Informática Teórica

- Segundo -

PROFESORES:

Juan Ríos

Alfonso Rodríguez-Patón

José Mª Barreiro

INFORMÁTICA TEÓRICA (200)

Curso: 2º (anual)

Carácter: Troncal

Créditos: 9

Profesorado:

José Mª Barreiro Sorrivas

Juan B. Castellanos Peñuela

Julio García del Real Ruizdelgado

Rafael Gonzalo Molina **(Coordinador)**

Juan Ríos Carrión

Alfonso Rodríguez-Patón Aradas

BREVE DESCRIPCIÓN

La asignatura se encuadra en el contexto de Teoría de la computación, y pretende describir los fundamentos teóricos de los ordenadores desde el punto de vista de la teoría de autómatas gramáticas y lenguajes. Es una ciencia multidisciplinar, pues se apoya, trata los mismos fenómenos desde áreas aparentemente desconectadas entre sí. De esta manera MÁQUINAS ABSTRACTAS Y ALGORITMOS, AÚTOMATAS Y MÁQUINAS SEQUENCIALES, GRAMÁTICAS Y LENGUAJES FORMALES, constituyen los tres eslabones que históricamente van a formar el cuerpo de la "INFORMÁTICA TEÓRICA".

Se sigue la jerarquía de Noam Chomsky, en la clasificación de los Lenguajes y Gramáticas, estableciéndose a continuación los correspondientes autómatas, de manera que:

Se desarrollan los lenguajes tipo 3, generados por las gramáticas tipo 3, lineales izquierdas o derechas, ambas equivalentes, y que se corresponden con los CONJUNTOS REGULARES, dados por las EXPRESIONES REGULARES, capaces de simbolizar conjuntos infinitos mediante especificaciones finitas; a estos lenguajes les corresponden cierto tipo de autómatas, deterministas y no deterministas – equivalentes ambos – con los que se es capaz de resolver ciertos problemas de índole menor desde el punto de vista matemático. Los lenguajes tipo 2 siguen a los anteriores y son generados por las gramáticas tipo 2, "INDEPENDIENTES DEL CONTEXTO" que resuelven problemas de mayor envergadura, y se corresponden con los "AUTÓMATAS A PILA", que a diferencia de los anteriores necesitan de una pila de memoria adicional. Se sigue con los lenguajes tipo1, " DEPENDIENTES DEL CONTEXTO" a cuyas gramáticas generativas se les exige menos restricciones, y que se corresponden con los autómatas acotados linealmente. Por último Se desarrollan los lenguajes tipo 0, generados por las gramáticas tipo 0 "SIN RESTRICCIONES" isomórficas con las "MÁQUINAS DE TURING", que resuelven problemas recursivamente enumerables. Se describen brevemente algunos problemas "no enumerables" que las máquinas deTuring no son capaces de resolver. Se finaliza el temario con una breve descripción de "LAS REDES DE NEURONAS", autómatas capaces de simular – en alguna medida – el comportamiento del sistema neuronal humano.

TEMARIO

CAPÍTULO 1: Lenguajes Formales.

CAPÍTULO 2: Gramáticas Formales.

CAPÍTULO 3: Máquinas Secuenciales.

CAPÍTULO 4: Autómatas Finitos.

CAPÍTULO 5: Lenguajes Regulares.

CAPÍTULO 6: Propiedades de los Lenguajes Regulares.

CAPÍTULO 7: Autómatas de Pila.

CAPÍTULO 8: Propiedades de los Lenguajes Independientes del Contexto.

CAPÍTULO 9: Máquinas de Turing.

CAPÍTULO 10: Redes de Neuronas Artificiales.

BIBLIOGRAFÍA

MACHINES, LANGUAGES AND COMPUTATION (P.J. Denning, J.B. Dennis, J.E. Qualitz.

Editorial Prentice Hall, 1978)

TEORIA DE AUTOMATAS Y LENGUAJES FORMALES. (Dean Kelly. Prentice Hall, 1995.)

INFORMÁTICA II (J.J.. Scala, J.M. Minguet. Editorial UNED 1974)

INTRODUCTION TO AUTOMATA THEORY, LANGUAGES AND COMPUTATION. (J.E.

Hopcroft, J.D. Ullman. Editorial Addison-Wesley 1979.)

FUNDAMENTOS DE INFORMÁTICA. (G.Fernández, F. Sáez Vacas. Editorial Alianza

Informática. Alianza Editorial 1987)

ELEMENTS OF THE TEORY OF COMPUTATION. (H.R. Lewis, C.H. Papadimitriou.

Editorial Prentice Hall 1981)

LENGUAJES, GRAMATICAS Y AUTOMATAS. Un enfoque Práctico. (P. Isasi., P.

Martínez, D. Borrajo. Addison-Wesley, 1997)

ESTRUCTURA DINÁMICA Y APLICACIONES DE R.N.A. (J. Ríos y otros. Editorial Centro

de Estudios Ramón Areces 1991)

Los libros referenciados son "exclusivamente recomendados", no constituyendo por lo tanto, elementos de ningún tipo con respecto a exámenes. En este sentido sólo será responsabilidad de los profesores de la Cátedra la materia explicada en clase.

NORMAS PARA LA EVALUACIÓN DE LA ASIGNATURA

FORMA DE EVALUACIÓN

Los exámenes versarán sobre lo explicado en las clases de Teoría y Prácticas.

Examen parcial de febrero:

El examen se realizará sobre el programa desarrollado hasta la última clase impartida antes de este

examen.

Examen parcial y final de junio:

Se realizará el mismo día y la opción se decidirá por el propio alumno, antes de comenzar el examen.

Para poder presentarse únicamente al segundo parcial es condición necesaria la obtención de una puntuación mínima de 15 puntos (3 sobre 10) en el Primer Parcial de febrero.

Examen de septiembre:

Será en único examen. El alumno deberá examinarse de toda la asignatura.

PUNTUACIONES.

Cada uno de los exámenes parciales de febrero y junio tendrán una valoración máxima de 50 puntos.

El examen de septiembre tendrá una valoración máxima de 100 puntos.

El aprobado exige una puntuación mínima igual a la mitad de la máxima (50 puntos).

Compensaciones.

Para poder presentarse en el examen de junio únicamente al segundo parcial, es condición necesaria la obtención de una puntuación mínima de 15 puntos (3 sobre 10) en el primer parcial de febrero.

Para el examen de septiembre, no se guardarán notas de 1º y 2º parciales de junio y, por tanto, el alumno deberá examinarse de toda la asignatura.

REVISIÓN DE EXÁMENES

Todos los exámenes son considerados oficiales y por tanto con derecho a revisión.

Para revisar algún ejercicio se entregará en la Secretaría del Departamento la solución correcta del mismo, así como los motivos razonados por los que se solicita la revisión.

Posteriormente se harán públicas las posibles modificaciones a que hubiera lugar, concretándose fecha o fechas para ver el examen correspondiente.

(29/9/05)

TEMA 1 LENGUAJES FORMALES

1. DEFINICIONES.

- **ALFABETO:** Un alfabeto es un conjunto finito, no vacío, de elementos llamados "símbolos del alfabeto".
  Lo representamos por: Σ (sigma)

  EJ: $\Sigma_1 = \{0,1\}$ $\quad 0 \in \Sigma_1$
  $\Sigma_2 = \{a, b, c\}$
  $\Sigma_3 = \{$ árbol, coche, .. $\}$

- **PALABRA:** Una palabra definida sobre un alfabeto es una concatenación finita de símbolos de dicho alfabeto.
  Se representan con letras minúsculas: x, y, z, u, v, w, ...

  EJ. $\Sigma_1 = \{0,1\}$
  Palabras sobre Σ_1: $x = 0$ $\quad z = 0111$ $\quad v = 01$
  $y = 1$ $\quad u = 101$ $\quad w = \lambda$
  $\Sigma_2 = \{A, B, .. Z\}$
  Palabra sobre Σ_2. $x =$ PEDRO

  * PALABRA VACÍA → Palabra cuya longitud es 0. Pertenece a todos los alfabetos.
  Se representa por λ (lambda minúscula).

- **LONGITUD DE PALABRA:** Nº de símbolos que tiene esa palabra.

  EJ. $x = 0111$ $\quad |x| = 4$
  $y = \lambda$ $\quad |y| = 0$
  $|COCHE| = 5$

- **LENGUAJE UNIVERSAL**: Llamamos lenguaje universal de Σ al conjunto de todas las palabras definidas sobre el alfabeto Σ; es decir, el conjunto de todas las palabras que se pueden formar con los símbolos del alfabeto Σ.

Se representa por: Σ^*

Cualquiera que sea el alfabeto Σ, $\lambda \in \Sigma^*$ siempre.

Aunque Σ es un conjunto finito, Σ^* es un conjunto infinito.

Ej: $\Sigma = \{0\}$ $\Sigma^* = \{\lambda, 0, 00, 000, 0000, \dots\}$

$\Sigma = \{0,1\}$ $\Sigma^* = \{\lambda, 0, 1, 00, 01, 10, 11, \dots\}$

2. OPERACIONES CON PALABRAS.

- **CONCATENACIÓN**: Es la operación mediante la cual a partir de 2 palabras se obtiene otra palabra conseguida al poner una palabra detrás de otra.

$$\begin{cases} \Sigma^* \times \Sigma^* \longrightarrow \Sigma^* \\ (x,y) \longrightarrow xy = x.y \end{cases} \qquad x, y, xy \in \Sigma^*$$

Ej: $x = 0111$ $y = 101$ $xy = x.y = 0111101$

- **PROPIEDADES.-**

1) Es una operación cerrada:

$$\left.\begin{array}{l} x \in \Sigma^* \\ y \in \Sigma^* \end{array}\right\} \rightarrow xy \in \Sigma^*$$

2) Es asociativa: $x(yz) = (xy)z \quad \forall x, y, z \in \Sigma^*$

3) NO es conmutativa: $xy \neq yx$

4) $\exists$ elemento neutro: $\lambda x = x\lambda = x$

Por cumplir las propiedades anteriores, la operación de concatenación de las palabras de un alfabeto es un monoide (semigrupo con elemento neutro).

La longitud de una concatenación es la suma de las longitudes de las palabras constituyentes:

$$\boxed{|xy| = |x| + |y|}$$

LEY DE EXCLUSIÓN: Sean x, y, z palabras, entonces: $\begin{cases} xy = xz \longrightarrow y = z \\ xy = zy \longrightarrow x = z \end{cases}$

Dadas 2 palabras $u, w \in \Sigma^*$ decimos que u es SUBPALABRA de w si $\exists$ 2 palabras $x, y \in \Sigma^*$ tales que: $w = xuy$

- POTENCIA DE UNA PALABRA: Se llama potencia i-ésima de una palabra a la concatenación consigo misma i veces.

$$x^i = \underbrace{xxx \ldots x}_{i}$$

$\boxed{x^0 = \lambda}$ $\quad x^{i+1} = x^i x \quad (i>0)$ $\quad$ LONGITUD: $|x^i| = i \cdot |x|$

$|x^1 = x|$ $\quad \boxed{x^i x^j = x^{i+j}}, \quad (i,j>0)$

EJ: $\Sigma_1 = \{0,1\}$ $\quad x = 01$ $\quad x^3 = 01\,01\,01$

$\Sigma_2 = \{A,B,C\}$ $\quad x = ABC$ $\quad x^2 = ABCABC$

EJ: Potencias de longitud 2 de las palabras definidas sobre Σ.

$\Sigma = \{a, b\}$

long = 2 $\longrightarrow$ $i=2$, $|x| = 1$ $\longrightarrow$ aa, bb

$\hookrightarrow$ $i=1$, $|x| = 2$ $\longrightarrow$ aa, bb, ab, ba

- PALABRA INVERSA: La palabra inversa $\boxed{x^{-1}}$ de una palabra x es aquella palabra cuyos símbolos del alfabeto están en orden inverso.

Si $x = x_1 x_2 \ldots x_n \longrightarrow x^{-1} = x_n \ldots x_2 x_1$

Definición recursiva

$$\begin{cases} |x| = 0 \longrightarrow x = \lambda \longrightarrow x^{-1} = \lambda^{-1} = \lambda \\ |x| = 1 \longrightarrow x = a \longrightarrow x^{-1} = a^{-1} = a \\ |x| > 0 \longrightarrow x = ay \longrightarrow x^{-1} = y^{-1}a \\ \qquad\qquad\quad x = ya \longrightarrow x^{-1} = ay^{-1} \end{cases}$$

EJ: $x = aaab \qquad x^{-1} = baaa$

EJERCICIOS: Comprobar que:

a) $(wx)^{-1} = x^{-1}w^{-1}$ (Se demuestra por inducción)

$|x| = 0 \longrightarrow x = \lambda \quad (wx)^{-1} = (w\lambda)^{-1} = w^{-1} = \lambda w^{-1} = \lambda^{-1}w^{-1} = x^{-1}w^{-1}$

$|x| = n+1 \longrightarrow x = ya \mid y \in \Sigma^* \quad |y| = n$
$\qquad\qquad\qquad\qquad\qquad\quad a \in \Sigma$

Hipótesis de inducción $\Rightarrow (wy)^{-1} = y^{-1}w^{-1}$ → lo supongo cierto.

$$(wx)^{-1} = (w(ya))^{-1} \underset{\text{asociativa}}{=} ((wy)a)^{-1} \underset{\text{definición}}{=} a(wy)^{-1} \underset{\text{hip. induc.}}{=} ay^{-1}w^{-1} \underset{\text{asociativa}}{=} (ay^{-1})w^{-1} =$$

$$\underset{\text{definición}}{=} (ya)^{-1}w^{-1} = x^{-1}w^{-1}$$

b) $(w^{-1})^{-1} = w$

$|w| = 0 \longrightarrow w = \lambda \quad (w^{-1})^{-1} = (\lambda^{-1})^{-1} = \lambda^{-1} = \lambda = w$

$|w| > 0 \quad |w| = n+1 \longrightarrow w = ya \mid y \in \Sigma^* \quad |y| = n$ Hipótesis de inducción $\Rightarrow (y^{-1})^{-1} = y$
$\qquad\qquad\qquad\qquad\qquad\quad a \in \Sigma$

$$(w^{-1})^{-1} = ((ya)^{-1})^{-1} \underset{\text{definición}}{=} (ay^{-1})^{-1} \underset{\text{def.}}{=} (y^{-1})^{-1}a \underset{\text{hip. induc.}}{=} ya = w$$

c) $(w^n)^{-1} = (w^{-1})^n$

4º caso → $n = 0 \quad (w^n)^{-1} = (w^0)^{-1} = \lambda^{-1} = \lambda = (\lambda^{-1})^0 = (w^{-1})^0 = \lambda$

2º caso → $n = n+1 \quad (w^{n+1})^{-1} = (w^n \cdot w)^{-1} = w^{-1}(w^n)^{-1} = w^{-1}(w^{-1})^n = (w^{-1})^{1+n}$ Se cumple.

3 LENGUAJES

- **LENGUAJE:** Un lenguaje definido sobre un alfabeto Σ es un subconjunto del lenguaje universal Σ^*.

Es también el conjunto de palabras de Σ^* que cumplen una determinada condición.

$$L \subseteq \Sigma^*$$

Ej: $\Sigma = \{0,1\} \longrightarrow L = \{01^n \ , \ n>0\}$

$$L = \{ x \in \{0,1\}^* : x = 01^n \ , n>0 \}$$
$$\Sigma^*$$

$L = \{a^n b^m \ , \ n \leq m \leq 3n \ , \ n>0\}$

$n=1 \ , \ 3n=3 \longrightarrow m = \{1,2,3\} \longrightarrow ab, abb, abbb$

$n=2 \ , \ 3n=6 \longrightarrow m = \{2,..,6\} \longrightarrow aabb, aabbb, aabbbb, aabbbbb, ...$

$L = \{ x \ / \ |x| = 1 \} \longrightarrow$ Palabras de longitud 1 = El propio alfabeto.

* El alfabeto es el lenguaje cuyas palabras son de longitud 1.

$\Sigma = \{0,1,2\}$

$\Sigma^* = \{ \lambda, 0, 1, 2, \begin{matrix} 00 & 10 & 20 \\ 01 & 11 & 21 \\ 02 & 12 & 22 \end{matrix}, \begin{matrix} 000 \\ 001 \\ 002 \end{matrix} \ ... \}$

- **LENGUAJE VACÍO** → Lenguaje que no tiene ninguna palabra.

Su cardinal es 0 → $c(\phi) = 0$ Cardinal $\phi = 0$

Se representa por: $\emptyset$

* El lenguaje vacío ϕ no debe confundirse con el lenguaje que contiene únicamente la palabra vacía λ.

$c(\phi) = 0$

$c(\{\lambda\}) = 1$

Tanto ϕ como $\{\lambda\}$ son lenguajes sobre cualquier alfabeto.

Como el universo Σ^* asociado al alfabeto Σ es infinito, hay infinitos lenguajes asociados a un alfabeto.

