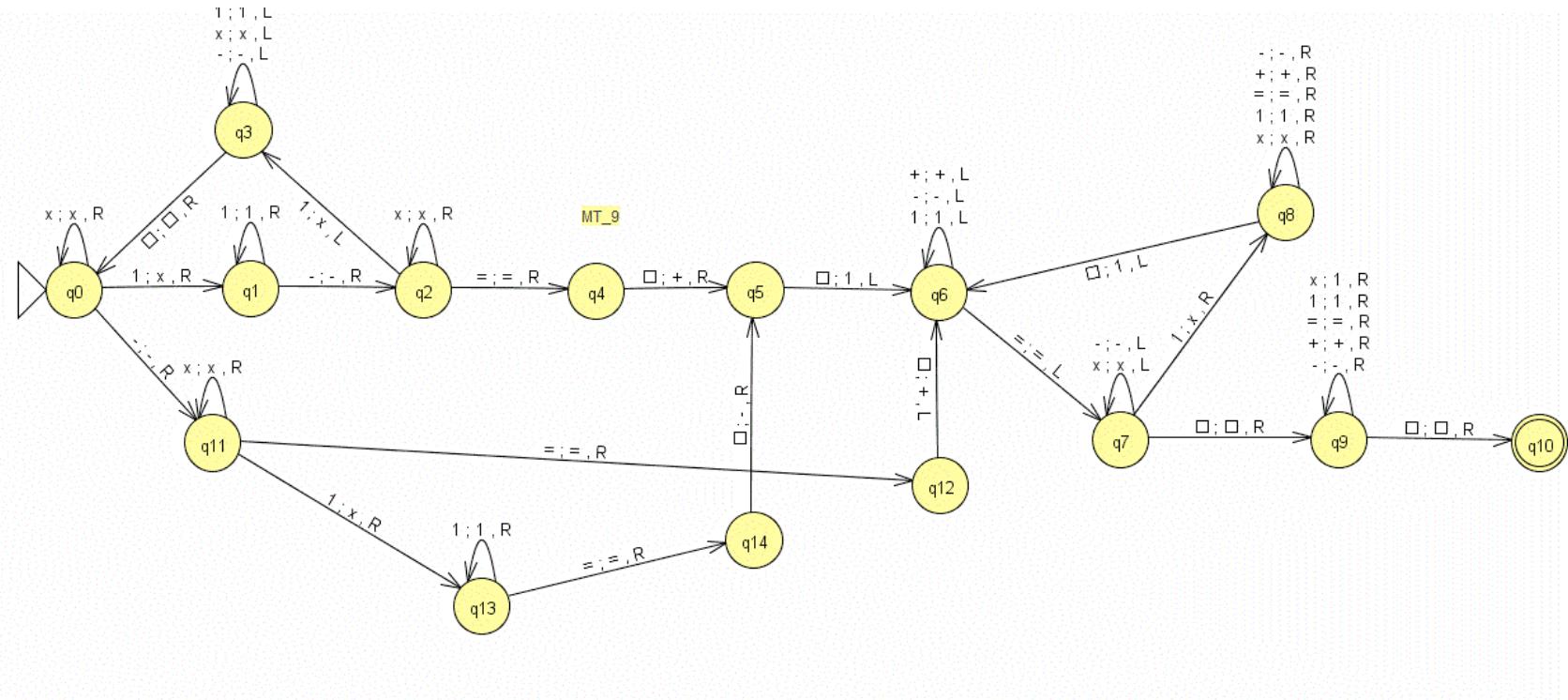
- Ejemplo de contenido inicial de la cinta: “111-11=“.
- Ejemplos de contenido final de la cinta: “1111-11=+11”, “11-1111=-11”, “11-11=+“.