4 OPERACIONES CON LENGUAJES.

UNIÓN: Dados dos lenguajes $\left.\begin{array}{l} L_1 \subseteq \Sigma^* \\ L_2 \subseteq \Sigma^* \end{array}\right|$ definimos el **LENGUAJE UNIÓN** como el conjunto de palabras x tales que $x \in L_1$ ó $x \in L_2$.

$$\left.\begin{array}{l} L_1 \subseteq \Sigma^* \\ \vdots \\ L_2 \subseteq \Sigma^* \end{array}\right| \longrightarrow \boxed{L_1 \cup L_2 = \{ x \in \Sigma^* : x \in L_1 \text{ ó } x \in L_2 \}}$$

- **PROPIEDADES.-**

1) Es una operación **cerrada**: La unión de dos lenguajes sobre el mismo alfabeto es también un lenguaje sobre dicho alfabeto.

2) Es **asociativa**: $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$

3) Es **conmutativa**: $L_1 \cup L_2 = L_2 \cup L_1 \quad \forall L_1, L_2$

4) $\exists$ elemento neutro: $\phi \cup L = L \cup \phi = L$

5) Propiedad **idempotente**: $L \cup L = L \quad \forall L$

Por cumplir las propiedades anteriores, la unión de lenguajes es un **monoide**.

CONCATENACIÓN: Dados dos lenguajes definidos sobre el mismo alfabeto, llamamos concatenación de lenguajes a:

$$\left.\begin{array}{l} L_1 \in \Sigma^* \\ L_2 \in \Sigma^* \end{array}\right| \longrightarrow \boxed{L_3 = L_1 L_2 = \{ w = xy \in \Sigma^* \ / \ x \in L_1 \wedge y \in L_2 \}}$$

$L_1 \cdot L_2$

$$\boxed{\phi L = L \phi = \phi}$$

Todas las palabras del lenguaje resultante se forman concatenando una palabra del primer lenguaje con otra del segundo.

(A la concatenación también se le puede llamar producto)

· **PROPIEDADES.-**

1) Es una operación **cerrada**: la concatenación de dos lenguajes sobre el mismo alfabeto es otro lenguaje sobre el mismo alfabeto.

2) Es **asociativa**: $L_1(L_2L_3) = (L_1L_2)L_3$

3) $\exists$ elemento neutro: El lenguaje de la palabra vacía:

$$\{\lambda\}L = L\{\lambda\} = L$$

Por cumplir las tres propiedades anteriores, la concatenación de lenguajes es un **monoide**.

-**POTENCIA DE UN LENGUAJE**: Se llama potencia i-ésima de un lenguaje a la operación que consiste en concatenarlo consigo mismo i veces.

$$L^i = \underbrace{LL\ldots L}_{i}.$$

$L^0 = \{\lambda\}$ → Lenguaje de la palabra vacía.

$L^1 = L$

$L^{i+1} = L^iL = LL^i \quad (i>0)$

$L^iL^j = L^{i+j} \quad (i,j>0)$

(3/10/05)

5. CIERRE. CLAUSURA. ESTRELLA DE KLEENE: (¡IMPORTANTE)

Definimos el **CIERRE** o **CLAUSURA** de un lenguaje como la unión de todas las potencias del lenguaje, incluida la potencia 0. Lo llamamos **ESTRELLA DE KLEENE**.

$$L^* = L^0 \cup L^1 \cup \ldots \cup L^n \cup \qquad L^* = \bigcup_{i=0} L^i$$

Todas las clausuras contienen la palabra vacía.

$L^+ = L^+ \cup \{\lambda\}$

$L^+ = LL^* = L^*L$ ← concatenación.

Puesto que el alfabeto Σ es también un lenguaje L sobre Σ, se le puede aplicar esta operación:

Si $L = \Sigma \longrightarrow L^* = \Sigma^* \longrightarrow$ El cierre del lenguaje es igual al lenguaje universal.

EJ: $\Sigma = \{0,1\}$

$L(\Sigma) = \{0,1\}$

$L(\Sigma) = \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$

EJ: $\Sigma = \{a, b, c\} = L$

$L^* = \Sigma^* = \{a, b, c\}^* = \{\{a\} \cup \{b\} \cup \{c\}\}^*$

Una palabra w pertenece al cierre de un lenguaje si ocurre que esa palabra la puedo subdividir en palabras consecutivas $w_1 w_2 \dots w_k$ y dichas palabras pertenecen al lenguaje.

• DEF. 2 - El cierre de un lenguaje L es el conjunto de palabras que cumplen la propiedad anterior.

Formalmente:

$$L^* = \{ w \in \Sigma^* \,/\, w = w_1 w_2, \dots, w_k \text{ para un } k \geq 0 \text{ y } w_1, w_2, \dots, w_k \in L \}$$

EJ: $L = \{0, 01, 100\}$

$L^* = \{\lambda, 01, 100, 001, 010, 0100, 1000, 0101, \dots\}$

$w = 0100010100$ ¿ $w \in L^*$? $0100010100 \longrightarrow w \in L^*$

Solo pertenecerán a L^* uniones de palabras que pertenezcan a L.

CLAUSURA POSITIVA:

La clausura positiva de un lenguaje L es el lenguaje que se forma por la unión de todas las potencias de L, excepto L^0.

$$L^+ = L \cup L^2 \cup \dots \cup L^n \cup \dots \qquad \boxed{L^+ = \bigcup_{i=1}^{\infty} L^i}$$

Obviamente, ninguna clausura positiva contiene la palabra vacía si L no la contiene $\longrightarrow$ $\boxed{\lambda \in L^+ \iff \lambda \in L}$

EJ: $\Sigma = \{0,1\}$

$L(\Sigma) = \{0,1\}$ $\qquad L^+ = \{0, 1, 00, 01, 10, 11, 000, 001, \dots\}$

Puesto que el alfabeto Σ es también un lenguaje sobre Σ, se le puede aplicar esta operación:

$$\text{Si } L(\Sigma) = \Sigma \longrightarrow L^+ = \Sigma^* - \{\lambda\}$$

(Σ^*: potencia 0 de un alfabeto)

Una palabra w pertenece a la clausura positiva de un lenguaje si ocurre que esa palabra la puedo subdividir en palabras consecutivas $w_1, w_2, \dots, w_k$ $(k>0)$ y dichas palabras pertenecen al lenguaje.

• DEF. 2- La <u>clausura positiva</u> de un lenguaje L es el conjunto de palabras que la propiedad anterior.

Formalmente:

$$\boxed{L^+ = \{ w \in \Sigma^* / \ w = w_1, w_2, \dots, w_k \text{ para } k>0 \text{ y } w_1, w_2, \dots, w_k \in L\}}$$

$$\boxed{L^* = L^+ \cup \{\lambda\}} \qquad \boxed{L^+ = LL^* = L^*L} \ (\text{concatenación}) \Big] \rightarrow (L^*L) + \{\lambda\} = L^*$$

EJ: $L = \phi$ $\qquad \phi^* = \{\lambda\} = \phi^0 \cup \phi^1 \cup \phi^2 \cup \dots \cup \phi^n \cup \dots$

$\phi^0 = \{\lambda\}$ (prop 1.) $\quad \phi^1 = \phi \quad \phi^2 = \phi\phi = \phi$ (prop 2) $= \phi$

prop 1.- $L^0 = \{\lambda\}$

prop 2.- $L\phi = \phi L = \phi$

prop 3.- $L \cup L = L$

$\phi \cup L = L$

LENGUAJE INVERSO:

El lenguaje inverso de un lenguaje L dado está formado por la aplicación de inversión a cada una de las palabras del lenguaje L.

$$L^{-1} = \{ w \in \Sigma^* / w = u^{-1}, u \in L \}$$

EJ. $\Sigma = \{0, 1\}$

$L(\Sigma) = \{0, 1, 00, 10\}$ $\qquad$ $L(\Sigma)^{-1} = \{0, 1, 00, 01\}$

INTERSECCIÓN DE 2 LENGUAJES:

Dados los lenguajes L_1 y L_2, su intersección $L_1 \cap L_2$ será el conjunto que contenga las palabras que pertenezcan a los dos lenguajes.

$$L_1 \cap L_2 = \{ x \in \Sigma^* / x \in L_1 \wedge x \in L_2 \}$$

EJ: $\Sigma = \{0, 1\}$

$L_1(\Sigma) = \{00, 01, 10\}$ $\qquad$ $L_1 \cap L_2 = \{01, 10\}$

$L_2(\Sigma) = \{000, 10, 001, 11, 01\}$

COMPLEMENTACIÓN:

Otra operación posible sería la complementación con respecto al lenguaje universal. La complementación de un lenguaje es el conjunto de palabras que pertenecen al lenguaje universal que son complementarias.

$$L = \Sigma^* - L \qquad x \in \Sigma^* - L$$

6. PRODUCCIONES.

Sea un alfabeto Σ.

Llamamos **producción** al par ordenado de palabras (x, y) definidas sobre el alfabeto.

$$(x, y) \quad \begin{cases} x \in \Sigma^* \\ y \in \Sigma^* \end{cases} \qquad \boxed{x ::= y} \quad (x \text{ produce } y)$$

Se dice que "x" es la parte izquierda de la producción e "y" la parte derecha.
Las producciones se llaman también reglas de derivación.

Ej: $\Sigma = (0,1)$

En Σ se podrían tener las siguientes producciones: $(000 ::= 010)$

$(10 ::= 01)$

- DERIVACIÓN DIRECTA:

Sea Σ un alfabeto y $x ::= y$ una producción sobre las palabras de ese alfabeto. Dadas dos palabras v y w del mismo alfabeto $(v, w \in \Sigma^*)$.

Se dice que "w es derivación directa de v" o que "v produce directamente w" $(v \to w)$ si existen dos palabras $\boxed{u, z \in \Sigma^*}$ tales que $\begin{cases} v = uxz \\ w = uyz \end{cases}$

Ej: Σ = Alfabeto de las letras mayúsculas = $\{A, B, \ldots Z\}$

$x ::= y \in P \qquad \underset{\substack{\| \\ x}}{BA} ::= \underset{\substack{\| \\ y}}{ME} \qquad BA \to ME \qquad \underline{CABALLO} \to \underline{CAMELLO}$

$u \; x \; z \qquad u \; y \; z$

$v = CABALLO$

$w = CAMELLO$

Si $u = z = \lambda \longrightarrow v = x, \; w = y$

- DERIVACIÓN:

Sea Σ un alfabeto y P un conjunto de producciones sobre las palabras de dicho alfabeto. Dadas dos palabras $v, w \in \Sigma^*$, se dice que "w es derivación de v" o que "v produce w" $\boxed{v \to^+ w}$ si existe una secuencia de n derivaciones directas tales que:

$v = u_0$
$u_0 \to u_1$
$u_1 \to u_2$
...
$u_{n-1} \to u_n$
$w = u_n$

Derivación de longitud n.

$v \to^+ w$

* Si $v \to w$, $v \to^+ w$ mediante una secuencia de longitud 1.

EJ: $\Sigma = \{CN, C, NN, 0, 1, N\}$

P [
$C ::= CN$
$CN ::= NN$
$N ::= 0$
$N ::= 1$
$CN ::= 0 \mid 1$

$N ::= 0 \mid 1 \to N$ produce 0 y 1.

$C \to CN$
$CN \to NN$
$NN \to ON$
$CN \to 00$

$C \to^+ 00$

$C \to CN$
$CN \to ON$
$CN \to 00$

$C \to^+ 00$

-DERIVACIÓN IZQUIERDA: En cada derivación directa se utiliza la producción aplicada a los símbolos más a la izquierda de la palabra.

-DERIVACIÓN DERECHA: En cada derivación directa se utiliza la producción aplicada a los símbolos más a la derecha de la palabra.

EJ: $\Sigma = \{0, 1\}$

$P_1 = (000 ::= 010)$

$P_2 = (10 ::= 01)$

Der. izquierda $\to$ $1000 \xrightarrow{P_2} 0100 \xrightarrow{P_2} 0010 \xrightarrow{P_2} 0001 \xrightarrow{P_1} 0101 \xrightarrow{P_2} 0011$

Der derecha $\to$ $1000 \xrightarrow{P_1} 1010 \xrightarrow{P_2} 1001 \xrightarrow{P_2} 0101 \xrightarrow{P_2} 0011$

EJERCICIOS

① - $\Sigma = \{0,1\}$ $\quad L = \{x \in \Sigma^* : x \text{ termina en } 1\}$

todas las cadenas que se pueden formar con el 0 y el 1.

$L = \{\{0\} \cup \{1\}\}^* \{1\} = \{0,1\}^* \{1\} = \{0 \cup 1\}^* \{1\}$

todos los posibles 0s y 1s

② - $\Sigma = \{0,1\}$ $\quad L = \{x \in \Sigma^* : \text{tal que el } 111 \text{ pertenezca al lenguaje}\}$

$L = \{\{0\} \cup \{1\}\}^* \{1\}\{1\}\{1\} \{\{0\} \cup \{1\}\}^*$

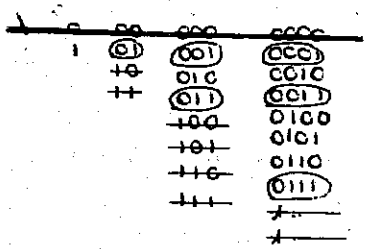
③ - $\Sigma = \{a,b\}$ $\quad L = \{x \in \Sigma^* : n^o a \text{ es impar}\}$

$L = \{b\}^* \{a\} \{b\}^* (\{a\}\{b\}^* \{a\} \{b\}^*)^*$

④ - Dado $L = \{0^m 1^n : m,n \geq 1\}$

Calcular $\bar{L}$ mediante la unión de 3 lenguajes: uno de ellos es el lenguaje que sólo tiene palabras 0, otro palabras que empiezan por 1 y el otro el resto de palabras.

$L = \{01, \begin{matrix}001\\011\end{matrix}, \begin{matrix}0001\\0011\\0111\end{matrix}, \begin{matrix}00001\\00011\\00111\\01111\end{matrix}, \ldots\}$ $\qquad \bar{L} = \Sigma^* - L$

$\Sigma^* =$ 

○ ⇒ palabras $\in L$

— $L_1 = \{0\}^*$ ⟶ Sólo palabras 0

— $L_2 = \{1\}\{\{0\} \cup \{1\}\}^* = \{1\}\Sigma^*$ ⟶ Las palabras que empiezan por 1.

○ $L_3 = \{0^m 1^n . \Sigma^* \; . \; m,n \geq 1\}$ ⟶ El resto de las palabras son aquellas que se forman de la siguiente manera: cualquier cadena de 0 seguida de una cadena de 1 y luego todas las posibles combinaciones de 0 y 1.

$\bar{L} = L_1 \cup L_2 \cup L_3$

⑤.- Hallar el lenguaje definido recursivamente de la siguiente forma:

$\Sigma = \{a, b\}$

1.- $\lambda \in L$

2.- $x \in L \Rightarrow axb \in L$ y $bxa \in L$

3.- $\left.\begin{array}{l} x \in L \\ y \in L \end{array}\right| \Rightarrow xy \in L$

$x=\lambda$ ó $x=a, y=b$ ó $x=b, y=a$

$L = \{\lambda, ab, ba, aabb, abab, baba, abba, baab, bbaa, \ldots\}$

$L = \{x : N_a(x) = N_b(x)\}$

⑥.- Dado el alfabeto $\Sigma = \{a, b, c, s, x\}$ con

$s ::= asb$ ①

$sb ::= bx$ ②

$abx ::= c$ ③

Calcular las derivaciones que se me ocurran, y que siguiendo un camino las derivaciones se paran o por otro camino llegamos a derivaciones del tipo $a^n c b^n$ $n \geq 0$

$s \xrightarrow{①} asb$

$asb \xrightarrow{②} abx$

$abx \xrightarrow{③} c$

$s \xrightarrow{3\text{ derivs.}} c$

$s \xrightarrow{①} asb \xrightarrow{①} aasbb \xrightarrow{①} aaasbbb \xrightarrow{①} \ldots$

$asb \xrightarrow{②} abx$

$aasbb \xrightarrow{②} aabxb$

$aaasbbb \xrightarrow{②} aaabxbb$

$abx \xrightarrow{③} c$

$aabxb \xrightarrow{③} acb$

$aaabxbb \xrightarrow{③} aacbb$

7. EXPRESIONES REGULARES.

Las expresiones regulares (ER) permiten representar concisamente lenguajes Regulares, de forma que resumen la descripción exhaustiva de dicho lenguaje. De esta forma, un lenguaje (conjunto de palabras que tienen una propiedad) infinito tendrá una representación de forma finita.

* **DEF:** - Dado un alfabeto Σ, construimos a partir de él un nuevo alfabeto que sea $B = \Sigma \cup \{(,), \lambda, \phi, +, *\}$

Símbolos:
- λ ≡ Palabra vacía.
- ϕ ≡ Conjunto vacío.
- $+$ ≡ Unión
- $*$ ≡ Cierre o Clausura.

Definimos una expresión regular α sobre el alfabeto Σ a las cadenas del alfabeto B.

$$\alpha = ER(\Sigma) \subseteq B^+$$

* **PROPIEDADES.-**

1) λ, ϕ, $a \in \Sigma$ Va son ER.

2) Si $\alpha \in ER(\Sigma)$, $\beta \subseteq B^*$ → entonces $\alpha + \beta$ y $\alpha\beta$ (concatenación) son $ER(\Sigma)$

↳ Dos formas distintas de decirlo. Significa: si α y β son ER.

3) Si α es $ER(\Sigma)$, α^* y (α) también lo son.

α^* se define así:

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

→ Concatenación de α consigo misma i veces.

$$\alpha^0 = \lambda$$

4) Sólo son $ER(\Sigma)$ las que se pueden obtener aplicando las reglas anteriores un número finito de veces.

El orden de prioridad de las operaciones cuando aparecen varias simultáneamente en una ER es:
1) * (cierre)
2) . (concatenación)
3) + (unión)

Este orden se puede modificar mediante paréntesis.

(4/10/05)

Toda ER definida sobre un alfabeto Σ representa un lenguaje regular que se define recursivamente de la siguiente forma:

1) $L(\phi) = \phi$ → Lenguaje vacío.

2) $L(\lambda) = \{\lambda\}$ → conjunto de la palabra vacía.

3) $L(a) = \{a\}$ $\forall a \in \Sigma$ → Lenguaje con una sola palabra. (palabra)

4) Si α y β son ER, $L(\alpha.\beta) = L(\alpha)\,L(\beta)$ (concatenación)

5) Si α y β son ER, $L(\alpha+\beta) = L(\alpha) \cup L(\beta)$

Si $\alpha = a+b$ → $L(a+b) = (L(a) \cup L(b)) = \{a\} \cup \{b\}$

6) Si α es una ER, $L(\alpha^*) = (L(\alpha))^*$

Vemos que al aplicar L a ER obtenemos los lenguajes.

Un lenguaje L decimos que es LENGUAJE REGULAR si $\exists \alpha \,/\, L(\alpha) = L$, es decir, si se puede representar mediante una ER.

Ej: $\Sigma = \{a, b, \dots, z\}$ $\quad \alpha = (a+b+\dots+z)^*$

$L(\alpha) = L(a+b+\dots+z)^* = (L(a+b+\dots+z))^* = (L(a) \cup L(b) \cup \dots \cup L(z))^*$

$(\{a\} \cup \{b\} \cup \dots \cup \{z\})^* = \Sigma^*$

Ej: $\Sigma = \{0, 1\}$ $\quad \alpha = 0^*10^*$

$L(\alpha) = L(0^*10^*) = L(0^*)\,L(1)\,L(0^*) = (L(0))^*\,L(1)\,(L(0))^* =$

$= \{0\}^*\{1\}\{0\}^* = 0^*10^*$

→ lenguaje de las palabras que tienen un solo 1.

EJ: $\Sigma = \{0,1\}$ $\quad \alpha = 01 + 000$

$L(\alpha) = L(01+000) = L(01) \cup L(000) = \{01\} \cup \{000\} = \{01, 000\}$

EJ. $\Sigma = \{a,b,c\}$ $\quad \alpha = a(a+b+c)^*$

$L(\alpha) = L(a(a+b+c)^*) = L(a)\,L(a+b+c)^* = \{a\} \cdot (L(a+b+c))^* =$

$= \{a\} \cdot (\{a\} \cup \{b\} \cup \{c\})^*$

↳ Lenguaje de las palabras que empiezan por a.

EJ: $\Sigma = \{a,b,c\}$ $\quad \alpha = a + bc + b^2a$

$L(\alpha) = L(a + bc + b^2a) = L(a) \cup L(bc) \cup L(b^2a) =$

$\{a\} \cup \{bc\} \cup \{b^2a\} = \{a, bc, bba\}$

EJ: $\Sigma = \{a,b\}$ $\quad \alpha = (a+b)^* a$

$L(\alpha) = L((a+b)^* a) = L(a+b)^* L(a) = (L(a+b))^* L(a) =$

$= (L(a) \cup L(b))^* L(a) = (\{a\} \cup \{b\})^* \{a\} = \{a,b\}^* a =$

$= (a \cup b)^* a$

↳ Lenguaje de las palabras que acaban en a.

EJ: $\Sigma = \{0,1\}$ $\quad \alpha = 0 + 01 + 101$

$L(\alpha) = L(0+01+101) = L(0) \cup L(01) \cup L(101) = \{0\} \cup L(0)L(1) \cup L(1)L(0)L$

$= \{0\} \cup \{0\}\{1\} \cup \{1\}\{0\}\{1\} = \{0, 01, 101\}$

EJ: $\Sigma = \{0,1\}$ $\quad \alpha = 01^*$

$L(\alpha) = L(01^*) = L(0)L(1^*) = \{0\}(L(1))^* = \{0\}\{\lambda, 1, 11, 111, \ldots\} =$

$= \{0, 01, 011, 0111, \ldots\}$

EJ: $\Sigma = \{0,1\}$ $\quad \alpha = (0+1)^*$

$L(\alpha) = L(0+1)^* = (L(0+1))^* = (L(0) \cup L(1))^* = (\{0\} \cup \{1\})^* = \{0,1\}^* =$

$= \{\lambda, 0, 1, 00, 01, 10, 11, \ldots\}$

- EQUIVALENCIAS DE EXPRESIONES REGULARES:

Dos expresiones regulares α y β decimos que, son equivalentes si representan el mismo lenguaje.

$$\alpha = \beta \iff L(\alpha) = L(\beta)$$

Las operaciones con ER tienen las siguientes propiedades.

• PROPIEDADES. -

1) Conmutativa respecto a la suma:

$$\alpha + \beta = \beta + \alpha$$

2) Asociativa respecto a la suma y la concatenación:

$$(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma)$$
$$\alpha(\beta\gamma) = (\alpha\beta)\gamma$$

3) $\exists$ elemento neutro respecto a la suma y la concatenación:

$$\alpha + \phi = \phi + \alpha = \alpha$$
$$\alpha\lambda = \lambda\alpha = \alpha$$

4) Distributiva respecto a la suma y la concatenación:

$$\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$$
$$(\alpha + \beta)\gamma = \alpha\gamma + \beta\gamma$$

Como consecuencia, tenemos:

SABERLAS

① $\lambda^* = \lambda$
② $\phi^* = \lambda$
③ $\phi \cdot \alpha = \alpha \cdot \phi = \phi$
④ $\alpha^* \cdot \alpha^* = \alpha^*$
⑤ $\alpha\alpha^* = \alpha^*\alpha$
⑥ $(\alpha^*)^* = \alpha^*$
⑦ $(\alpha + \lambda)^* = (\lambda + \alpha)^* = \alpha^*$
⑧ $\alpha^* = \lambda + \underbrace{\alpha\alpha^*}$ Clausura positiva.

⑨ $(\alpha^* + \beta^*)^* = (\alpha + \beta)^{**}$
⑩ $(\alpha^*\beta^*)^* = (\alpha + \beta)^*$
⑪ $(\alpha^*\beta)^*\alpha^* = (\alpha + \beta)^*$
⑫ $(\alpha\beta)^*\alpha = \alpha(\beta\alpha)^*$
⑬ $(\alpha^*\beta)^* = (\alpha + \beta)^*\beta + \lambda$
⑭ $(\beta\alpha^*)^* = \beta(\alpha + \beta)^* + \lambda$

IMPORTANTES

EJ: $\alpha_1 = (0^*1 0^*1)^* 0^*$ | $\alpha_2 = (0^*10^*10^*)$

→ Para ver si son o no equivalentes se puede comprobar si las palabras de uno pertenecen también al otro o no.

$0^i \in L(\alpha_1)$

$0^i \notin L(\alpha_2)$

$\longrightarrow L(\alpha_1) \neq L(\alpha_2) \rightarrow \alpha_1 \neq \alpha_2$ No son equivalentes.

EJ: $\alpha_1 = (0^*1)^* 0^* 0$ | $\alpha_2 = (0^*1^*)^* (0+1)^* 00^*$

Se empieza por la ER más larga y se intenta conseguir la otra aplicando las fórmulas.

$$\alpha_2 = (0^*1^*)^*(0+1)^*00^* \overset{⑩}{=} (0+1)^*(0+1)^*00^* \overset{④}{=} (0+1)^*00^* \overset{⑪}{=}$$

$$= (0^*1)^* 0^* 00^* \overset{⑤}{=} (0^*1)^* 0^* 0^* 0 \overset{④}{=} (0^*1)^* 0^* 0 = \alpha_1$$

$L(\alpha_1) = L(\alpha_2) \longleftrightarrow \alpha_1 = \alpha_2$ Son equivalentes.

EJERCICIOS.-

① Simplificar la expresión:

$$\alpha = ab + ab^*bb + aab + aa^*aab \overset{\text{distributiva}}{=\!=} a(b + b^*bb + ab + a^*aab) =$$

$$a\big(\underbrace{(\lambda + bb^*)}_{b^*} b + \underbrace{(\lambda + a^*a)}_{a^*} ab\big) = a(b^*b + a^*ab) = a(b^* + a^*a)b$$

↑ ¡¡¡ La pongo detrás porque está a la derecha en los términos !!!

② Dada la expresión regular $\alpha_1 = b(ab + b)^*a + \lambda$ y $\alpha_2 = (bb^*a)^*$ comprobar si ambas son expresiones equivalentes.

Se pueden utilizar las siguientes propiedades: $(\alpha\beta)^*\alpha = \alpha(\beta\alpha)^*$

$(\alpha + \beta)^* = (\alpha^*\beta)^*\alpha^*$

$$\alpha_1 = b\underbrace{(ab + b)^*}_{\text{distributiva}}a + \lambda = b\underbrace{((a+\lambda)b)^*}_{\text{distributiva}}a + \lambda = \underbrace{(b(a+\lambda))^*}_{\text{distributiva}}ba + \lambda = (ba + b)^*ba + \lambda =$$

$$= (b + ba)^*ba + \lambda = (b^*ba)^*b^*ba + \lambda = (b^*ba)^* = (bb^*a)^* = \alpha_2$$

③ Calcular el lenguaje de las palabras $w \in \{0,1\}^*$ tales que en w aparece el "1" dos o tres veces, la primera y la segunda de las cuales no son consecutivas.

Hay que aplicar $\emptyset^* = \lambda$

DEPARTAMENTO DE INTELIGENCIA ARTIFICIAL

FACULTAD DE INFORMÁTICA
UNIVERSIDAD POLITÉCNICA DE MADRID

INFORMÁTICA TEÓRICA

Curso 2004-2005

Prácticas Tema 1

LENGUAJES FORMALES

Práctica 1.1: Operaciones con palabras

Dado Σ^*, lenguaje universal sobre el alfabeto Σ, y siendo $x \in \Sigma^*$, se define por recursividad sobre la longitud de la palabra x, la *inversa* de ésta, x^{-1}, como:

1^o - Si $|x| = 0 \Rightarrow x^{-1} = x = \lambda$
2^o - Si $|x| > 0 \Rightarrow x = u \,.\, e$ para algún $e \in \Sigma, u \in \Sigma^* \Rightarrow \underline{x^{-1} = e \,.\, u^{-1}}$

Probar que: $\forall x, y \in \Sigma^* \Rightarrow \underline{(x.y)^{-1} = y^{-1} \,.\, x^{-1}}$

Práctica 1.2: Operaciones con lenguajes

1.- Dado el lenguaje $L = \{ x \in \{a,b\}^* \mid N_a(x) \neq N_b(x) \}$, calcular el cierre de Kleene L^*.
($N_a(x)$ es el nº de a's de la palabra x).

2.- Dado el lenguaje $L = \{ 0^m 1^n \mid m,n \geq 1 \}$ expresar $\overline{L}$ como unión de tres lenguajes. (Indicación: agrupar las palabras de $\overline{L}$ como: a) palabras que sólo tienen 0's, b) palabras que empiezan por un 1, y c) las restantes palabras).

Práctica 1.3: Definición recursiva de lenguajes

a) Sea el lenguaje L sobre el alfabeto {a,b} definido recursivamente de la forma siguiente:

i) $\lambda \in L$

ii) $x \in L \Rightarrow axb \in L$ y $bxa \in L$

iii) $x,y \in L \Rightarrow xy \in L$

iv) son palabras de L todas las que se obtienen aplicando las reglas i), ii) y iii) un número finito de veces.

Describir razonadamente el lenguaje L (i.e, ¿cómo son las palabras de L?)

b) Dar una definición recursiva del lenguaje

$L' = \{ x \in \{a,b\}^* \mid x \neq \lambda \quad N_a(x) = 2.N_b(x) \}$,

Práctica 1.4: Expresiones regulares

1.- Construir directamente una expresión regular que represente los siguientes lenguajes :

$L_1 = \{ x \in \{0,1\}^* \mid$ en x aparece el 1 dos o tres veces , la primera y la segunda de las cuales no son consecutivas $\}$

$L_2 = \{ x \in \{a,b,c\}^* \mid$ ac no es parte de x $\}$.
(Indicación: ¿cómo, dónde, deben ir las c's?

$L_3 = \{ x \in \{a,b,c\}^* \mid$ x tiene un nº par de ocurrencias de ac $\}$.
(Indicación: utilizar el lenguaje L_2)

2.- Estudiar si son o no equivalentes las siguientes expresiones regulares :

a) $E_1 = \alpha\alpha^*(\alpha^*\beta)^* + \alpha\alpha^*(\alpha^* + \beta^*)^* + \lambda$

$E_2 = \alpha^*(\alpha^*\beta^*)^* + \alpha^*\alpha(\alpha + \beta)^*\beta + \lambda$

b) $E_3 = \beta(\alpha\beta + \beta)^*\alpha + \lambda$

$E_4 = (\beta\beta^*\alpha)^*$

Indicación: Se necesita hacer uso, entre otras, de las propiedades siguientes:

$(\alpha + \beta)^* = (\alpha^*\beta)^*\alpha^*$

$(\alpha\beta)^*\alpha = \alpha(\beta\alpha)^*$

TEMA 2 GRAMÁTICAS FORMALES.

1. DEFINICIONES.

- **GRAMÁTICA FORMAL:** Las gramáticas permiten describir lenguajes. También se dice que una gramática genera un lenguaje.

Una gramática formal G se define como una cuádrupla

$$G = (\Sigma_T, \Sigma_N, S, P) \qquad \Sigma_T \cap \Sigma_N = \emptyset \; ; \; \Sigma_T \cup \Sigma_N =$$

siendo:
- Σ_T ≡ Alfabeto de los "símbolos terminales."
- Σ_N ≡ " " " "no terminales."
- S ≡ Un símbolo no terminal ($S \in \Sigma_N$) llamado "axioma de la gramática"
- P ≡ Un conjunto de producciones.

- **RELACIÓN DE THUE:** Sean v y w dos palabras del mismo alfabeto Σ: $v, w \in \Sigma^*$. Decimos que entre v y w existe una relación de Thue si $v \rightarrow^* w$ ó $v = w$.
Se representa por: $v \rightarrow^* w$

- **FORMA SENTENCIAL:** Decimos que una palabra es forma sentencial si existe una relación de Thue entre el axioma y dicha palabra: $S \rightarrow^* x$, es decir, si existe una derivación desde el axioma hasta la palabra.
Una forma sentencial es una sentencia si todos sus símbolos pertenecen a Σ_T: $x \in \Sigma_T^*$

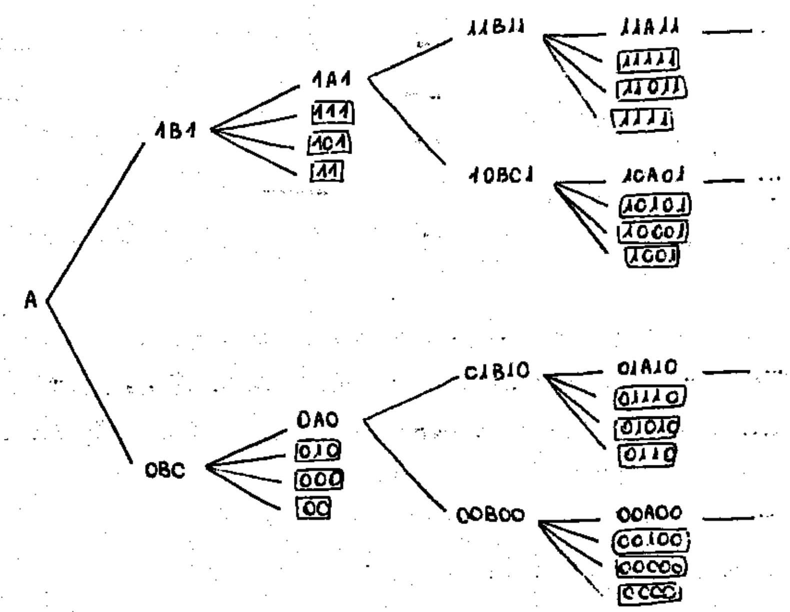
- LENGUAJE GENERADO POR UNA GRAMÁTICA:

$L(G)$ es el conjunto de todas las sentencias de la gramática G; es decir, todas las palabras que se pueden obtener a partir del axioma de la gramática por la aplicación de derivaciones.

$$L(G) = \{ x \in \Sigma_T^* \ : \ S \rightarrow_* x \}$$

UN LENGUAJE PUEDE SER GENERADO POR VARIAS GRAMÁTICAS, PERO UNA GRAMÁTICA GENERA SÓLO UN LENGUAJE.