Problemas



MT que computa $x+y$

Problemas

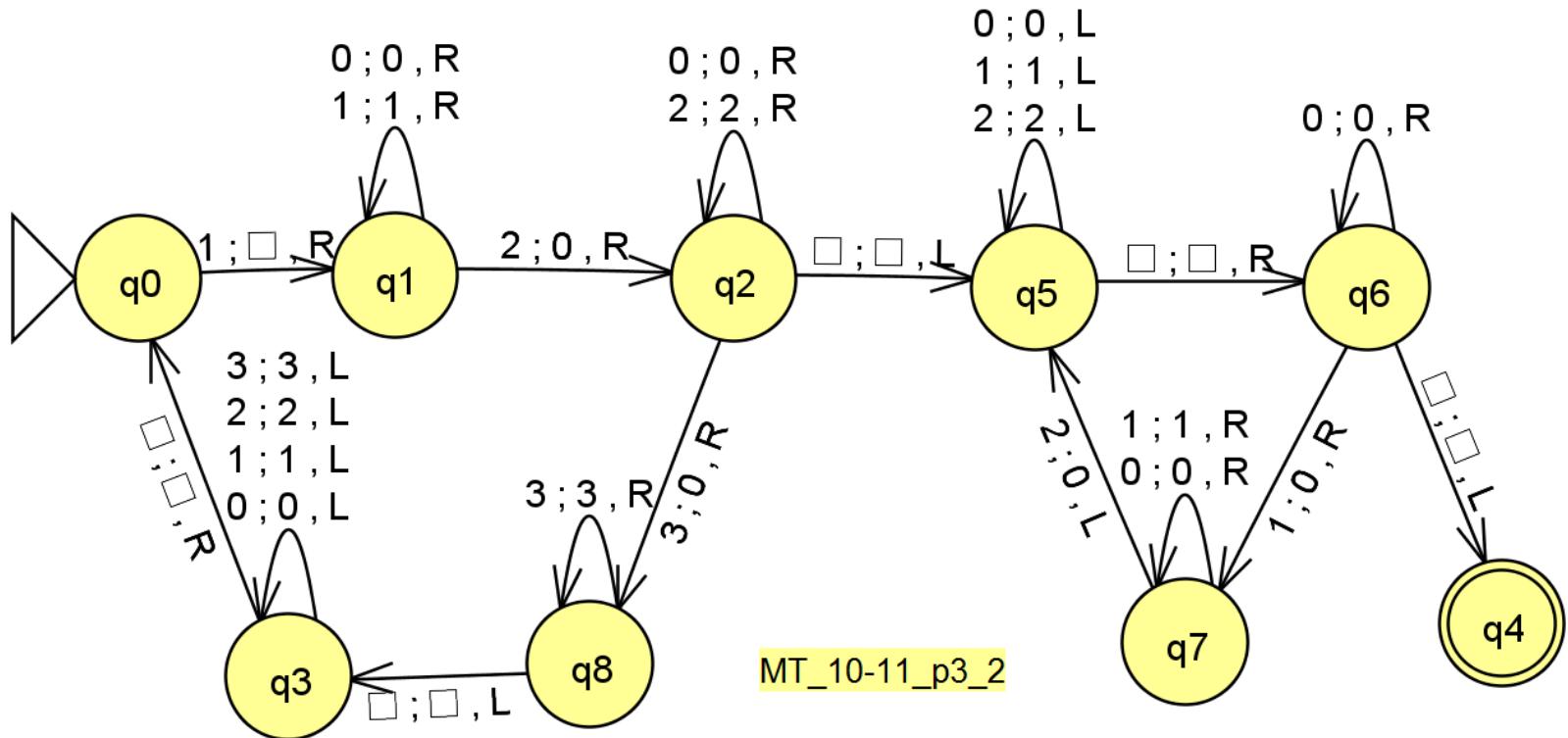


MT que realiza la resta

Problemas

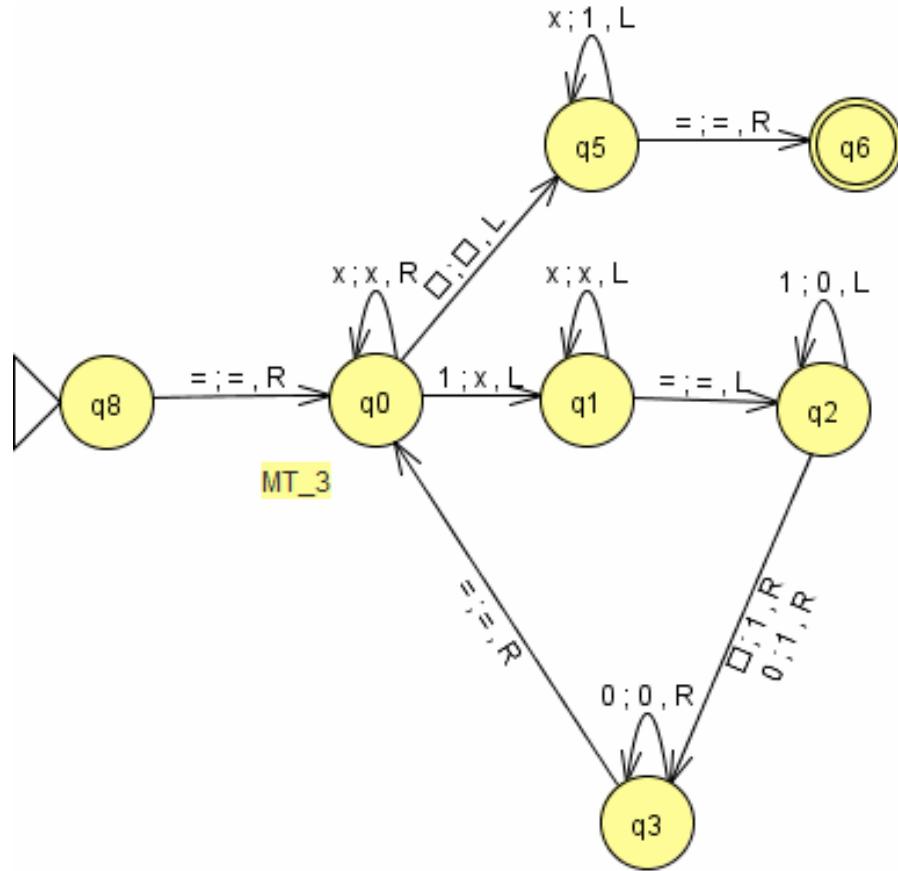
1. MT que reconozca expresiones del siguiente lenguaje: $L = \{1^n 2^n 3^k \mid n > k\}$
2. MT que, dados dos números enteros (a, b) , acepte cuando $a \geq b$.
3. MT que convierta un número entero en formato unario a formato binario.

Problemas



MT asociada al lenguaje: $L = \{1^n 2^n 3^k \mid n > k\}$

Problemas



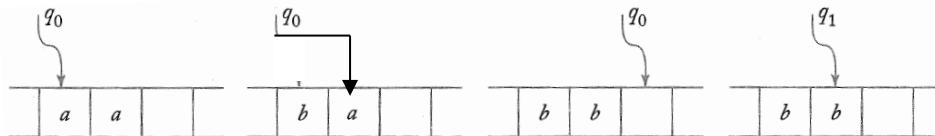
MT que transforma números unarios en binarios

Máquina de Turing (MT)

- La MT finaliza el procesamiento cuando llega a un estado de parada
 - No hay transiciones definidas para esa combinación de estado y símbolo
 - Se asume que los estados finales no tienen transiciones definidas
 - Una MT se parará siempre que alcance un estado final
- En una MT **no es necesario leer todo el contenido** de la cinta para aceptar
 - Los AF y AP sí requieren leer toda la entrada

Máquina de Turing

- Ejemplo: $Q = \{q_0, q_1\}$, $\delta(q_0, a) = (q_0, b, R)$,
 $\Sigma = \{a, b\}$, $\delta(q_0, b) = (q_0, b, R)$,
 $\Gamma = \{a, b, \square\}$, $\delta(q_0, \square) = (q_1, \square, L)$.
 $F = \{q_1\}$,



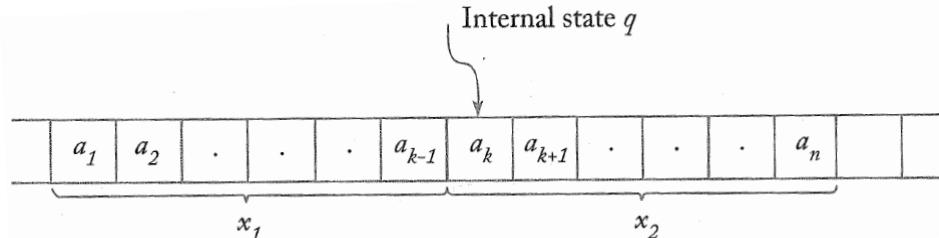
- Ejemplo: no hay parada de la MT

$$\begin{array}{ll} Q = \{q_0, q_1\}, & \delta(q_0, a) = (q_1, a, R), \\ \Sigma = \{a, b\}, & \delta(q_0, b) = (q_1, b, R), \\ \Gamma = \{a, b, \square\}, & \delta(q_0, \square) = (q_1, \square, R), \\ F = \{ \quad \}, & \delta(q_1, a) = (q_0, a, L), \\ & \delta(q_1, b) = (q_0, b, L), \\ & \delta(q_1, \square) = (q_0, \square, L). \end{array}$$



Máquina de Turing

- Resumen de las características de la MT estándar
 - Cinta infinita en ambas direcciones
 - La función de transición es determinista
 - Máximo un movimiento definido para cada configuración
 - No hay una entrada ni una salida específicas: codificados en la cinta
- Descripción instantánea
 - Estado
 - Contenido de la cinta
 - Posición de la cabeza de lectura/escritura
 - $a_1 \dots a_{k-1} q a_k \dots a_n$
 - Blancos sólo incluidos si son relevantes



Máquina de Turing

Sea $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Descripción instantánea de M : $a_1 \dots a_{k-1} q_1 a_k \dots a_n$, $a_i \in \Gamma$, $q_1 \in Q$

Movimiento a la derecha:

$a_1 \dots a_{k-1} q_1 a_k \dots a_n \vdash a_1 \dots a_{k-1} b q_2 a_{k+1} \dots a_n$, si $\delta(q_1, a_k) = (q_2, b, D)$

Movimiento a la izquierda:

$a_1 \dots a_{k-1} q_1 a_k \dots a_n \vdash a_1 \dots a_{k-2} q_2 a_{k-1} b a_{k+1} \dots a_n$, si $\delta(q_1, a_k) = (q_2, b, I)$

M se para partiendo de $x_1 q_i x_2$, si $x_1 q_i x_2 \vdash^* y_1 q_j a y_2$, para algún $\delta(q_j, a)$ no definido

- Computación: secuencia de configuraciones que llevan a la parada (entre ellas alcanzar un estado final, claro)
- No parada: $x_1 q_i x_2 \vdash^* \infty$

Máquina de Turing y Lenguajes

- $L(M) = \{w \in \Sigma^+ : q_0 w \xrightarrow{*} x_1 q_f x_2, q_f \in F, x_1, x_2 \in \Gamma^*\}$
- Los espacios en blanco se usan para delimitar la cadena de entrada
 - La cadena vacía no forma parte del lenguaje para poder limitar la región en la que se busca la entrada.
- Si $w \notin L(M)$:
 - M se para en un estado no final
 - M entra en un bucle infinito: no hay parada
- Ejemplo: MT que reconoce $L = \{0^n 1^n \mid n \geq 1\}$

- Una TM que acepta: $\{0^n 1^n \mid n \geq 1\}$
- La podemos definir como: $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\})$
- Con la siguiente tabla de transición:

Estado	0	1	X	Y	B
q_0	(q_1, X, R)	—	—	(q_3, Y, R)	—
q_1	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)	—
q_2	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)	—
q_3	—	—	—	(q_3, Y, R)	(q_4, B, R)
q_4	—	—	—	—	—

Máquina de Turing

$L = \{0^n 1^n \mid n \geq 1\}$

Máquina de Turing: $L = \{0^n 1^n \mid n \geq 1\}$

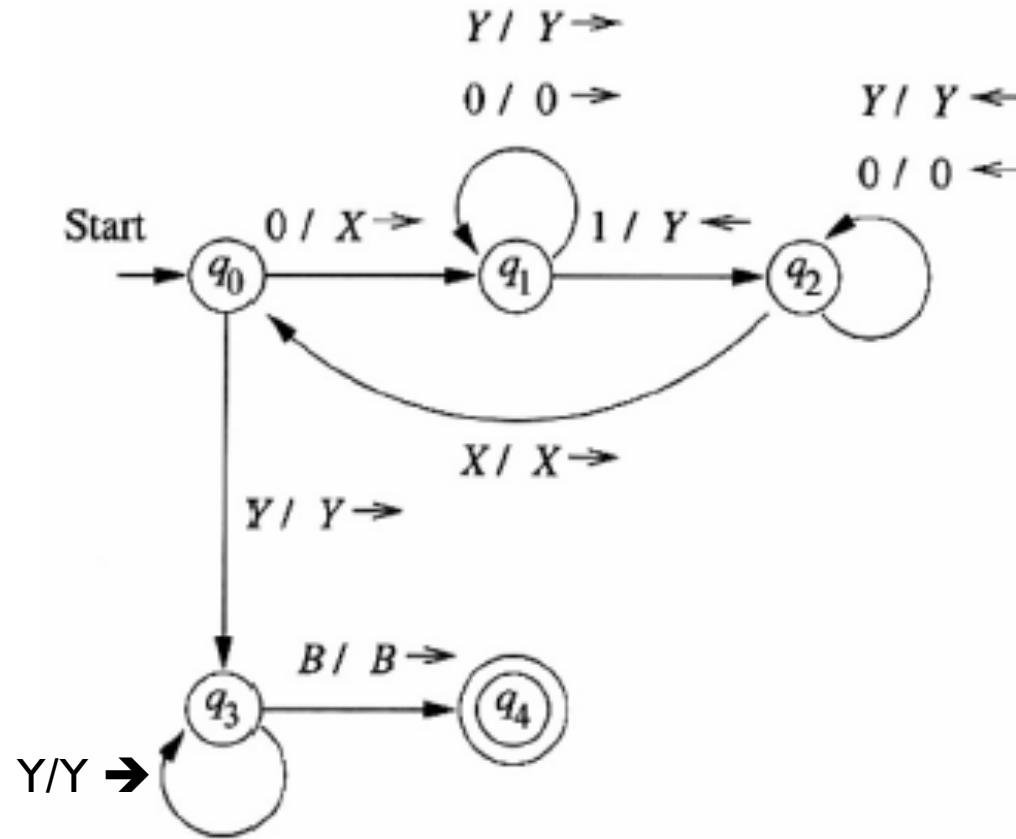


Diagrama de transición del ejemplo

- Diseñar una TM que calcula la función $\dot{-}$ llamada *monus* o substracción propia, que se define como:
 $m \dot{-} n = \max(m - n, 0)$.
- La siguiente tabla y diagrama de transición lo definen, con entrada $0^m 1 0^n$:

Estado	0	1	Símbolo
	0	1	B
q_0	(q_1, B, R)	(q_5, B, R)	—
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	—
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)
q_6	—	—	—