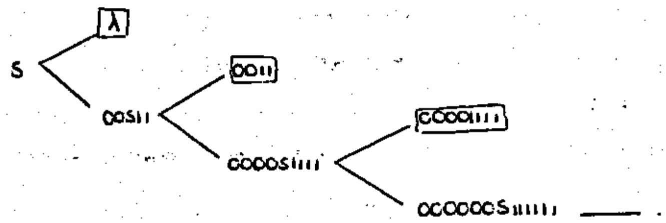
EJ: $G = (\{0,1\}, \{A,B\}, A, P)$ donde $P = \begin{cases} A ::= 1B1 \mid 0B0 \\ B ::= A \mid 1 \mid 0 \mid \lambda \end{cases}$



Lenguaje de las palabras binarias simétricas (palíndromos)

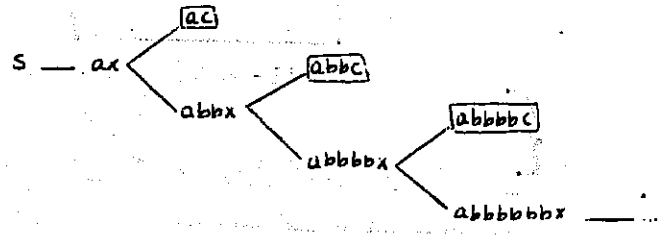
$L(G) = \{ x \ / \ x = x^{-1} \}$

EJ: $G = (\{0,1\}, \{S\}, S, P)$ donde $P = \{ S ::= \lambda \mid 0S11 \}$ ($\{0,1\} = \Sigma_T$, $\{S\} = \Sigma_N$)



$L = \{ 0^{2n} 1^{2n} : n \geq 0 \}$ ...... $L(G) = \{ x \in \{0,1\}^* : x = 0^{2n} 1^{2n}, n \geq 0 \}$

Ej. $G = \{\{a,b,c\}, \{S,X\}, S, P\}$ donde $P = \begin{cases} S ::= aX \\ X ::= bbX \mid c \end{cases}$



$L = \{ab^{2n}c, n \geq 0\}$

Ej. "El hombre comió pan".

< Artículo > ::= El

< Nombre > ::= hombre | pan

< Verbo > ::= comió

< Oración > ⟶ < Sujeto >< Predicado > ⟶ < Frase nominal >< Predicado > ⟶ < Grupo nominal >< Predic ⟶ < Artículo >< Nombre >< Predicado > ⟶ El < Nombre >< Predicado > ⟶ El hombre < Verbo >< Comple mentos > ⟶ El hombre comió < Complementos >

(NO) → Tenemos las siguientes reglas de producción:
(Las palabras encerrado entre <,> son símbolos de un alfabeto)

< oración >	::=	< sujeto >< predicado >
< sujeto >	::=	< frase nominal >
< frase nominal >	::=	< grupo nominal > | < grupo nominal >< calificativo >
< grupo nominal >	::=	< nombre >
< grupo nominal >	::=	< artículo >< nombre >
< predicado >	::=	< verbo >< complementos >
< complementos >	::=	< directo >< indirecto >< circuns > | < indirecto >< circuns > | < directo >< circuns > | < circuns >
< calificativo >	::=	< conjunción >< oración > | < adjetivo >
< circuns >	::=	λ | < circun >< circuns >
< directo >	::=	< frase nominal >
< indirecto >	::=	"a" < frase nominal >
< circun >	::=	< preposición >< frase nominal >

- **RECURSIVIDAD**: Una producción es recursiva si aparece en el antecedente y en el consecuente de la producción.

$$A = xAy \qquad x, y \in \Sigma^*$$

Recursiva por la izquierda → $A = Ay$, $x = \lambda$
" " derecha → $A = xA$, $y = \lambda$

Si la gramática tiene alguna producción recursiva será una gramática recursiva.

* Si un lenguaje es ∞, la gramática que genera dicho lenguaje es recursiva.

2. TIPOS DE GRAMÁTICAS.

Chomsky definió 4 tipos de gramáticas formales, que se diferencian en los tipos de producciones de la gramática. Esta clasificación permitirá introducir al mismo tiempo una clasificación en los lenguajes que las gramáticas generan y una clasificación en los autómatas que reconocen los lenguajes.

(Más específica) $G_3 \subset G_2 \subset G_1 \subset G_0$ (Más general)

- **TIPO 0** (sin restricciones):

Es aquella cuyas producciones son de la forma:

→ A la izq. tiene que haber al menos un símbolo

$y = xAy$

$$\left[\begin{array}{ll} u ::= v & u \in \Sigma^+ ;\ v \in \Sigma^* \\ u ::= xAy & A \in \Sigma_N ,\ x, y \in \Sigma^* \end{array}\right]$$ sin ninguna restricción adicional.

Los lenguajes representados por estas gramáticas se llaman "lenguajes sin restricciones".

Todo lenguaje representado por una gramática de Tipo 0 puede describirse también por las **GRAMÁTICAS DE ESTRUCTURA DE FRASES**, que son aquellas cuyas producciones son de la forma:

$$xAy ::= xvy \qquad x, v, y \in \Sigma^* \;,\; A \in \Sigma_N$$

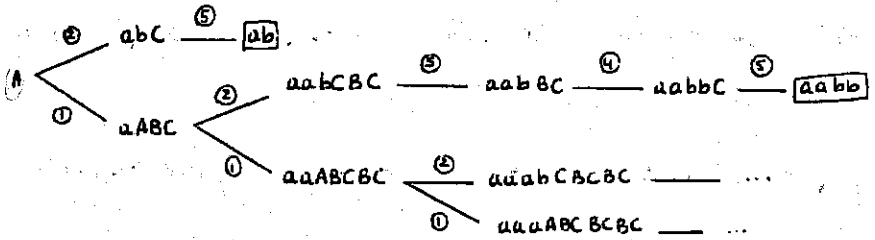
Puesto que puede ser que $v = \lambda$, algunas de las reglas de estas gramáticas pueden tener la parte derecha más corta que la izquierda. Si esto ocurre, se dice que es una **REGLA COMPRESORA**.

- **GRAMÁTICA COMPRESORA**: Gramática que tiene al menos una regla compresora. Reduce la longitud de la palabra a medida que vamos derivando.

Ej: $G = (\{a,b\}, \{A,B,C\}, A, P)$ donde $P =$ (axioma)

$A ::= aABC \mid abC$	① ②
$CB ::= BC$	③
$bB ::= bb$	④
$bC ::= b$	⑤ → Regla compresora. Es una gramática compresora.

G es de Tipo 0.



$$L(G) = \{ a^n b^n \;/\; n = 1, 2, \ldots \}$$

Esta es una gramática de Tipo 0, pero no de estructura de frases, pues la regla $CB ::= BC$ no cumple las condiciones. Sin embargo, esta regla se puede sustituir por las siguientes:

$CB ::= XB$
$XB ::= XY$
$XY ::= BY$
$BY ::= BC$

De esta manera se obtienen las mismas derivaciones. Aunque se dan más pasos, si cumplen las condiciones para ser una gramática con estructura de frases.
Las dos gramáticas son equivalentes.
↑ Esta gramática tiene 3 reglas de producción más y dos símbolos no terminales más X, Y.

CUALQUIER GRAMÁTICA TIPO 0 PODEMOS CONVERTIRLA EN UNA GRAMÁTICA DE ESTRUCTURA DE FRASES.

- **Tipo 1** (Dependientes de contexto o Gram. de contexto sensitivo):

Las reglas de producción de estas gramáticas tienen la forma:

$$[\, xAy ::= xvy \,] \qquad x, y \in \Sigma^* \quad v \in \Sigma^+ \quad A \in \Sigma_N$$

$\downarrow v \neq \lambda$

No puede tener reglas compresoras. Únicamente se permite: $S ::= \lambda$

→ En consecuencia, en una gramática Tipo 1, la palabra vacía λ pertenece a $L(G) \iff S ::= \lambda \in P$ (última)

$$\lambda \in L(G) \rightarrow S ::= \lambda \in P$$

Los lenguajes representados por las gramáticas de Tipo 1 se llaman "lenguajes dependientes de contexto", ya que la forma de las reglas ($xAy ::= xvy$) puede interpretarse diciendo que A sólo se puede transformar en v si A está precedido por x y seguido por y. Es decir, hay que tener en cuenta el "contexto" de A para derivar.

Es evidente que toda gramática de Tipo 1 es también una gramática de Tipo 0 y, por tanto, todo lenguaje dependiente de contexto es también un lenguaje sin restricciones.

Ej: $G = (\{0,1\}, \{A, B\}, A, P)$ donde $P =$

$A ::= 1B1 \rightarrow xAy \quad x=y=\lambda \quad A \rightarrow 1B1$

$A ::= 11 \rightarrow x=y=\lambda \quad A \rightarrow 11$

$1B1 ::= 101$

$1B1 ::= 111$

G es de tipo 1.

$L(G) = \{11, 101, 111\}$

- **Tipo 2** (Independientes de contexto o de contexto libre):

Las reglas de producción de estas gramáticas tienen la forma:

$$[\, A ::= v \,] \qquad v \in \Sigma^+ \quad A \in \Sigma_N$$

Se permite también $S ::= \lambda$ (última)

↳ La parte izquierda sólo puede tener un símbolo no terminal.

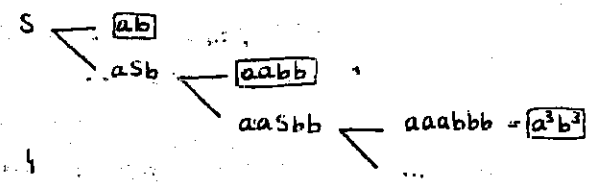
$\lambda \in L(G) \iff S ::= \lambda \in P$

Los lenguajes descritos por gramáticas de Tipo 2 se llaman "lenguajes independientes de contexto" porque a la hora de transformar una palabra en otra (A en v) no importa el contexto donde se encuentre A.

Toda gramática Tipo 2 cumple los requisitos de las gramática Tipo 1. Por tanto, todo lenguaje independiente de contexto pertenecerá también a la clase de lenguajes dependientes de contexto.

Ej: $G_1 = (\{a,b\}, \{S\}, S, P)$ donde $P = (S ::= aSb \mid ab)$

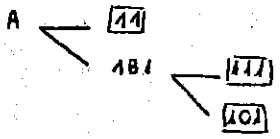
G_1 es de tipo 2.



$L(G_1) = \{a^n b^n : n = 1, 2, \ldots\}$

Ej: $G_2 = (\{0,1\}, \{A,B\}, A, P)$ donde $P \equiv \begin{cases} A ::= 1B1 \\ A ::= 11 \\ B ::= 1 \\ B ::= 0 \end{cases}$

G_2 es de tipo 2.



$L(G_2) = \{11, 101, 111\}$

- **Tipo 3 (regulares o lineales):**

Estas gramáticas pueden ser de dos tipos:

* Gramáticas lineales por la izquierda → Las reglas pueden tener estas formas:

$$\begin{bmatrix} A ::= a \\ A ::= Va \\ S ::= \lambda \end{bmatrix}$$

GLI

$A, V \in \Sigma_N \quad a \in \Sigma_T \quad S$ el axioma.

* Gramáticas lineales por la derecha → Las reglas pueden tener estas formas:

$$\begin{bmatrix} A ::= a \\ A ::= aV \\ S ::= \lambda \end{bmatrix}$$

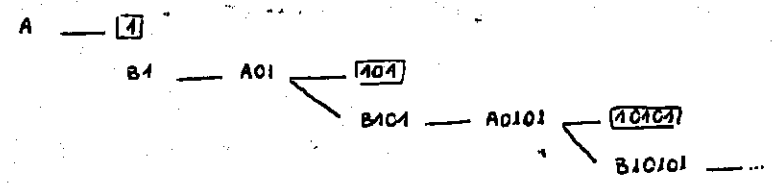
GLD

$A, V \in \Sigma_N \quad a \in \Sigma_T \quad S$ el axioma.

Los lenguajes que pueden representarse mediante gramáticas Tipo 3 se llaman "lenguajes regulares". Todo lenguaje regular pertenecerá también a la clase de los lenguajes independientes de contexto (Tipo 2).

Ej: $G_1 = (\{0,1\}, \{A, B\}, A, P)$ donde $P \equiv \begin{cases} A ::= B1 \mid 1 \\ B ::= A0 \end{cases}$

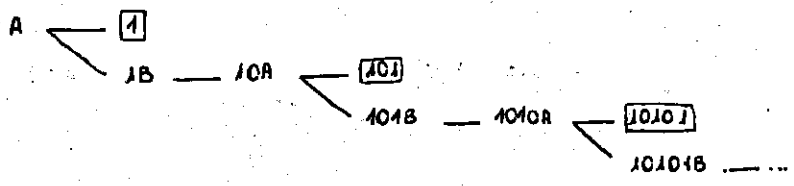
Se trata de una gramática lineal por la izquierda.



$L(G_1) = \{1, 101, 10101, \ldots\} = \{1(01)^n, n = 0,1 \ldots\}$

Ej: $G_2 = (\{0,1\}, \{A, B\}, A, P)$ donde $P \equiv \begin{cases} A ::= 1B \mid 1 \\ B ::= 0A \end{cases}$

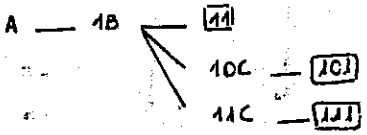
Se trata de una gramática lineal por la derecha.



El lenguaje es el mismo que el anterior

$L(G_2) = \{1, 101, 10101, \ldots\} = \{1(01)^n, n = 0, 1 \ldots\}$

Ej: $G_3 = (\{0,1\}, \{A, B\}, A, P)$ donde $P \equiv \begin{cases} A ::= 1B \\ B ::= 1 \mid 0C \mid 1C \\ C ::= 1 \end{cases}$ Gram. lineal por la derecha



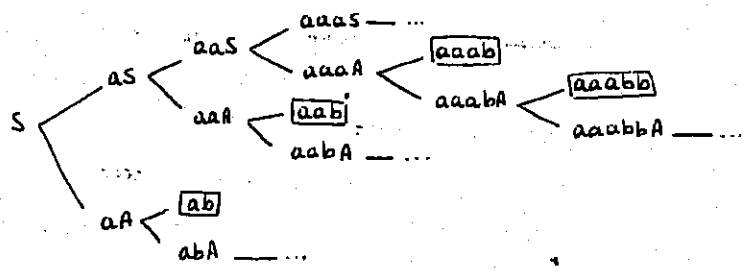
$L(G_3) = \{11, 101, 111\}$

3. DISEÑO DE GRAMÁTICAS.

① Gramática que genere el lenguaje $L = \{a^m b^n : m>0, n>0\}$

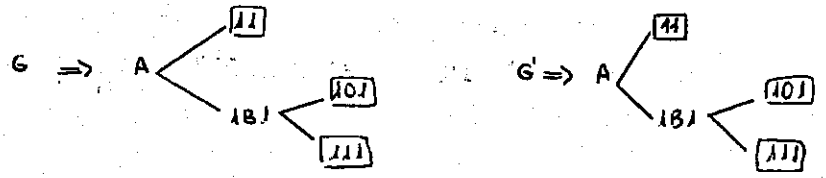
$G = \{\Sigma_T, \Sigma_N, S, P\}$ $\quad G = (\{a,b\}, (S,A), S, P)$ donde $P = \begin{cases} S ::= aS \mid aA \\ A ::= bA \mid b \end{cases}$

$L = aa^* bb^*$



② Dada la gramática $G = (\{0,1\}, \{A,B\}, A, \{A ::= 1B1 \mid 11,\ 1B1 ::= 101 \mid 111\})$.
Tenemos también una gramática de Tipo 2 equivalente:

$G' = (\{0,1\}, \{A,B\}, A, \{A ::= 1B1 \mid 11,\ B ::= 0 \mid \lambda\})$



EQUIVALENCIA DE GRAMÁTICAS: Se dice que dos gramáticas son equivalentes cuando generan el mismo lenguaje.

* Dada una gramática de Tipo 0, existe una gramática equivalente expresada en estructura de frases:

$A ::= aABC$		$CB ::= XB$	Sustituyendo $CB ::= BC$ por estas 4 producciones conseguimos que la gramática esté en estructura de frases y sea equivalente.
$A ::= abC$		$XB ::= XY$	
$CB ::= BC$	$\longrightarrow$	$XY ::= BY$	
$bB ::= bb$		$BY ::= BC$	
$bC ::= b$			

(17/10/05)

-ÁRBOLES DE DERIVACIÓN. DERIVACIONES

Ej: Sea la gramática.

$G = (\{a, b, c, +, \times, (,)\}, \{S\}, S, P)$ donde $P \equiv \{S ::= S+S \mid (S) \mid a \mid b \mid c \mid S \times S$

Queremos hacer palabras del tipo: $x = a+b\times c$

- Definimos una derivación izquierda de una gramática (D.I.) a aquella derivación cuyo primer símbolo no terminal que se sustituye es el que se encuentra más a la izquierda de la forma sentencial.
- Existe también la derivación derecha de una gramática (D.D.), que consiste en sustituir primero el símbolo no terminal que se encuentre más a la derecha.