Máquina de Turing

Función *monus*

Máquina de Turing: ejemplo

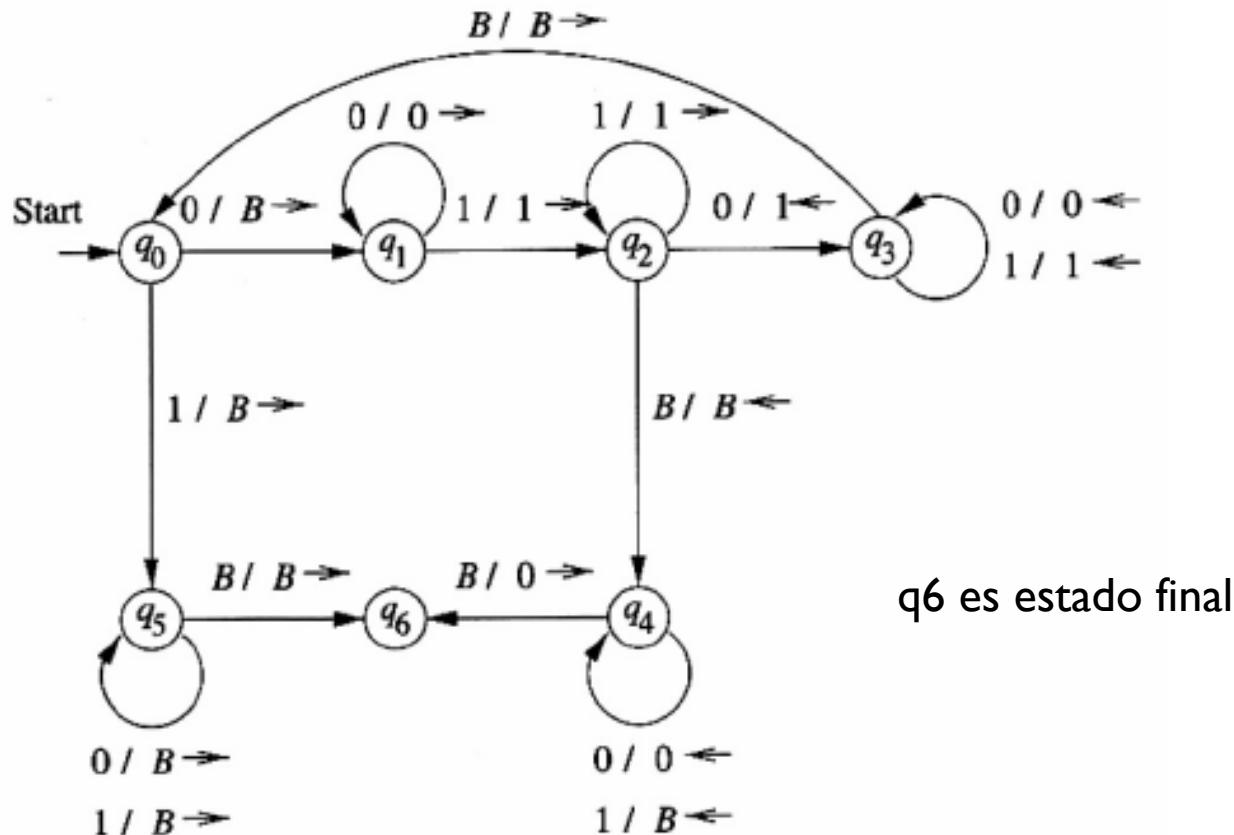


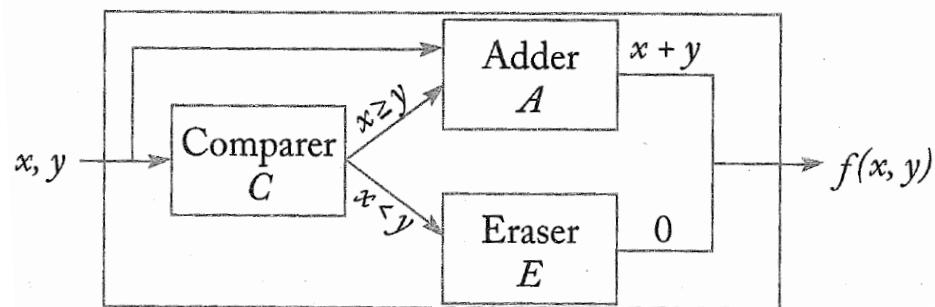
Diagrama de transición de la función *monus*

Computación de funciones

- Una función f con dominio D es Turing-computable o computable sin más, si existe una MT $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ tal que:
$$q_0 w \vdash^* q_f f(w), q_f \in F, \text{ para todo } w \in D$$
- Todas las funciones matemáticas comunes, no importa lo complicadas que sean, son Turing-computables
- Ejemplos:
 - MT que, dados dos enteros x e y en alfabeto unario, compute $x+y$
 - Contenido de la cinta: $w(x)0w(y) \vdash^* w(x+y)0$
 - MT que duplique cadenas de 1's
 - MT que realice la resta de dos números en alfabeto unario
 - Ejemplo de contenido inicial de la cinta: "111-11= "
 - Ejemplos de contenido final de la cinta: "1111-11=+11", "11-1111=-11", "11-11=+"

Combinación de MT

- Ejemplo: $f(x, y) = x + y \text{ si } x \geq y; f(x, y) = 0 \text{ si } x < y$



- Comparador:

- $q_{C,0}w(x)0w(y) \vdash^* q_{A,0}w(x)0w(y) \text{ si } x \geq y$
- $q_{C,0}w(x)0w(y) \vdash^* q_{E,0}w(x)0w(y) \text{ si } x < y$

- Sumador:

- $q_{A,0}w(x)0w(y) \vdash^* q_{A,f}w(x+y)$

- Borrador:

- $q_{E,0}w(x)0w(y) \vdash^* q_{E,f}0$

Tesis de Church-Turing

- Hipótesis: cualquier problema de decisión resoluble puede ser transformado en un problema equivalente para una MT
- Argumentos:
 - Cualquier problema que se pueda resolver en una computadora también se puede resolver con una MT
 - No se ha encontrado ningún problema resoluble (por un algoritmo) para el que no se pueda escribir un programa para una MT
 - Se han propuesto modelos alternativos de computación, pero ninguno ha probado ser más potente que el modelo de MT

Tesis de Church-Turing

- **Definición de algoritmo:** un algoritmo para una función $f: D \rightarrow R$ es una MT, la cual dada cualquier entrada $d \in D$ en su cinta, finalmente se para con la respuesta correcta $f(d) \in R$ en la cinta:

$$q_0 d \vdash^* q_f f(d), \quad q_f \in F \quad \text{y para todo } d \in D$$

- Usando la tesis de Church-Turing, podemos sustituir en la definición “MT” por “programa en Java”, “programa en C”, etc.

Otros modelos de MT

- Pequeñas variaciones:
 - MT con opción de no-movimiento
 - MT con cinta semiinfinita
 - MT con cinta de entrada
- Almacenamiento más complejo
 - MT multicinta
 - MT multidimensionales
- MT no deterministas
- Y más...

MT con opción de no-movimiento

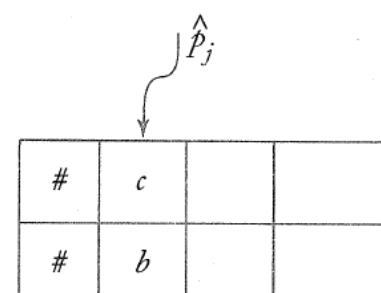
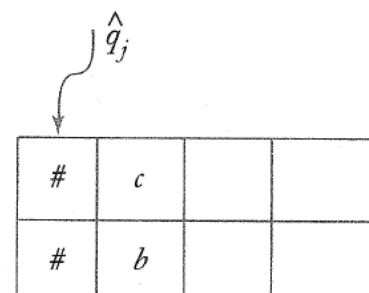
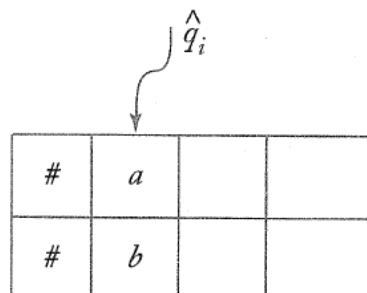
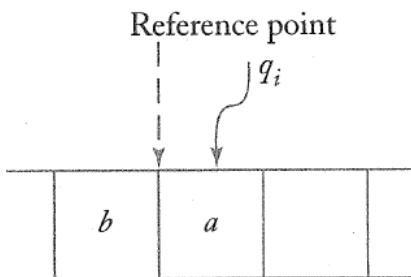
- $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{I, D, E\}$
 - E indica que la cabeza de lectura/escritura permanece estática
- Teorema: la clase de MT con opción de no-movimiento es equivalente a la clase de MT estándar
- Simulación de una MT con opción de no-movimiento (δ) con una MT (δ'):
 - Por cada $\delta(q_i, a) = (q_j, b, I \text{ o } D)$, se incluye $\delta'(q_i, a) = (q_j, b, I \text{ o } D)$
 - Por cada $\delta(q_i, a) = (q_j, b, E)$, se incluyen $\delta'(q_i, a) = (q_{jS}, b, D)$ y $\delta'(q_{jS}, c) = (q_j, c, I)$ (una transición por cada $c \in \Gamma$)