Quiero conseguir la sentencia $x = a+b\times c$ mediante una D.I.

D.I.- $S \rightarrow \underline{S}+S \rightarrow a+\underline{S} \rightarrow a+\underline{S}\times S \rightarrow a+b\times\underline{S} \rightarrow a+b\times c$

D.D.- $S \rightarrow \underline{S}\times S \rightarrow \underline{S}+S\times S \rightarrow a+\underline{S}\times S \rightarrow a+b\times\underline{S} \rightarrow a+b\times c$

- Los árboles de derivación son una forma de representar derivaciones. Sólo se pueden definir árboles de derivación para gramáticas de Tipo 1, 2 o 3. A cualquier derivación le corresponde un árbol de derivación construido de la siguiente manera:

  - La raíz del árbol es el axioma de la gramática.

  - Los nodos hojas del árbol son símbolos terminales de la gramática. Los nodos intermedios son símbolos no terminales.

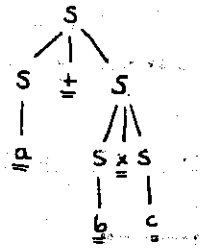
  - Una derivación directa se representa por un conjunto de ramas que salen de un nodo dado. Al aplicar una producción, un símbolo de la parte izquierda ($\_ ::= x$) queda sustituido por un palabra (x) de la parte derecha ($\_ ::= x$). Por cada uno de los símbolos de x se dibuja una rama que parte del nodo dado y termina en otro con dicho símbolo.

    Si el símbolo A está a la izq. de B en la palabra x, la rama que termina en A se dibuja a la izq. de la de B

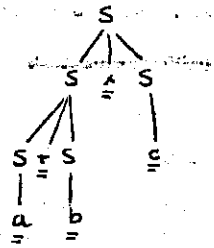
EJ: ① $S \rightarrow \underline{S} + S \rightarrow a + \underline{S} \rightarrow a + \underline{S} \times S \rightarrow a + b \times \underline{S} \rightarrow a + b \times c$

② $S \rightarrow \underline{S} \times S \rightarrow \underline{S} + S \times S \rightarrow a + \underline{S} \times S \rightarrow a + b \times \underline{S} \rightarrow a + b \times c$

ÁRBOL ①



ÁRBOL ②



* Todo árbol de derivación representa una única derivación de la sentencia. Toda derivación de la sentencia se puede representar con un único árbol. Por tanto, si una sentencia tiene 2 o más derivaciones, entonces tendrá 2 o más árboles que la representen (y viceversa).

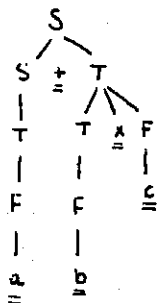
- **AMBIGÜEDAD**: Cuando una sentencia se puede obtener mediante dos árboles de derivación distintos (tiene dos derivaciones distintas) decimos que esa sentencia es ambigua.
  ↳ ① ≠ ②

  Si una gramática tiene al menos una sentencia ambigua decimos que esa gramática es ambigua. Para ver si una gramática no es ambigua habría que comprobar todas las sentencias de la gramática y ver que ninguna lo es
  ↳ palabras del lenguaje quiero

EJ: $G = (\{a, b, c, +, \times, (,)\}, \{S, T, F\}, S, \{ S ::= S+T \mid T$
$T ::= T \times F \mid F$
$F ::= (S) \mid a \mid b \mid c \})$

Sentencia: $x = a + b \times c$

$S \rightarrow \underline{S} + T \rightarrow \underline{T} + T \rightarrow \underline{F} + T \rightarrow a + \underline{T} \rightarrow a + \underline{T} \times F \rightarrow$
$\rightarrow a + \underline{F} \times F \rightarrow a + b \times \underline{F} \rightarrow a + b \times c$



Esta sentencia no es ambigua para esta gramática.
(No podemos garantizar que no sea ambigua la gramática)

4. DEPURACIÓN DE GRAMÁTICAS.

independiente de contexto

El objetivo de la depuración de una gramática es obtener una gramática equivalente bien formada para agilizar el proceso sintáctico y pueda ser tratada eficientemente por un autómata.

Para ello, lo que hacemos es eliminar una serie de reglas o producciones:

a) **AXIOMA INDUCIDO**: Son aquellas producciones en las que el axioma se encuentra en la parte derecha.

Si $S ::= \lambda \in P$ (conjunto de producciones) debemos eliminarla de esta manera:

$S' ::= S$ → S' nuevo axioma (nuevo símbolo no terminal).

b) **REGLAS INNECESARIAS**: Son aquellas de la forma: $A ::= A$

Se deben eliminar de las gramáticas porque no tienen ninguna utilidad.

c) **REGLAS DE RENOMINACIÓN**: Son aquellas de la forma: $A ::= B$ $(A, B \in \Sigma_N)$

Estas reglas se eliminan y se añaden las siguientes:

Para cada símbolo A / $A ::= B$ y, para cada regla de la forma $B ::= x$ (x símbolo no terminal) hay que añadir una regla de la forma $A ::= x$

d) **REGLAS REDUCTORAS**: Aquellas de la forma $A ::= \lambda$ (Si A no es el axioma).

Se elimina $A ::= \lambda$ y por cada regla de la forma $B ::= xAy$ añado $B ::= xy$.

$B ::= x_1 A_1 x_2 A_2 \dots A_{k-1} x_k$

$B ::= x_1 x_2 A_2 x_3 \dots A_{k-1} x_k$

$B ::= x_1 A_1 x_2 x_3 \dots A_{k-1} x_k$

;

$B ::= x_1 x_2 x_3 A_3 x_4 \dots A_{k-1} x_k$

→ Se añaden las producciones resultantes de eliminar 1, 2, ... todas las veces y formas A.

e) **SÍMBOLOS INACCESIBLES**: Son aquellos símbolos no terminales que no son accesibles desde el axioma. Deben eliminarse.

g) **REGLAS NO GENERATIVAS**: Son aquellas que no participan en la generación de ninguna palabra del lenguaje. Deben eliminarse.

Una **GRAMÁTICA está BIEN FORMADA** si no contiene reglas innecesarias, ni de redenominación, ni reductoras, ni no generativas, ni símbolos inaccesibles. En el caso de que la producción $S ::= \lambda$ pertenezca a la gramática, tampoco debe tener el axioma inducido.

No hay ningún orden concreto, pero lo primero que se quita son:

- símbolos inaccesibles.
- reglas innecesarias.
- " no generativas.

EJEMPLO: Depurar

$S ::= ABA \mid A\cancel{B}E$

$A ::= a \mid \lambda \mid \cancel{E}$ / Reglas no generativas

$B ::= A \mid b$

$\cancel{E ::= Ea}$ → Nunca saldrá una palabra, siempre habrá algún símbolo no terminal. Por tanto, también elimino todo lo que haga referencia a E.

$\boxed{C ::= a}$ Inaccesible

⇓

$S ::= ABA$

$A ::= a \mid \cancel{\lambda}$ $A ::= \lambda$ Regla reductora. Elimino $A ::= \lambda$

$B ::= A \mid b$ Añado $S ::= BA \mid AB \mid B$ ⊛, $B ::= \lambda$

⇓

⊛ Todas las posibles resultados si sustituyo una o las dos A en ABA.

$S ::= ABA \mid AB \mid BA \mid B$

$A ::= a$ $B ::= \lambda$ Regla reductora. Elimino $B ::= \lambda$

$B ::= A \mid b \mid \cancel{\lambda}$ Añado $S ::= AA \mid A \mid \lambda$

⇓

$S ::= ABA \mid AB \mid BA \mid B \mid AA \mid A \mid \lambda$ Elimino $B ::= A$

$A ::= a$ Añado $B ::= a$

$B ::= \underline{A} \mid b$

Redenominación.

⇓

$$\begin{cases} S ::= ABA \mid AB \mid BA \mid \underline{B} \mid AA \mid \underline{A} \mid \lambda \\ A ::= a \\ B ::= a \mid b \end{cases}$$

Redenominación

Elimino $S ::= A \mid B$

Añado $S ::= a \mid b$

$$\Downarrow$$

$$\begin{cases} S ::= ABA \mid AB \mid BA \mid AA \mid b \mid a \mid \lambda \\ A ::= a \\ B ::= a \mid b \end{cases}$$

EJERCICIOS

(11) Gramática que genere $N_a(x) = N_b(x)$. (n^o de a = n^o de b).

$\Sigma = \{a, b\}$

$S ::= abS \mid baS \mid aSb \mid bSa \mid Sab \mid Sba \mid \lambda$ → Pongo el S en todos los sitios posibles.

(12) $L = \{a^m b^n c^k : \underline{m = n+k} \; ; \; n, k > 0\} \Rightarrow \{a^n a^k b^n c^k\} = \{a^k a^n b^n c^k\}$

$$\begin{cases} S ::= aSc \mid A \\ A ::= aAb \mid \underline{\underline{\lambda}} \end{cases}$$

La depuramos:

Elimino $A ::= \lambda$

Añado $\begin{cases} S ::= \lambda \\ A ::= ab \end{cases}$

Reductora

$$\Downarrow$$

$$\begin{cases} S ::= aSc \mid \underline{A} \mid \lambda \\ A ::= aAb \mid ab \end{cases}$$

Redenominación

Elimino $S ::= A$

Añado $S ::= aAb \mid ab$

$$\begin{cases} S ::= aSc \mid aAb \mid ab \mid \lambda \\ A ::= aAb \mid ab \end{cases} \equiv P$$

$G = (\{a, b, c\}, \{S, A\}, S, P)$

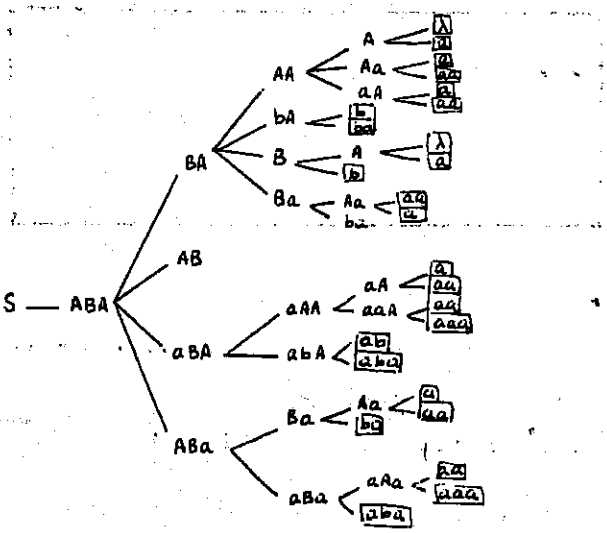
(13) Calcular los lenguajes que generan cada gramática.

Mirar si son equivalentes.

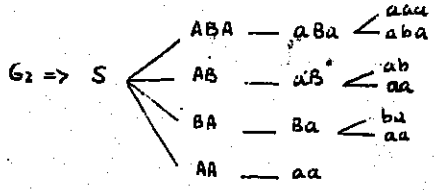
Depurar G_1 para llegar a G_2.

$G_1 = (\{a,b\}, \{A,B,S\}, S, \{S ::= ABA,\ A ::= \lambda \mid a,\ B ::= A \mid b\})$

$G_2 = (\{a,b\}, \{A,B,S\}, S, \{S ::= ABA \mid AB \mid BA \mid AA \mid a \mid b \mid \lambda\})$



$L_1 = \{\lambda, a, b, aa, ab, ba, aaa, aba\}$



$L_2 = L_1$ Luego, G_1 y G_2 son equivalentes.

(14) Dada la gramática cuyas producciones son P calcular el lenguaje que genera. Comprobar que ese lenguaje es $L = \{a^n b^m : n > m > 0\}$ y calcular una gramática G' equivalente.

$$P = \begin{cases} S ::= aAb \\ A ::= aAb \mid aB \mid a \\ B ::= aB \mid a \end{cases}$$

(15) $S ::= S+E \mid E$ $\quad$ E $\quad$ S+E $\quad$ E+E $\quad$ S+E+E

$E ::= ET \mid T$ $\quad$ T $\quad$ TT $\quad$ ETT $\quad$ TTT

$T ::= 0 \mid 1 \mid \lambda \mid \emptyset \mid T^* \mid (S)$

$S \to E \to ET \to ETT \to ETTT \to ETTTTTT \to TTTTTTT \to \dots$

E T

$a ::= 0^* 1 0^* 0 1 0^* \ (10^* \cup \emptyset^*)$

- FORMA NORMAL DE CHOMSKY (FNC)

Las gramáticas independientes de contexto (Tipo 2) se pueden transformar en gramáticas equivalentes expresadas en la Forma Normal de Chomsky.

Una gramática se dice que está en FNC cuando sus producciones son de la forma:

$$\begin{bmatrix} A ::= BC \\ A ::= a \\ S ::= \lambda \end{bmatrix} \qquad a \in \Sigma_T \qquad A, B, C \in \Sigma_N$$

¿Cómo pasar de una gramática de Tipo 2 a una FNC?

0) Depurar la gramática para conseguir una bien formada.

Dada la regla $A ::= x$ con $|x| = 1$, entonces $x \in \Sigma_T$ porque no hay reglas de redenominación. Por tanto, esta regla ya está en FNC.

1) Para las producciones $A ::= x$, $|x| \geq 2$, se sustituyen todos los símbolos terminales a_i que aparezcan por símbolos no terminales nuevos X_i y se añaden las producciones $X_i ::= a_i$ $a_i \in \Sigma_T$.

Ej: $S ::= aBb$ $\left\{ \begin{array}{l} S ::= X_1 B X_2 \\ X_1 ::= a \\ X_2 ::= b \end{array} \right.$ $X_1 X_2 \in N_T$

2) Si el consecuente de la producción tiene 2 símbolos, ya está en FNC. Si tiene la forma $A ::= B_1 B_2 \dots B_k$ $k > 2$, entonces la descomponemos de la siguiente forma:

$$A ::= B_1 \overbrace{B_2 \dots B_k}^{Y_1}$$

$$A ::= B_1 Y_1 \qquad Y_1 = B_2 \overbrace{B_3 \dots B_k}^{Y_2}$$

$$Y_1 ::= B_2 Y_2$$

$$\vdots$$

$$Y_{k-2} ::= B_{k-1} B_k$$

$Y_1, Y_2, \dots, Y_{k-2}$ son nuevos símbolos no terminales.

EJ : Sea el lenguaje $L = \{a^m b^n c^p d^n e^m : m \geq 2, n \geq 1, p \geq 2\}$

1) Obtener una gramática independiente de contexto que genere L con 6 producciones.

2) Obtener una gramática equivalente en FNC.