MT con cinta semiinfinita

- MT cuya cinta está limitada por un extremo
- Simulación de una MT estándar M por medio de una MT con cinta semiinfinita P:
 - Cinta de P con dos pistas
 - Pista superior: contenido de la cinta de M a la derecha de la referencia (situación inicial de la cabeza de M)
 - Pista inferior: contenido de la cinta de M a la izquierda de la referencia y en orden inverso
 - Los estados de P se dividen en dos conjuntos: un conjunto trabaja con la pista superior y otro con la inferior
 - Unos marcadores especiales (#) en el extremo izquierdo permiten cambiar de pista

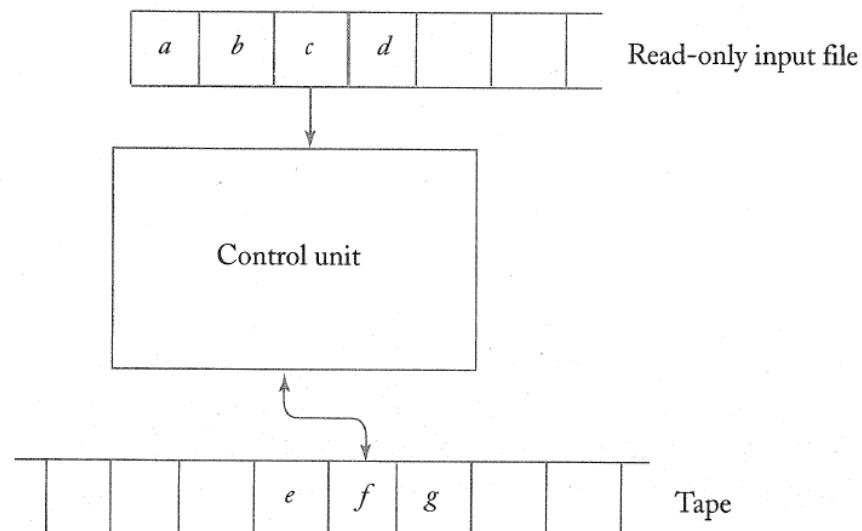
MT con cinta semiinfinita

- Ejemplo:
 - $\delta(q_i, a) = (q_j, c, l)$ es simulado con
 - $\delta'(q'_i, (a, b)) = (q'_j, (c, b), l)$ y $\delta'(q'_j, (\#, \#)) = (p'_j, (\#, \#), D)$
 - $\{q'_i, q'_j\} \in Q_U$ y $p'_j \in Q_L$



MT con cinta de entrada

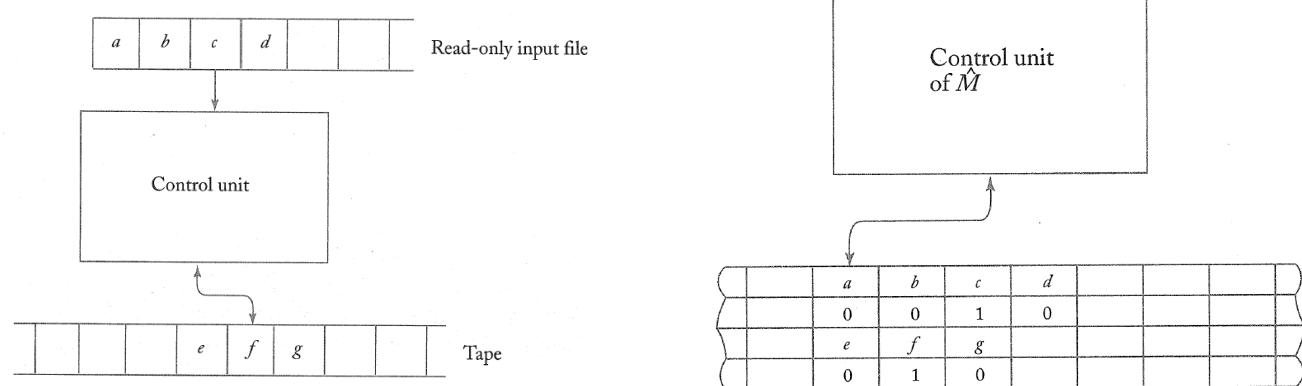
- La entrada está escrita en una cinta de sólo lectura
- Las transiciones se realizan en función del estado, el símbolo leído de la entrada y el símbolo leído por la cabeza de lectura/escritura en la cinta



- Simulación de una MT con una MT con cinta de entrada
 - Copiar el contenido de la cinta de entrada en la cinta

MT con cinta de entrada

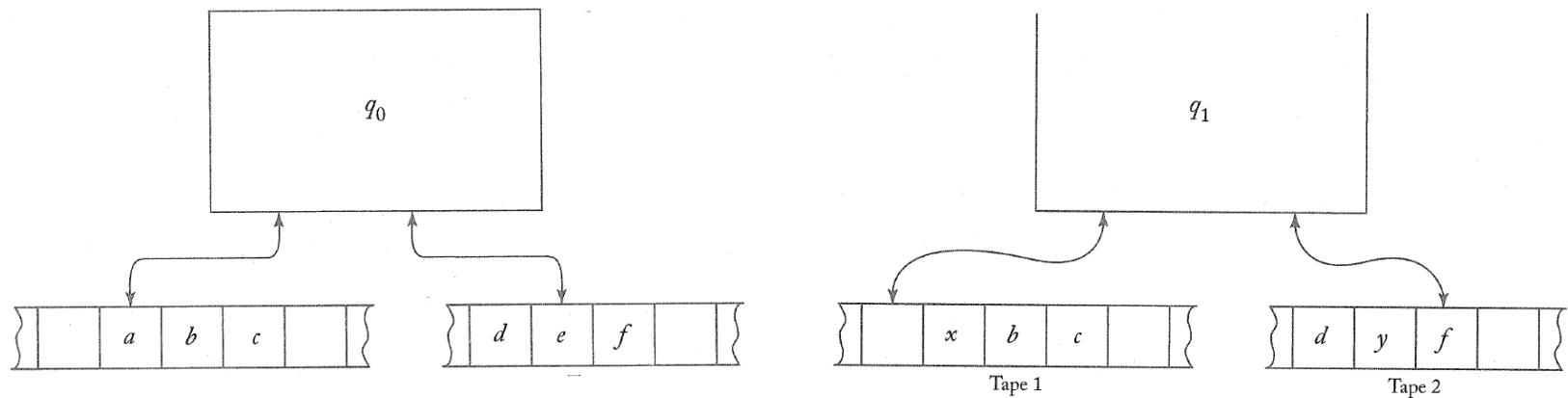
- Simulación de una MT con cinta de entrada M_1 con una MT M_2
 - Cinta con 4 pistas: valores de entrada, posición de la cabeza de lectura, contenido de la cinta y posición de la cabeza de lectura/escritura
 - Ejemplo:



- La simulación requiere varios movimientos en M_2 por cada movimiento en M_1
 - Posición de partida: extremo izquierdo de la cinta
 - Búsqueda de la posición de la cabeza de lectura en la pista 2
 - Lectura del símbolo correspondiente en la pista 1 y transición de estado
 - Búsqueda de la posición de la cabeza de lectura/escritura en la pista 4
 - Lectura del símbolo correspondiente en la pista 3 y transición de estado
 - Modificación de las pistas para representar el movimiento en M_1
 - Vuelta a la posición de partida para simular el siguiente movimiento

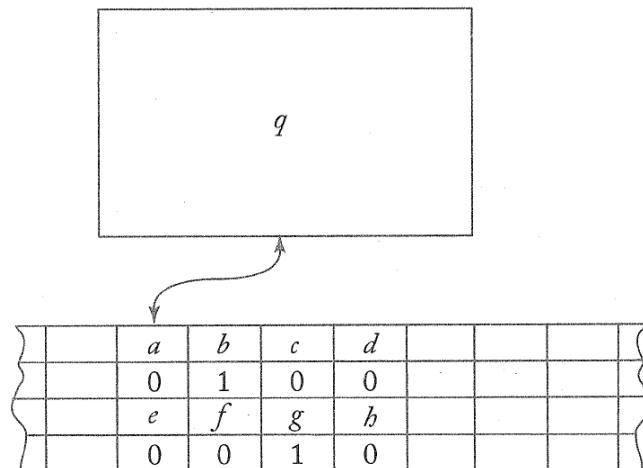
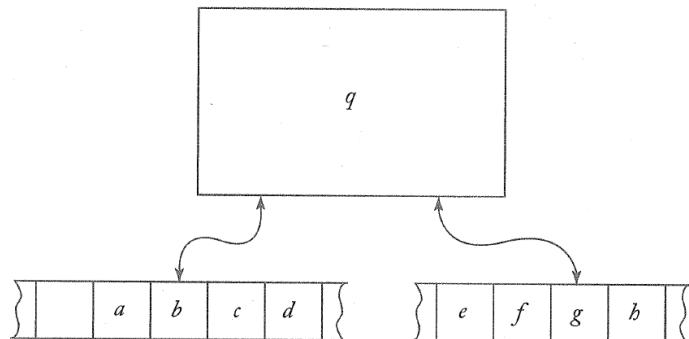
MT multicinta

- $\delta: Q \times \Gamma^n \rightarrow Q \times \Gamma^n \times \{I, D\}^n$
- Ejemplo: $\delta(q_0, a, e) = (q_1, x, I, D)$



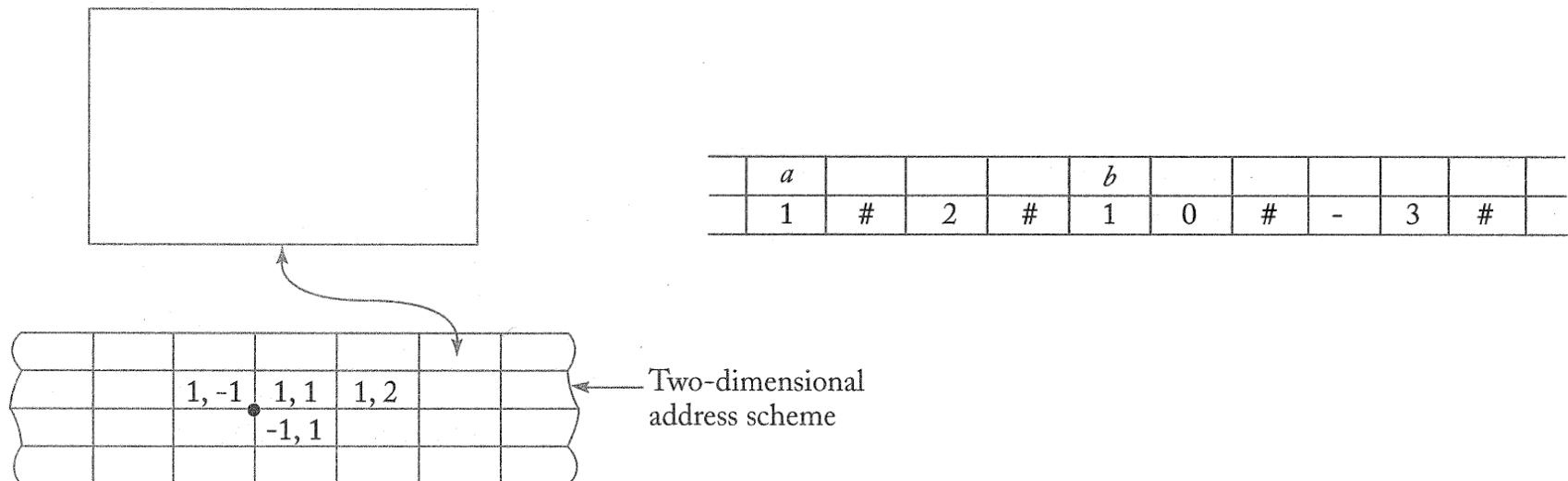
MT multicinta

- Simulación de una MT multicinta con una MT: uso de $2n$ pistas (n es el número de cintas de la MT multicinta)
 - Pistas impares: representan el contenido de las cintas
 - Pistas pares: representan la posición de la cabeza en las cintas
 - Los pasos a ejecutar son similares al de la simulación de MT con cinta de entrada
- Ejemplo:



MT multidimensional

- La cinta es infinita en más de una dimensión
- MT bidimensional: $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{I, D, AR, AB\}$
- Simulación de una MT bidimensional con una MT: cinta con dos pistas
 - Pista 1: almacena el contenido de la cinta bidimensional
 - Pista 2: contiene las direcciones asociadas al contenido de la pista 1
 - Para simular un movimiento, se busca en la pista 2 la dirección de la celda a la que se debe desplazar la cabeza de lectura/escritura



MT no determinista

- $\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{I, D\}}$
 - Ejemplo: $\delta(q_0, a) = \{(q_1, b, D), (q_2, c, I)\}$
- Una MT no determinista puede verse como una MT que puede replicarse a sí misma cuando sea necesario
 - Por cada posible transición se crea una réplica
- Simulación de una MT no determinista con una MT
 - Se crea una cinta con $2n$ pistas, donde n es el número de máquinas a simular
 - Cada nueva MT implica la inicialización de dos nuevas pistas
 - Una pista representa el contenido de la cinta y la otra el estado de la MT

MT no determinista

- Ejemplo: $\delta(q_0, a) = \{(q_1, b, D), (q_2, c, I)\}$

	#	#	#	#	#
	#	a	a	a	#
	#	q_0			#
	#	#	#	#	#

	#	#	#	#	#	#
	#		b	a	a	#
	#			q_1		#
	#		c	a	a	#
	#	q_2				#
	#	#	#	#	#	#

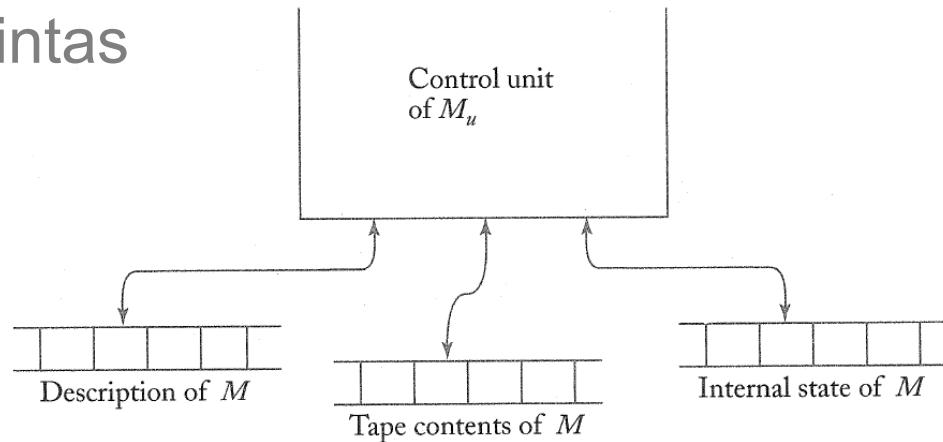
- MT deterministas y no deterministas son equivalentes