1) $G = (\{a,b,c,d,e\}, \{S,A,B\}, S, P)$

$P =$
- $S ::= aSe \mid aaAee$
- $A ::= bAd \mid bBd$
- $B ::= cB \mid cc$

2)
- $S ::= x_1 \overbrace{S x_2}^{Y_1} \mid x_1 x_1 A x_2 x_2$ (con Y_2, Y_3, Y_4)
- $A ::= x_3 \overbrace{A x_4}^{Y_5} \mid x_3 \overbrace{B x_4}^{Y_6}$
- $B ::= x_5 B \mid x_5 x_5$

$x_1 ::= a$	$S ::= x_1 Y_1$	$A ::= x_3 Y_5$
$x_2 ::= e$	$S ::= x_1 Y_2$	$Y_5 ::= A x_4$
$x_3 ::= b$	$Y_1 ::= S x_2$	$A ::= x_3 Y_6$
$x_4 ::= d$	$Y_2 ::= x_1 Y_3$	$Y_6 ::= B x_4$
$x_5 ::= c$	$Y_3 ::= A Y_4$	
	$Y_4 ::= x_2 x_2$	

(= P')

$G' = (\{a,b,c,d,e\}, \{S,A,B, x_1,x_2,x_3,x_4,x_5, Y_1,Y_2,Y_3,Y_4,Y_5,Y_6\}, S, P')$

EJERCICIO : Poner en FNC las gramáticas cuyas producciones son:
(HOJA)

①
- $S ::= A \mid \lambda$
- $A ::= ABBA \mid a$
- $B ::= bDb$
- $D ::= c$

②
- $S ::= 1A \mid 1B$
- $A ::= 0 \mid 0S \mid 1AA$
- $B ::= 1 \mid 1S \mid 0BB$

EJ: Sea el lenguaje $L = \{a^m b^n c^p d^n e^m ;\ m \geq 3, n \geq 1, p \geq 3\}$

1) Obtener una gramática que genere L con 6 producciones
- $S ::= aSe \mid aaAee$
- $A ::= bAd \mid bBd$
- $B ::= cC \mid ccc$

$G = \{(a,b,c,d,e)\ (S,A,B)\ S\ P\}$

2) obtener una gramática equivalente en FNC
- $S ::= aSE \mid aFAEE$
- $F ::= a$
- $E ::= e$
- $A ::= bAD \mid bBD$ → en cada producción un no terminal seguido de todos
- $D ::= d$
- $B ::= cB \mid cCC$
- $C ::= c$

$G = \{(abcde)(SABEF)\ S\ P\}$

3) Obtener una gramática equivalente en FNG

1)
- $S ::= x_1 S x_2 \mid x_1 x_1 A x_2 x_2$
- $A ::= x_3 A x_4 \mid x_3 B x_4$
- $B ::= x_5 B \mid x_5 x_5 x_5$

	2)		
$x_1 ::= a$	$S ::= x_1 Y_1$	$A ::= x_3 Y_5$	$B ::= x_5 B$
$x_2 ::= e$	$Y_1 ::= S Y_2$	$Y_5 ::= A x_4$	$B ::= x_5 Y_7$
$x_3 ::= b$	$S ::= x_1 Y_2$	$A ::= x_3 Y_6$	$Y_7 ::= x_5 x_5$
$x_4 ::= d$	$Y_2 ::= x_1 Y_3$	[illegible]	

EJ: $G = (\{0,1\}, \{S, A, B\}, S, P)$

$P =$ $S ::= AB \mid 0S1 \mid 0AB \mid 0B \mid 0A \mid 0 \mid B1 \mid 1 \mid \lambda$

$A ::= 0AB \mid 0B \mid 0A \mid 0$

$B ::= B1 \mid 1$

Reglas que ya están en FNC $\left|\begin{array}{l} S ::= AB \mid 0 \mid 1 \mid \lambda \\ A ::= 0 \\ B ::= 1 \end{array}\right.$ $\left|\begin{array}{l} X_1 ::= 0 \\ X_2 ::= 1 \end{array}\right.$

La regla $S ::= 0S1$ puede sustituirse $\longrightarrow$ $\left|\begin{array}{l} S ::= X_1 Y_1 \\ Y_1 ::= S X_2 \end{array}\right.$

$S ::= 0AB \longrightarrow \left|\begin{array}{l} S ::= X_1 Y_2 \\ Y_2 ::= AB \end{array}\right.$ $\quad S ::= 0B \rightarrow S ::= X_1 B$ $\quad S ::= 0A \rightarrow S ::= X_1 A$ $\quad S ::= B1 \longrightarrow S ::= B X_2$

$A ::= 0AB \longrightarrow A ::= X_1 Y_2$ $\quad A ::= 0B \rightarrow A ::= X_1 B$ $\quad A ::= 0A \rightarrow A ::= X_1 A$

$B ::= B1 \longrightarrow B ::= B X_2$

La gramática equivalente en FNC es:

$G' = (\{0,1\}, \{S, A, B, Y_1, X_2, Y_1, Y_2\}, S, P')$

donde $P' = \left|\begin{array}{l} S ::= AB \mid BX_2 \mid X_1 B \mid X_1 A \mid X_1 Y_1 \mid X_1 Y_2 \mid 0 \mid 1 \mid \lambda \\ A ::= 0 \mid X_1 A \mid X_1 B \mid X_1 Y_2 \\ B ::= 1 \mid BX_2 \\ X_1 ::= 0 \\ X_2 ::= 1 \\ Y_1 ::= S X_2 \\ Y_2 ::= AB \end{array}\right.$

- **FORMA NORMAL DE GREIBACH: (FNG)**

Toda gramática independiente de contexto puede reducirse a otra equivalente sin reglas recursivas por la izquierda.

Una gramática está en FNG si sus producciones son de la forma:

$$\begin{bmatrix} A ::= aX \\ S ::= \lambda \end{bmatrix} \quad A ::= aABCD \qquad a \in \Sigma_T \quad X \in \Sigma_N^*$$

El consecuente de todas las producciones (salvo $S ::= \lambda$) es un símbolo terminal seguido de símbolos no terminales o de la palabra vacía λ.

EJ: $L = \{ a^n c^r d^p b^n \; ; \; n, p \geq 1 \}$

$S ::= aSb \mid aAb \longrightarrow S ::= aSB \mid aAB$

$A ::= cAd \mid cd \qquad A ::= cAD \mid cD$

$B ::= b$

$D ::= d$

5. GRAMÁTICAS LINEALES

- **GRAMÁTICAS EQUIVALENTES:** Para cada gramática lineal derecha (GLD) existe otra GLD equivalente que no contiene ninguna producción de la forma:

$$\begin{cases} A ::= \lambda \text{ y} & A \in \Sigma_N \quad a \in \Sigma_T \\ A ::= aS & S \text{ el axioma.} \end{cases}$$

Para cada producción de la forma $S ::= X$ añadimos una producción de la forma $B ::= X$ y la producción $A ::= aS$ se sustituye por $A ::= aB$

EJ: $\begin{cases} S ::= bA \\ A ::= aS \mid a \end{cases} \longrightarrow \begin{cases} S ::= bA \\ A ::= aB \mid a \\ B ::= bA \end{cases}$

¿Qué lenguaje genera dicha GLD?

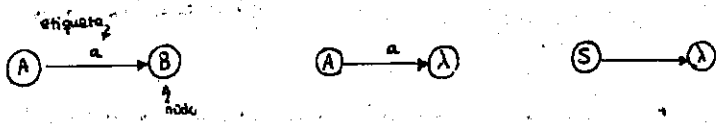
Se halla de la siguiente manera:

1.- Hacemos un grafo con tantos nodos como símbolos no terminales + 1

2.- Etiquetamos cada nodo con los símbolos no terminales y λ.

3.- Para cada producción de la siguiente forma se construyen los grafos:

$A ::= aB$ $A ::= a$ $S ::= \lambda$



* El lenguaje de esta gramática será el que se obtenga al ir de (S) a (λ) por cualquier camino y leyendo las etiquetas de izquierda a derecha.

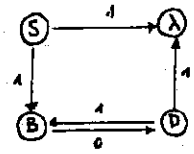
ASÍ SE OBTIENE EL GRAFO DE LA GLD.

PARA CONSTRUIR LA GLI EQUIVALENTE ES A PARTIR DEL SIGUIENTE PASO.

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

$$P \equiv \begin{cases} S ::= 1B \mid 1 \\ B ::= 0S \end{cases} \longrightarrow \begin{cases} S ::= 1B \mid 1 \\ B ::= 0D \\ D ::= 1B \mid 1 \end{cases} \Rightarrow GLD$$

↑ primero la etiqueta y luego el nodo.



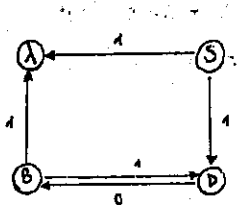
$L(G_D) = 1(01)^n \quad 1(01)^*$

PARA TODA GLD EXISTE GLI EQUIVALENTE.

Se obtiene de la siguiente manera (partiendo del grafo de la GLD).

4. Se construye un grafo cambiando el nodo (S) por (λ) y el resto permanecen igual.

5. Cambiar el sentido de las flechas, dejándolas etiquetadas como estaban.



$S ::= 1$
$S ::= D1$
$B ::= 1$
$B ::= D1$
$D ::= B0$

↑ primero el nodo y

Partiendo de (S) y leyendo de derecha a izquierda, al ir a (λ) obtenemos el lenguaje generado (el mismo de antes).

EJERCICIO: Dada la gramática cuyas producciones son P, probar que es ambigua.

$P \equiv$ $S ::= aSa \;(1) \mid bSb \;(2) \mid abSba \;(3) \mid baSab \;(4) \mid c \;(5) \mid \lambda \;(6)$

Tipos de producciones: $\begin{cases} A ::= x & x \in \Sigma^+ \\ S ::= \lambda \end{cases}$

$L = \{ xyx^{-1} : y \in \{\lambda, c\},\ x \in \{a,b\}^+ \}$

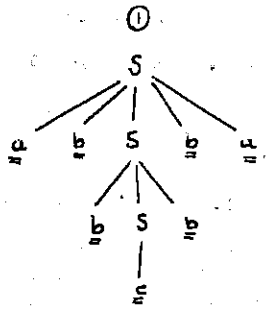
Para comprobar si es ambigua cojo una palabra:

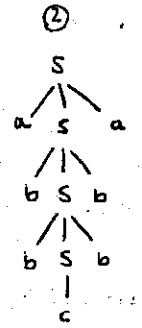
abbcbba

↳ ojo!! Si esta palabra no es ambigua no puedo decir que la gram. no lo sea.

① $S \xrightarrow{3} abSba \xrightarrow{2} abbSbba \xrightarrow{5} abbcbba$

② $S \xrightarrow{1} aSa \xrightarrow{2} abSba \xrightarrow{2} abbSbba \xrightarrow{5} abbcbba$





Quitando la producción 3 y 4 la gramática deja de ser ambigua.

EJERCICIO: $G = (\{a,b\}, \{S,A,B\}, S, \{S ::= aAb \quad A ::= aAb \mid aB \mid a \quad B ::= aB \mid a\})$

$L = \{ a^n b^m,\ n > m \quad m \geq 1 \}$

Gramática equivalente → $S ::= aSb \mid aS \mid aab$

EJERCICIO: Diseñar la gramática que genera el lenguaje:

$L = \{ a^n b^m c^p : m = n+p,\ n \geq 0,\ p \geq 0 \}$

$\downarrow$

$a^n b^n b^p c^p$

$S ::= AB$

$A ::= aAb \mid ab$

$B ::= bBc \mid \lambda$

EJERCICIO: Gramática que genera los nº pares en el Sistema Decimal.

$$S ::= 1A \mid 2A \mid \dots \mid 9A \mid 0 \mid 2 \mid 4 \mid 6 \mid 8$$
$$A ::= 0A \mid 1A \mid \dots \mid 9A \mid 0 \mid 2 \mid 4 \mid 6 \mid 8$$

EJERCICIO: $L = \{ 0^i 1^j 2^k : i=j \text{ ó } j=k \}$

$0^i 1^i 2^k$, $0^i 1^k 2^k$ ⟶

$S ::= AB \mid CD$
$A ::= 0A1 \mid \lambda$
$B ::= 2B \mid \lambda$
$C ::= 0C \mid \lambda$
$D ::= 1D2 \mid \lambda$

EJ: Comprobar si es ambigua:

$$S ::= \overset{1}{S}T \mid \overset{2}{T}$$
$$T ::= \underset{3}{(}S\underset{4}{)} \mid ()$$

Cojo la palabra → (())() (S T)

$S \to ST \to TT \to (S)T \to (T)T \to ((\,))T \to (())()$

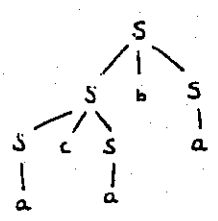
no es ambigua esta sentencia

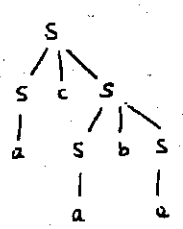
Hay un sólo árbol

EJ: Dada la gramática cuyas producciones son $S ::= \overset{1}{S}bS \mid \overset{2}{S}cS \mid \overset{3}{a}$, comprobar que es ambigua, hallar el lenguaje que genera, la gramática equivalente lineal, la ER.

a) DI $S \xrightarrow{1} SbS \xrightarrow{2} ScSbS \xrightarrow{3} acSbS \xrightarrow{3} acabS \xrightarrow{3} acaba$

DI $S \xrightarrow[2]{} ScS \xrightarrow[3]{} acS \xrightarrow[1]{} acSbS \xrightarrow[3]{} acabS \xrightarrow[3]{} acaba$



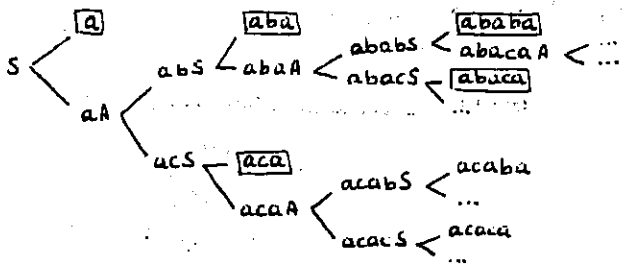


b) L = b y c intercaladas con a y que empieza y termina por a.

c) S ::= a | aA

A ::= bS | cS ← así intercalo b y c con a.

d)



ER → a[(b+c)a]*

EJ: Depurar la gramática cuyas producciones son:

S ::= AB | 0S1 | A | λ	regla no generativa →	S ::= AB | 0S1 | A
A ::= 0AB | λ		A ::= 0AB | λ
B ::= B1 | λ		B ::= B1 | λ

Reglas reductoras →

Quitar: A ::= λ

Añadir: A ::= 0B | S ::= B | λ → S ::= AB \| 0S1 \| A \| B \| λ

A ::= 0AB \| 0B

B ::= B1 \| λ

→ Quitar: B ::= λ

Añadir: S ::= A | λ →

A ::= 0A | 0 B = 1

S ::= AB \| 0S1 \| A \| B \| λ Reglas de redenominación

A ::= 0AB \| 0B \| 0A \| 0

B ::= B1 \| 1 → Quitar S ::= A → S ::= 0AB \| 0B \| 0A \| 0 →

S ::= B \| 1 \| λ

↑ Añadir

S ::= AB | 0S1 | λ | 0AB | 0B | 0A | 0 | B | 1	
A ::= 0AB | 0B | 0A | 0	→ GRAMÁTICA LIMPIA.
B ::= B1 | 1	

EJ Dado el lenguaje $L = \{ a^m b^n c^k \;\; k = m - n \}$ hallar dos gramáticas de contexto libre que genere L

si $m = k + n$ $L = \{ a^k a^n b^n c^k \}$

S ::= aSc | aAc

A ::= aAb | ab

EJ: Dada una gramática cuyas producciones son P:

- Escribir G' equivalente a G bien formada.
- Descripción del lenguaje que genera G.
- G'' que genere el mismo lenguaje, pero más simple.

$$P.= \begin{cases} S ::= AB \mid \lambda \\ A ::= aBb \mid B \mid \lambda \\ B ::= bAa \mid A \mid \lambda \end{cases}$$

a) R. de redenominación

Quitamos: $A ::= B$ ⟹ $\begin{cases} S ::= AB \mid \lambda \\ A ::= aBb \mid \lambda \mid bAa \mid \not{A} \text{ innecesaria} \\ B ::= bAa \mid A \mid \lambda \end{cases}$ →

Añadir: $A ::= bAa \mid A \mid \lambda$

Quitar: $B ::= A$ ⟹ $\begin{cases} S ::= AB \mid \lambda \\ A ::= aBb \mid \lambda \mid bAa \\ B ::= bAa \mid \lambda \mid aBb \end{cases}$ →

Añadir: $B ::= aBb \mid \lambda \mid bAa$

Reductora

Quitar: $A ::= \lambda$ ⟹ $\begin{cases} S ::= AB \mid B \mid \lambda \\ A ::= aBb \mid bAa \mid ba \mid \not{\lambda} \\ B ::= bAa \mid aBb \mid ba \mid \lambda \end{cases}$ →

Añadir: $\begin{cases} S ::= B \\ A ::= ba \\ B ::= ba \end{cases}$

Quitar: $B ::= \lambda$ ⟹ $\begin{cases} S ::= AB \mid A \mid B \mid \lambda \\ A ::= aBb \mid bAa \mid ba \mid ab \\ B ::= bAa \mid aBb \mid ba \mid ab \end{cases}$ →

Añadir: $\begin{cases} S ::= A \mid \lambda \\ A ::= ab \\ B ::= ab \end{cases}$

Quitar: $S ::= A$, $S ::= B$ ⟹ $\begin{cases} S ::= AB \mid aBb \mid bAa \mid ba \mid ab \mid \lambda \\ A ::= aBb \mid bAa \mid ba \mid ab \\ B ::= bAa \mid aBb \mid ba \mid ab \end{cases}$ GRAMÁTICA LIMPIA

Añadir: $\begin{cases} S ::= aBb \mid bAa \mid ba \mid ab \\ S ::= bAa \mid aBb \mid ba \mid ab \end{cases}$

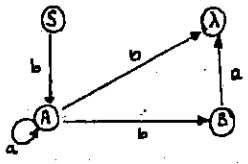
b) $L(G) = \{ x \in \{a,b\}^* \;/\; N_a^x(x) = N_b^x(x) \}$

EJ: Sean las producciones

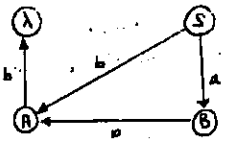
$$\begin{cases} S ::= bA \\ A ::= aA \mid bB \mid b \\ B ::= a \end{cases}$$

a) Calcular la GLI equivalente.

b) Ver el lenguaje generado mediante expresiones regulares



$L = b \cdot a^* (b + ba)$



GLI

$$\begin{cases} S ::= Ba \mid Ab \\ B ::= Ab \\ A ::= Aa \mid b \end{cases}$$

EJ: Pasar la gramática a FNC.

$$P = \begin{cases} S ::= A \mid \lambda \\ A ::= ABBA \mid a \\ B ::= bDb \\ D ::= c \end{cases}$$

Primero hay que depurarla

Quitar: $S ::= A$

Añadir: $S ::= ABBA \mid a$

$$\begin{cases} S ::= ABBA \mid a \mid \lambda \\ A ::= ABBA \mid a \\ B ::= bDb \\ D ::= c \end{cases}$$

Pasar a FNC

$$\begin{cases} S ::= ABBA \mid a \mid \lambda \\ A ::= ABBA \mid a \\ B ::= XDX \\ D ::= c \\ X ::= b \end{cases} \longrightarrow \begin{cases} S ::= AY_1 \mid a \mid \lambda \\ Y_1 ::= BY_2 \\ Y_2 ::= BA \\ A ::= AY_4 \mid a \\ B ::= XY_3 \\ Y_3 ::= DX \\ X ::= b \\ D ::= c \end{cases}$$

EJ: Sea la G con las producciones $S ::= AB \mid \lambda$ $A ::= aBb \mid B \mid \lambda$ $B ::= bAa \mid A \mid \lambda$. Construir una G' bien formada equivalente. Hallar el lenguaje generado y otra gramática más simple.