MT universal (MTU)

- Las MTU son reprogramables
- Dada una descripción de cualquier MT M y una cadena w, una MTU puede simular la computación de M para w
- Descripción de una MT:
 - $Q = \{q_1, q_2, \dots, q_n\}$, siendo q_1 el estado inicial y q_n el final
 - $\Gamma = \{a_1, a_2, \dots, a_m\}$, siendo a_1 un espacio en blanco
- Codificación de una MT
 - $q_1 = 1$ $q_2 = 11, \dots$
 - $a_1 = 1$ $a_2 = 11, \dots$
 - $I = 1$, $D = 11$
 - 0 es el símbolo separador
 - Ejemplo: $\delta(q_1, a_2) = (q_2, a_3, I)$ se codifica como 10110110111010

MT universal (MTU)

- Una MTU tiene tres cintas



- Funcionamiento:
 - Se examina el contenido de las cintas 2 y 3: configuración de M
 - Se consulta la cinta 1 para determinar la transición a realizar
 - Se modifican las cintas 2 y 3 como resultado del movimiento realizado
- Una MT, dado cualquier programa, puede realizar las computaciones especificadas y es, por tanto, un modelo adecuado de una computadora de propósito general
- Cualquier MT puede ser codificada con ceros y unos

Gramáticas sensibles al contexto

Gramáticas sin restricciones (GSR), $G = (NT, T, S, P)$

- Producciones:

$$\begin{aligned}x &\rightarrow y \\x &\in (NT/T)^+ \\y &\in (NT/T)^*\end{aligned}$$

- Las GSR generan los LRE

Gramáticas sensibles al contexto (GSC), $G = (NT, T, S, P)$

- Producciones:

$$\begin{aligned}\alpha &\rightarrow \beta ; |\alpha| \leq |\beta| \\ \alpha &= z_1 x z_2 \\ \beta &= z_1 y z_2 \\ z_1, z_2 &\in T^* \\ x &\in NT \\ y &\in (NT/T)^+\end{aligned}$$

- Un lenguaje L es sensible al contexto (LSC) si existe una GSC tal que $L = L(G)$ o $L = L(G) \cup \{\lambda\}$

Lenguajes sensibles al contexto

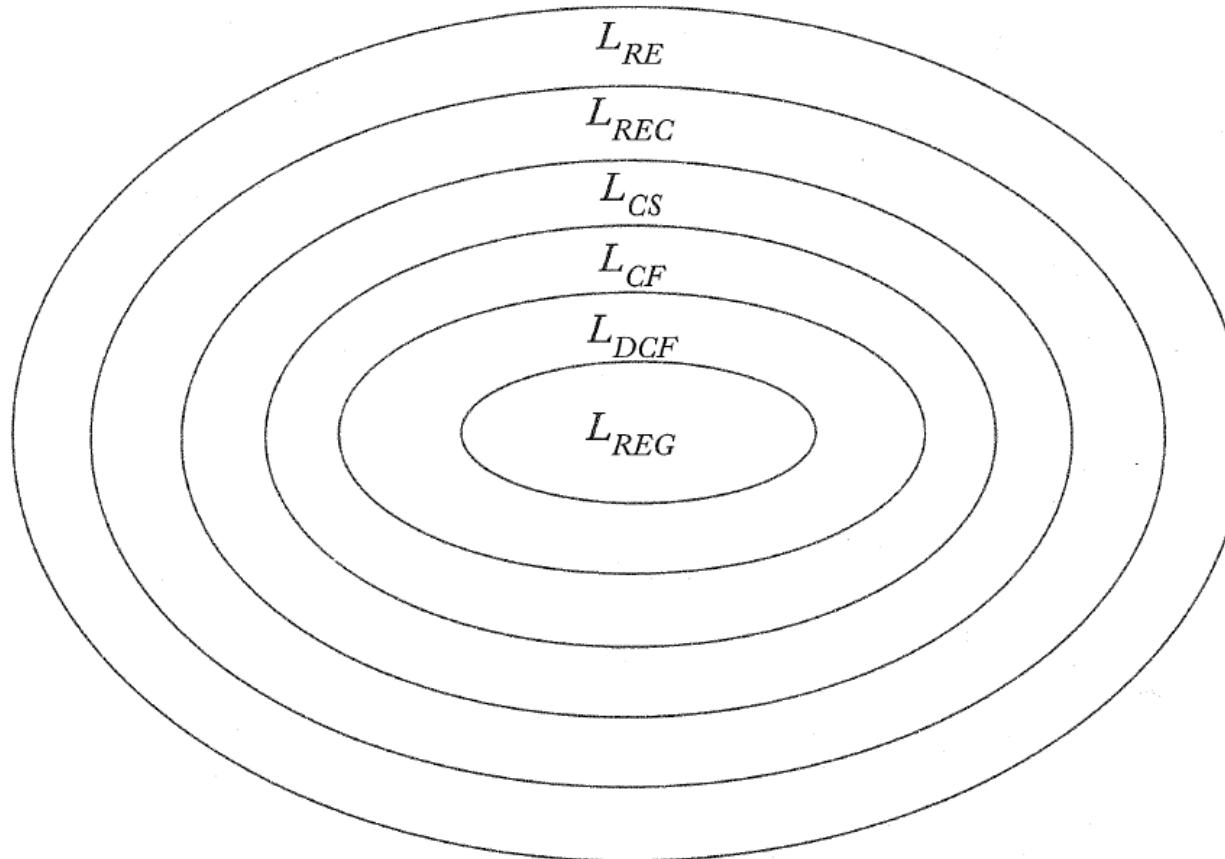
- Ejemplo: GSC que genera $L = \{a^n b^n c^n : n \geq 1\}$

$$\begin{array}{ll} S \rightarrow abc | aAbc, & S \Rightarrow aAbc \Rightarrow abAc \Rightarrow abBbcc \\ Ab \rightarrow bA, & \Rightarrow aBbbcc \Rightarrow aaAbbcc \Rightarrow aabAbcc \\ Ac \rightarrow Bbcc, & \Rightarrow aabbAcc \Rightarrow aabbBbcc \\ bB \rightarrow Bb, & \Rightarrow aabBbbccc \Rightarrow aaBbbbccc \\ aB \rightarrow aa | aaA. & \Rightarrow aaabbbccc. \end{array}$$

- Los LSC (que no contengan λ) son reconocidos por los Autómatas Linealmente Acotados (ALA)
- $\text{LSC} \subset \text{LREC}$

Autómatas linealmente acotados (ALA)

- Restricciones en el uso de la cinta
 - Funcionamiento tipo pila: autómata con pila
 - Uso de una parte finita de la cinta: autómata de estados finitos
 - Uso de la parte de la cinta ocupada por la cadena de entrada: ALA
 - Cuanta mayor longitud tenga la cadena de entrada, mayor es el espacio a utilizar
- Definición: un ALA es una **MT determinista** $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ con la restricción de que Σ debe contener dos símbolos especiales:
 - [: marcador izquierdo, con transiciones del tipo $\delta(q_i, [) = (q_j, [, D)$
 -]: marcador derecho, con transiciones del tipo $\delta(q_i,]) = (q_j,], I)$
- $L(M) = \{w \in \Sigma^+ : q_0[w] |-^* [x_1 q_f x_2], q_f \in F, x_1, x_2 \in \Gamma^*\}$
- Los ALA son más potentes que los AP y menos que las MT



- L_{REG} : Lenguaje Regular
- L_{DCF} : Lenguaje Determinista Independiente de Contexto
- L_{CF} : Lenguaje Independiente de Contexto
- L_{CS} : Lenguaje Dependiente de Contexto
- L_{REC} : Lenguaje Recursivo
- L_{RE} : Lenguaje Recursivamente enumerable

Lenguajes formales

Jerarquía de Chomsky

Decidibilidad y complejidad



Senén Barro Ameneiro, CiTIUS

@SenenBarro

Material elaborado fundamentalmente por
el profesor Manuel Mucientes Molina

Bibliografía

- J.E. Hopcroft, R. Motwani y J.D. Ullman,
"Teoría de Autómatas, Lenguajes y
Computación", Addison Wesley, 2008.
 - Capítulo 9 y 10
- P. Linz, "An Introduction to Formal Languages
and Automata", Jones and Bartlett Publishers,
Inc., 2001.
 - Capítulos 11, 12 y 14

Lenguajes recursivos y recursivamente enumerables

Un lenguaje L es **recursivamente enumerable** (LRE) si existe una MT que acepta cualquier cadena del lenguaje y se para:

- Si $w \in L$, $q_0 w \xrightarrow{*} x_1 q_f x_2$, $q_f \in F$

Sin embargo:

- Si $w \notin L$, la MT **se parará** en un estado no final – rechazándola- o **entrará en un bucle infinito**

Un lenguaje L sobre un alfabeto Σ es **recursivo** (LRC) si existe una MT que acepta L y se para con cualquier cadena $w \in \Sigma^+$ -**se para la acepte o no-**

Hay lenguajes que no son LRE. La demostración es compleja, puesto que todos los lenguajes descritos de forma algorítmica son aceptados por MT (son por tanto LRE)

Computabilidad y decibilidad

- Una función f es computable en un dominio si existe una MT que computa el valor de f para **todos los argumentos del dominio**
- Si el resultado de la computación de un problema es si/no, se habla de decidibilidad/indecidibilidad
- Problemas decidibles: existe una MT que da la respuesta correcta (si/no) para cada argumento del dominio

Problema de la parada en MT

Sea w_M una cadena que describe la MT M, y sea w una cadena sobre el alfabeto de M. Una solución al problema de la parada sería una MT H en la que, para cualesquiera w_M y w , se ejecuta la computación:

- $q_0 w_M w \xrightarrow{*} x_1 q_y x_2$, si M aplicada a w se para
 - $q_0 w_M w \xrightarrow{*} x_1 q_n x_2$, si M aplicada a w no se para
 - q_y y q_n son estados finales de H
-
- No existe ninguna MT H que se comporte como se requiere para el problema de la parada: problema indecidible
 - Si el problema de la parada fuese decidable, entonces todos los LRE serían LRC

Complejidad computacional

- Ejemplo: dado un vector de enteros, ¿cuánto tiempo se tardará en ordenarlo?
- Para determinar la complejidad
 - Usaremos MT
 - El tamaño del problema será n
 - Interesa saber cuánto aumenta el tiempo al incrementar n
- Si una computación tiene complejidad (de tiempo) $T(n)$ significa que puede ser resuelta en no más de $T(n)$ movimientos de una MT para un tamaño problema de n
- No nos interesará el tiempo exacto, pero sí el orden de magnitud: $O(\dots)$

MT y complejidad

Desde el punto de vista de la decidibilidad, todas las MT son equivalentes. Desde el punto de vista de la complejidad no

Ejemplo: $L = \{a^n b^n : n \geq 1\}$

- MT estándar: $O(n^2)$
- MT con dos cintas: $O(n)$

Problema de la satisfacibilidad (SAT)

- Expresiones en forma normal conjuntiva: $e = t_i \wedge t_j \wedge \dots \wedge t_k$
 - $t_i = s_m \vee s_p \vee \dots \vee s_q$: s_l son variables o sus negaciones
- Dada una expresión e en forma normal conjuntiva, ¿hay alguna asignación de valores a sus variables que haga e verdadera?
- Ejemplos: $e_1 = (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3)$, $e_2 = (x_1 \vee x_2) \wedge \neg x_1 \wedge \neg x_2$
- MT estándar: $O(2^n)$
- MT no determinista: $O(n)$

MT y complejidad

Una MT acepta un lenguaje L en tiempo $T(n)$ si cualquier cadena w de L ($|w| \leq n$) es aceptada en $O(T(n))$ movimientos

- Si la MT es no determinista, para cualquier cadena w de L existe al menos una secuencia de movimientos de longitud $O(T(|w|))$ que lleva a la aceptación
- Un lenguaje L pertenece a la clase $TD(T(n))$ si hay una MT **multicinta determinista** que acepta L en tiempo $T(n)$
- Un lenguaje L pertenece a la clase $TND(T(n))$ si hay una MT **multicinta no determinista** que acepta L en tiempo $T(n)$
 - $TD(T(n)) \subseteq TND(T(n))$

Complejidades P y NP

$$P = \bigcup_{i \geq 1} TD(n^i)$$

- Lenguajes aceptados por una MT determinista en tiempo polinómico

$$NP = \bigcup_{i \geq 1} TND(n^i)$$

- Lenguajes aceptados por una MT no determinista en tiempo polinómico

$$P \subseteq NP$$

○ ¿P = NP?

Problemas intratables: problemas computables pero que requerirían, para entradas grandes, tal cantidad de recursos (tiempo y memoria) que su implementación no es viable

- Los problemas de la clase P son tratables y el resto intratables (tesis de Cook-Karp)

Complejidades P y NP

Un lenguaje L_1 es reducible en tiempo polinómico a otro lenguaje L_2 si existe una MT determinista tal que cualquier cadena $w_1 \in \Sigma_1^+$ puede ser transformada en tiempo polinómico en otra cadena $w_2 \in \Sigma_2^+$ de tal forma que $w_1 \in L_1$ si y sólo si $w_2 \in L_2$

- Si L_1 es reducible en tiempo polinómico a L_2 y $L_2 \in P$, entonces $L_1 \in P$
- De igual forma, si $L_2 \in NP$, entonces $L_1 \in NP$

Un lenguaje L es NP-completo si $L \in NP$ y todo $L' \in NP$ es reducible en tiempo polinómico a L