$P \equiv \begin{cases} S ::= AB \mid \lambda \\ A ::= aBb \mid B \mid \lambda \\ B ::= bAa \mid A \mid \lambda \end{cases}$ → Quitar: $A ::= B$ Añadir: $A ::= bAa \mid \lambda$ → $\begin{cases} S ::= AB \mid \lambda \\ A ::= aBb \mid bAa \mid \lambda \\ B ::= bAa \mid A \mid \lambda \end{cases}$ → Quitar: $B ::= A$ Añadir: $B ::= aBb \mid bAa \mid \lambda$

$\begin{cases} S ::= AB \mid \lambda \\ A ::= aBb \mid bAa \mid \lambda \\ B ::= bAa \mid aBb \mid \lambda \end{cases}$ → Quitar: $A ::= \lambda$ Añadir: $S ::= B$ $A ::= ba$ $B ::= ba$ → $\begin{cases} S ::= AB \mid B \mid \lambda \\ A ::= aBb \mid bAa \mid ba \\ B ::= bAa \mid aBb \mid \lambda \mid ba \end{cases}$ → Quitar: $B ::= \lambda$ Añadir: $S ::= A \mid \lambda$ $A ::= ab$ $B ::= ab$

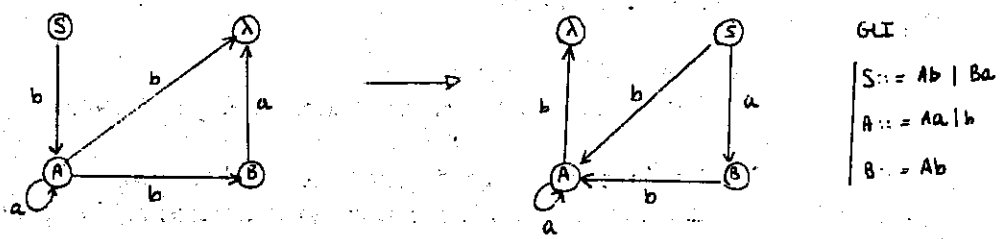
$\begin{cases} S ::= AB \mid B \mid A \mid \lambda \\ A ::= aBb \mid bAa \mid ba \mid ab \\ B ::= bAa \mid aBb \mid ba \mid ab \end{cases}$ → Quitar $S ::= A$, $S ::= B$ Añadir $S ::= aBb \mid bAa \mid ba \mid ab$ → $\begin{cases} S ::= AB \mid aBb \mid bAa \mid ab \mid ba \mid \lambda \\ A ::= aBb \mid bAa \mid ab \mid ba \\ B ::= bAa \mid aBb \mid ab \mid ba \end{cases}$

$L = \{x \in \{a,b\}^* : N_a(x) = N_b(x)\}$

Una más sencilla: $P \equiv S ::= abS \mid baS \mid aSb \mid bSa \mid Sab \mid Sba \mid \lambda$

EJ. Sea la gramática G de $P \equiv \begin{cases} S ::= bA \\ A ::= aA \mid bB \mid b \\ B ::= a \end{cases}$

Calcular la gramática lineal izquierda equivalente y la expresión regular de L.



GLI:

$\begin{cases} S ::= Ab \mid Ba \\ A ::= Aa \mid b \\ B ::= Ab \end{cases}$

ER ≡ $L = ba^*(b + ba)$

DEPARTAMENTO DE INTELIGENCIA ARTIFICIAL

FACULTAD DE INFORMÁTICA
UNIVERSIDAD POLITÉCNICA DE MADRID

INFORMÁTICA TEÓRICA

Curso 2004-2005

Prácticas Tema 2

GRAMÁTICAS FORMALES

Práctica 2.1: Construcción de gramáticas

Construir gramáticas que generen los siguientes lenguajes, indicando de qué tipo es la gramática propuesta.

1.- $L = \{a^m b^n \mid m>0, n\geq 0\}$
(Indicación: Más adelante se probará que $LG_3(\Sigma) = LER(\Sigma) = LAF(\Sigma)$.)

2.- $L = \{a0^n1^n \mid n>0\} \cup \{0^n b1^{2n} \mid n\geq 0\}$
(Ind.: ¿Cómo puede ser el primer paso de las derivaciones en una gramática que genere la unión de dos lenguajes?)

3.- $L = \{a^m b^n \mid n>m\}$

4.- $L = \{(ac)^m b^n \mid n\neq m, n,m>0\}$
(Ind.: Generar por separado palabras m>n y palabras m<n)

5.- $L = \{a^m b^n c^p d^n e^m f^k (gh^*)^k \mid m>1, n\geq 0, p>2, k>1\}$
(Ind.: Considerar el lenguaje como concatenación de otros dos)

6.- $L = \{ a^m b^n c^k \mid m=n+k \}$

7.- L es el lenguaje formado por las palabras sobre el alfabeto $\Sigma = \{a,b\}^*$ tales que inmediatamente después de cada a hay una b.

8.- $L = \{x \in \{0,1\}^* \mid N_0(x) = 2.N_1(x)\}$ ($N_0(x)$ es el número de ceros de la palabra x).

9.- $L = \{x \in N \mid x \text{ es múltiplo de } 3\}$

a) 003 y 000 son válidas.

b) No son palabras del lenguaje las que tienen ceros no significativos a la izquierda.

10.- Un pequeño descanso. ¿Qué lenguaje genera la siguiente gramática?:

A ::= aABC | abC
CB ::= BC
bB ::= bb
bC ::= bc
CC ::= cc

11.- $L = \{a^n b^n c^n d^n \mid n>0\}$

12.- $L = \{ a^m b^n c^k \mid m>n>k\geq 0 \}$
(Ind.: Generar el número adecuado de a, b, y c y luego ordenarlas).

13.- Otro descanso. ¿Qué lenguaje genera la siguiente gramática?:

S ::= BAB
BA ::= BC
CA ::= AAC
CB ::= AAB
A ::= 0
B ::=1

14.- Vamos acabando. Ahí van otros tres lenguajes más o menos relacionados:

$L = \{xcx^{-1} \mid x \in \{a,b\}^*\}$

$L = \{xx^{-1} \mid x \in \{a,b\}^*\}$

$L = \{x \in \{a,b\}^* \mid x = x^{-1}\}$

Práctica 2.2: Ambigüedad en Lenguajes Independientes del Contexto

1.- Dada la gramática $G = (\{x, y, z, \vee, \wedge\}, \{S\}, S, \{ S ::= S \vee S \mid S \wedge S \mid x \mid y \mid z \})$, comprobar que es ambigua estudiando las derivaciones y árboles de derivación de la palabra $w = x \vee y \wedge z$.

2.- Sea la gramática G definida por sus producciones

$$S ::= aSa \mid bSb \mid abSba \mid baSab \mid c \mid \lambda$$

a) ¿Qué tipo de gramática es según la jerarquía de Chomsky?
b) ¿Qué lenguaje genera?
c) Probar que es ambigua mostrando una palabra ambigua de longitud 7.
d) Construir una gramática G' equivalente a la gramática G, no ambigua, mostrando que la palabra elegida en el apartado c) no es ambigua.

3.- a) Obtener una gramática independiente de contexto que genere el lenguaje siguiente:

$$L = \{ a^m b^n c^p \mid m = n \text{ ó } m = p \}$$

b) Estudiar la ambigüedad de la gramática construida en el apartado anterior.

Práctica 2.3: Depuración de gramáticas

1 Introducción

Considérese una gramática independiente del contexto (g.i.c.) $G = (\Sigma_T, \Sigma_N, S, P)$ y una palabra $w \in \Sigma_T^*$.

Problema de la pertenencia: Dada una g.i.c. G y una cadena $w \in \Sigma_T^*$, ¿se cumple que $w \in L(G)$?

Motivación:

G es una gramática del lenguaje de programación C.
w es un programa escrito en C.
¿Es w sintácticamente correcto?

Comprobación: Análisis sintáctico. Dos formas posibles:

1. *Análisis descendente.* Comenzar con el axioma S y tratar de derivar la cadena w. (búsqueda exhaustiva).
2. *Análisis ascendente.* Empezar por la cadena w, y derivar 'hacia atrás' hasta alcanzar el axioma S.

Teorema 1.1. *Si G es una g.i.c. que no contiene reglas de la forma*

$$A ::= \lambda$$
$$A ::= B$$

donde $A, B \in \Sigma_N$, *entonces se puede determinar si* $w \in L(G)$ *o si* $w \notin L(G)$.

Para comprobar si una palabra pertenece o no a un lenguaje dado $L(G)$ es más eficiente (conlleva menos tiempo de ejecución) partir de una gramática G' equivalente a G, esto es, $L(G') = L(G)$, que sea lo más sencilla posible.

La depuración de gramáticas persigue este objetivo. Conseguir gramáticas bien formadas y, por tanto, sencillas para optimizar el proceso de análisis sintáctico de los compiladores. Las gramáticas bien formadas son también imprescindibles en la obtención de muchos resultados teóricos sobre g.i.c.

2 Depuración de gramáticas

La depuración de una gramática $G = (\Sigma_T, \Sigma_N, S, P)$ consiste en una serie de transformaciones tras las cuales se obtiene otra gramática $G' = (\Sigma_T, \Sigma_N', S, P')$ equivalente, $L(G) = L(G')$, y que no contiene:

1. axioma inducido (en el caso de que $S ::= \lambda \in P$),

2. reglas innecesarias,
3. reglas no generativas,
4. símbolos inaccesibles,
5. reglas de redenominación ni
6. reglas reductoras.

Definición 2.1. Una gramática se dice que es una **gramática bien formada** si no posee reglas innecesarias, reglas no generativas, símbolos no accesibles, reglas de redenominación ni reglas reductoras. Si, además, posee la regla $S ::= \lambda$ tampoco se permite la existencia del axioma inducido.

2.1. Axioma inducido

En las gramáticas bien formadas no se permite que el axioma esté inducido (aparezca en la parte derecha de alguna producción) si la regla $S ::= \lambda \in P$. Para eliminar el axioma inducido en estos casos, simplemente se añade un nuevo símbolo no terminal S' que pasa a ser el nuevo axioma y se añade la producción $S' ::= S$. Por tanto, la nueva gramática será:

$G' = (\Sigma_T, \Sigma_N \cup \{S'\}, S', P \cup \{S' ::= S\})$.

Fijarse que se consigue eliminar el axioma inducido pero a expensas de introducir una regla de redenominación que habrá que eliminar posteriormente para depurar la gramática.

2.2. Reglas innecesarias

Estas reglas son de la forma $A ::= A$, $A \in \Sigma_N$. Se eliminan siempre sin que se modifique el lenguaje generado por la gramática G.

2.3. Reglas no generativas

Son reglas que no se emplean nunca en la generación de palabras del lenguaje. Por tanto, su eliminación no modifica el lenguaje generado por la gramática de partida.

Algoritmo para determinar el conjunto de producciones generativas:

Sea $P_G = \{A ::= x \mid \exists\ S \rightarrow_+ uAv \rightarrow_+ uxv \rightarrow_+ y \in \Sigma_T^*\}$ el conjunto de las producciones generativas.

Paso 1 $P_0 = N_0 = \varnothing$, $i = 0$.

Paso 2 $P_{i+1} = \{A ::= x \mid x \in (\Sigma_T \cup N_i)^*\}$

$N_{i+1} = \{A \in \Sigma_N \mid \exists\ A ::= x \in P_{i+1}\}$

Paso 3 $N_{i+1} \neq N_i$ hacer $i = i + 1$ y volver al Paso 2.

Paso 4 $N_{i+1} = N_i$ entonces $P_G = P_{i+1}$. Fin.

Por tanto, en la depuración se eliminarán todas las producciones que no se encuentren en el conjunto P_G.

Ejemplo:

Sea $G = (\{a, b\}, \{S, A\}, S, \{S ::= aSb \mid aAb \mid ab\})$. Examinemos esta gramática G para ver si posee alguna regla no generativa. Apliquemos el algoritmo de obtención del conjunto de producciones generativas P_G.

Paso 1: $P_0 = N_0 = \varnothing$, $i = 0$.

Paso 2: $P_1 = \{A ::= x \mid x \in \Sigma_T\} = \{S ::= ab\}$

$N_1 = \{S\}$.

Paso 3: Como $N_1 \neq N_0$, volvemos al Paso 2 y seguimos calculando P_{i+1} pero ahora con $i = 1$.

Paso 2: $P_2 = \{A ::= x \ / \ x \in (\Sigma_T \cup N_1) = \{S ::= a \ / \ aSb\}$

$N_2 = \{S\}$

Paso 4: Ahora si se verifica que $N_2 = N_1$. Por tanto, $P_G = P_2$.

Entonces, se eliminan las reglas no generativas $P - P_G = \{S ::= aAb\}$.

La gramática resultante $G' = (\{a,b\}, \{S, A\}, S, \{S ::= aSb \mid ab\})$ es equivalente a G pero no posee reglas no generativas.

2.4. Símbolos inaccesibles

Son los símbolos no terminales a los que no se puede acceder por medio de derivaciones desde el axioma de la gramática.

Algoritmo para determinar el conjunto de símbolos de Σ_N accesibles desde el axioma S:

Sea el conjunto $N_A = \{A \in \Sigma_N / \exists\ S \rightarrow_+ xAy\}$ formado por los símbolos no terminales accesibles desde el axioma.

Paso 1 $N_0 = \{S\},\ i = 0.$

Paso 2 $N_{i+1} = N_i \cup \{B \in \Sigma_N / \exists\ A ::= xBy,\ A \in N_i\}$

Paso 3 $N_{i+1} \neq N_i$ hacer i = i + 1 y volver al Paso 2.

Paso 4 $N_{i+1} = N_i$ entonces $N_A = N_i$. Fin.

Por tanto, en N_1 se encuentran los símbolos accesibles desde el axioma S utilizando una producción. En N_2 se encuentran los símbolos accesibles desde el axioma empleando dos producciones y, así sucesivamente. El conjunto $\Sigma_N - N_A$ contiene todos los símbolos inaccesibles desde el axioma.

Se eliminarán todas las producciones en las que aparezca algún símbolo inaccesible.

Ejemplo:

Sea la gramática $G = (\{a,b\}, \{S, A\}, S, \{S ::= aSb \mid ab, C ::= a\})$

Por inspección directa se observa que esta gramática posee un símbolo inaccesible, el símbolo C. Aplicando el algoritmo se obtiene el conjunto de símbolos accesibles desde el axioma N_A.

Paso 1: $N_0 = \{S\},\ i = 0.$

Paso 2: $N_1 = \{S\} \cup \{B \in \Sigma_N \mid \exists\ A ::= xBy,\ A \in N_0\} = \{S\}$

Paso 4: $N_1 = N_0.$ Por tanto, $N_A = N_0 = \{S\}$.

Entonces, se elimina la producción $C ::= a$ ya que el símbolo C es inaccesible.

La gramática obtenida $G' = (\{a, b\}, \{S\}, S, \{S ::= aSb \mid ab\})$ es equivalente a la G.

2.5. Reglas de redenominación

Son las reglas de reeescritura del tipo: $A ::= B$ donde $A, B \in \Sigma_N$.

Eliminación de reglas de redenominación:

Se eliminan sucesivamente las producciones $A ::= B$ de la gramática y se añaden:

Por cada producción del tipo $B ::= x,\ B \in \Sigma_N$ y $x \in (\Sigma_T \cup \Sigma_N)^*$, que exista en la gramática se añade una producción $A ::= x$ manteniéndose el resto de producciones de la gramática.

Ejemplo:

Sea la gramática $G = (\{a,b\}, \{S, A\}, S, \{S ::= aSb \mid A, A ::= ab\})$

Esta gramática posee una regla de redenominación $S ::= A$. Se elimina y se añade la producción $S ::= ab$. La gramática resultante es $G' = (\{a, b\}, \{S, A\}, S, \{S ::= aSb \mid ab, A ::= ab\})$. Esta gramática todavía no está depurada ya que posee un símbolo A inaccesible desde el axioma. Sin embargo, es fácil observar que es equivalente a la gramática de partida ya que: $L(G) = L(G') = \{a^n b^n, n > 0\}$.

2.6. Reglas reductoras

Las reglas reductoras son del tipo $A ::= \lambda$ donde $A \in \Sigma_N - \{S\}$. Se eliminan estas producciones añadiendo:

Eliminación de reglas reductoras:

Por cada producción $B :: = x_1A_1x_2A_2x_3 \ldots x_kA_kx_{k+1}$, donde hay k apariciones del símbolo no terminal A para el que existe una producción reductora $A :: = \lambda$, se añaden las producciones resultantes de eliminar 1, 2, ... todas las veces posibles y de todas las formas posibles el símbolo A. Así, se añaden producciones como las siguientes:

$$B :: = x_1x_2A_2x_3 \ldots x_kA_kx_{k+1}$$

$$B :: = x_1A_1x_2x_3 \ldots x_kA_kx_{k+1}$$

...

$$B :: = x_1A_1x_2A_2x_3 \ldots x_kx_{k+1}$$

resultado de eliminar de la citada producción el símbolo A una vez de todas las formas posibles.

Se añaden también las producciones

$$B :: = x_1x_2x_3 \ldots x_kA_kx_{k+1}$$

$$B :: = x_1A_1x_2x_3x_4 \ldots x_kA_kx_{k+1}$$

...

$$B :: = x_1A_1x_2A_2x_3 \ldots x_{k-1}x_kx_{k+1}$$

resultado de eliminar dos apariciones del símbolo A de todas las formas posibles, y así sucesivamente hasta añadir la producción

$$B :: = x_1x_2x_3 \ldots x_kx_{k+1}$$

en la que se han eliminado todas las apariciones del símbolo A.

Ejemplo:

Sea la gramática $G = (\{a, b\}, \{S, A\}, S, \{S ::= aSb \mid aAb, A ::= \lambda\})$. Esta gramática genera el lenguaje $L(G) = \{a^n b^n, n > 0\}$.

Posee una regla reductora $A ::= \lambda$.

Esta regla se elimina y se añaden producciones resultantes de eliminar A en la parte derecha de la producción $S ::= aAb$. Es decir, se añade la producción $S ::= ab$.

La gramática resultante $G' = (\{a, b\}, \{S, A\}, S, \{S ::= aSb \mid aAb \mid ab\})$ es equivalente a la gramática G pero todavía no está depurada. Posee una regla no generativa como se ha estudiado en la sección 2.3.

Teorema 2.1. Sea G una g.i.c. cualquiera. Entonces existe una g.i.c. G', sin el axioma inducido (si $S ::= \lambda \in P$), sin reglas innecesarias, reglas de redenominación, reglas reductoras, reglas no degenerativas ni símbolos inaccesibles tal que $L(G) = L(G')$.

Dem.: Hay que demostrar que cada transformación aplicada no modifica la capacidad generativa de la gramática G de partida.

ORDEN DE APLICACIÓN:

No hay un orden determinado en el que aplicar las diferentes transformaciones que depuran una gramática. Sin embargo, es aconsejable realizar primero los pasos que implican eliminación de reglas y símbolos (reglas no generativas y símbolos no accesibles) y posteriormente eliminar el axioma inducido (en el caso de que $S ::= \lambda \in P$) y las reglas reductoras y de redenominación. Sin embargo, quizás haya que repetir algún proceso varias veces hasta conseguir la depuración de una gramática.

Ejercicios de depuración de gramáticas

1.- Depurar la gramática G con $\Sigma_T = \{a, b\}$; $\Sigma_N = \{S, A, B, C, E\}$, axioma S y producciones:

$$S ::= ABA \mid ABE$$
$$A ::= a \mid \lambda \mid E$$
$$B ::= A \mid b$$
$$E ::= E\,a$$
$$C ::= a$$

2.- Obtener una gramática bien formada equivalente a la del ejercicio 2.2.3

Práctica 2.4: Formas Normales o Canónicas

1 Introducción

Las producciones de una gramática de contexto libre, según su definición, tienen una estructura sencilla. El antecedente es un símbolo no terminal, y la única restricción para el consecuente es que no sea la palabra vacía. En múltiples ocasiones es conveniente que la parte derecha de las producciones, que, en principio puede ser una palabra cualquiera, no vacía, con símbolos terminales y no terminales en cualquier orden y número, tenga una estructura más rígida o que responda a una determinada forma.

Por ello, se definen diferentes estructuras de los consecuentes de las producciones de una gramática de contexto libre, denominándose a las gramáticas cuyas producciones responden a un determinado tipo de estructura, gramática en forma normal o estándar.

Puede ser el caso de demostrar determinadas propiedades de los lenguajes independientes del contexto, para lo cual puede ser útil que la gramática de contexto libre que genera un determinado lenguaje se suponga esté en forma normal de Chomsky, o en problemas de análisis sintáctico, en los que interesa que las gramáticas que se utilizan no sean recursivas por la izquierda, para lo que es conveniente utilizar gramáticas en forma normal de Greibach.

Estos dos tipos de formas normales, la forma normal de Chomsky y la forma normal de Greibach son las más utilizadas, y sobre ellas trata esta práctica.

2 Forma Normal de Chomsky

Definición: Una gramática de contexto libre (o gramática independiente del contexto, o gramática tipo 2) se dice que está en **forma normal de Chomsky** cuando sus producciones son de la forma

$$\begin{array}{c} \\ \text{ó} \\ \text{ó (eventualmente)} \end{array} \begin{cases} A ::= BC, \\ A ::= a, \\ S ::= \lambda \end{cases}$$

Algoritmo: Para obtener una gramática en forma normal de Chomsky equivalente a una gramática de contexto libre :

- Se parte de una gramática independiente del contexto bien formada.
- Paso 1 : Para aquellas producciones $A ::= x$, $|x| \geq 2$, que no responden a la forma normal de Chomsky, en conjunto, se cambian los símbolos terminales b_i que

aparezcan por símbolos no terminales nuevos X_i y se añaden las producciones $X_i ::= b_i$.

- Paso 2 : Cada producción de la forma $A ::= B_1 B_2 \ldots B_k$, $k > 2$, se reemplaza por:

$$A ::= B_1 Y_1$$
$$Y_1 ::= B_2 Y_2$$
$$\ldots\ldots\ldots\ldots$$
$$Y_{k-2} ::= B_{k-1} B_k$$

donde $Y_1, Y_2, \ldots, Y_{k-2}$ son nuevos símbolos no terminales.

Se ve de forma bastante intuitiva que la gramática obtenida con este proceso es equivalente a la gramática de la que se parte.

(1) Ejemplo 1: Obtener una gramática en FN Chomsky equivalente a la siguiente gramática :

$$S ::= 1A \mid 1B$$
$$A ::= 0 \mid 0S \mid 1AA$$
$$B ::= 1 \mid 1S \mid 0BB$$

(2) Ejemplo 2: Sea la gramática

$$S ::= A \mid \lambda$$
$$A ::= ABBA \mid a$$
$$B ::= bCb$$
$$C ::= c$$

1,- Obtener una gramática bien formada equivalente.

2.- Obtener una gramática en FN Chomsky equivalente.

(3) En el ejemplo anterior sería posible obtener otra gramática, también en FN Chomsky con menos producciones, si se hace alguna pequeña transformación previa.

Sugerir y formalizar alguna condición y las correspondientes transformaciones para obtener gramáticas en FN Chomsky más reducidas.

3 Forma Normal de Greibach

Definición: Una gramática de contexto libre se dice que está en **forma normal de Greibach** si sus producciones son de la forma

$$A ::= ax \qquad a \in \Sigma_T \qquad x \in \Sigma_N^*$$
$$S ::= \lambda$$

es decir, salvo la producción $S ::= \lambda$, el consecuente de todas las producciones es un símbolo terminal seguido de una palabra formada solamente por símbolos no terminales que puede ser vacía (A ::= aBCDN ó A ::= a).

(4) Para la siguiente gramática indicar qué producciones no están en forma normal de Greibach:

$$S ::= SBA \mid ABaB$$
$$A ::= Sb \mid b$$
$$B ::= aBa \mid b$$

(27/10/05)

TEMA 3: MÁQUINAS SECUENCIALES.

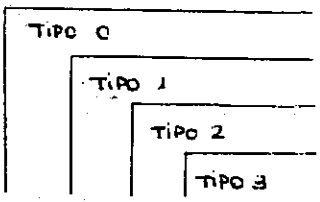
0: JERARQUÍA DE CHOMSKY. RELACIONES.

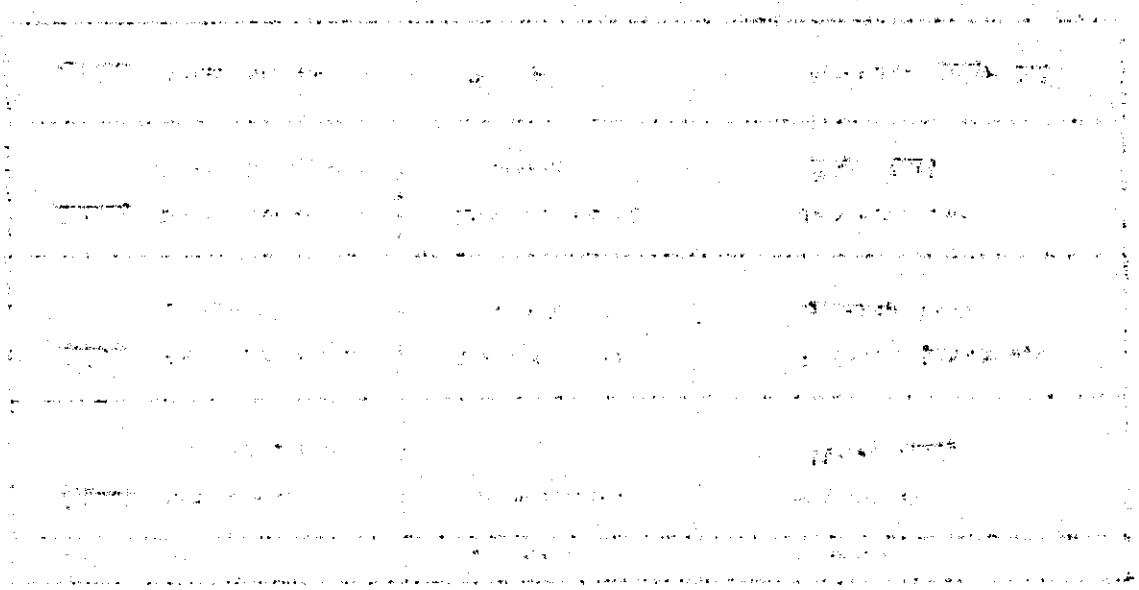
Dado que una gramática describe un lenguaje y que este lenguaje puede ser aceptado por una determinada máquina, entonces es posible establecer la siguiente relación:

GRAMÁTICA	LENGUAJE	MÁQUINA
Tipo 0: Gramática sin restricciones.	Sin restricciones.	Máquina de Turing (MT)
Tipo 1: Gram. sensible al contexto.	Dependiente de contexto	Autómata linealmente acotado (ALA)
Tipo 2: Gram. independiente de contexto.	Independiente de contexto.	Autómata con Pila (AP)
Tipo 3: Gram. regular.	Regular.	Autómata Finito (AF)

Cada uno de estos tipos de máquinas es capaz de resolver problemas cada vez más complicados: los más sencillos corresponden a los AF y los más complejos a las MT.

- JERARQUÍA DE CHOMSKY:





(27/10/05)

TEMA 3: MÁQUINAS SECUENCIALES.

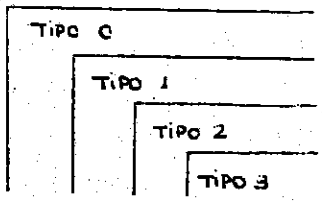
0. JERARQUÍA DE CHOMSKY. RELACIONES.

Dado que una gramática describe un lenguaje y que este lenguaje puede ser aceptado por una determinada máquina, entonces es posible establecer la siguiente relación:

GRAMÁTICA	LENGUAJE	MÁQUINA
Tipo 0: Gramática sin restricciones.	Sin restricciones.	Máquina de Turing (MT)
Tipo 1: Gram. sensible al contexto	Dependiente de contexto	Autómata linealmente acotado (ALA)
Tipo 2: Gram. independiente de contexto.	Independiente de contexto.	Autómata con Pila (AP)
Tipo 3: Gram. regular.	Regular.	Autómata Finito (AF)

Cada uno de estos tipos de máquinas es capaz de resolver problemas cada vez más complicados: los más sencillos corresponden a los AF y los más complejos a los MT.

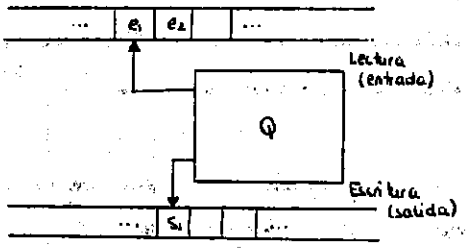
- JERARQUÍA DE CHOMSKY:



1. MÁQUINAS SECUENCIALES. DEFINICIÓN.

Las máquinas secuenciales representan un tipo de autómata que es capaz de, dada una palabra de entrada, generar otra palabra de salida. Para ello se define un conjunto finito de estados que "memorizan" la parte de la palabra de entrada leída en cada momento y generan una salida al mismo tiempo que transitan entre los estados.

Se pueden ver como un autómata que tiene dos cintas asociadas: una de entrada, por la que va leyendo las palabras y otra de salida.



El control Q se encuentra en un estado inicial. Al recibir una entrada se produce una transición de Q a otro estado y se genera una salida.

Los **AUTÓMATAS FINITOS** aceptan **lenguajes regulares**.
Cualquier lenguaje regular tiene un autómata finito.

Una **MÁQUINA SECUENCIAL** se define por la quintupla:

$$M = (\Sigma_E, \Sigma_S, Q, f, g)$$

Σ_E ≡ Alfabeto de entrada.
Las cadenas que lee se construyen sobre este alfabeto.
Σ_E^* => Acepta cualquier cadena.

Σ_S ≡ Alfabeto de salida.

Q ≡ Conjunto de estados (es finito).

f ≡ Función de transición. ①

g ≡ Función de salida o función respuesta. ②

① Indica a qué estado va a transitar para cada par estado-entrada ($Q \times \Sigma_E$)
② Indica el símbolo de salida.

2. TIPOS DE MÁQUINAS SECUENCIALES.

Existen dos tipos de máquinas secuenciales, que se diferencian únicamente en la definición de la función de salida (g).

- **MÁQUINA DE MEALY:**

$$\begin{cases} f: Q \times \Sigma_E \to Q \\ g: Q \times \Sigma_E \to \Sigma_S \end{cases}$$

(f: estado actual, símbolo que lee → estado al que transitará)

(Para cada transición.)

(En cada transición emite un símbolo a la salida).

En un instante determinado, la máquina secuencial sólo puede estar en un estado, recibir un símbolo del alfabeto de entrada y generar un símbolo del alfabeto de salida.

En una máquina de Mealy, la salida que genera depende tanto del estado en el que se encuentra como del símbolo de entrada que lee.

- **MÁQUINA DE MOORE:**

$$\begin{cases} f: Q \times \Sigma_E \to Q \\ g: Q \to \Sigma_S \end{cases}$$

Para cada transición.

En una máquina de Moore cada estado tiene asociado un símbolo del alfabeto de salida. Por tanto, la salida no depende del símbolo de entrada, sólo del estado en el que se encuentre.

Ej: Máquina secuencial que emite un 1 si el nº de unos leídos es par y 0 si es impar.

DETECTOR DE PARIDAD

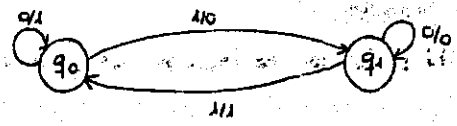
$\Sigma_E = \{0, 1\}$ Recibe cadenas binarias: $w \in \Sigma_E^*$

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

$Q = \{q_0, q_1\}$ → Nº de estados = 2 ⇒ Necesita recordar si $\begin{cases} \text{- El nº de 1s leídos es par: } q_0 \\ \text{- El nº de 1s leídos es impar: } q_1 \end{cases}$

⇒

MEALY



(DETECTOR DE PARIDAD)

La salida está asociada a cada par (q, e)

TABLA DE TRANSICIÓN:

ENTRADAS / ESTADOS	e_1	e_2	e_3	e_i	e_n
q_0					
q_1					
q_2					
q_j			1	q_k/s_i	
q_p					

$t(q_j, e_i) = q_k$
$g(q_j, e_i) = s_i$

f (ESTADOS) / (ENTRADAS)	0	1
q_0	q_0	q_1
q_1	q_1	q_0

$f(q_0, 0) = q_0$ $\quad f(q_1, 0) = q_1$
$f(q_0, 1) = q_1$ $\quad f(q_1, 1) = q_0$

TABLA DE SALIDA:

g	0	1
q_0	1	0
q_1	0	1

$g(q_0, 0) = 1$ $\quad g(q_1, 0) = 0$
$g(q_0, 1) = 0$ $\quad g(q_1, 1) = 1$

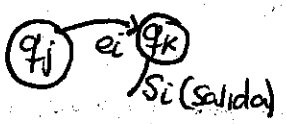
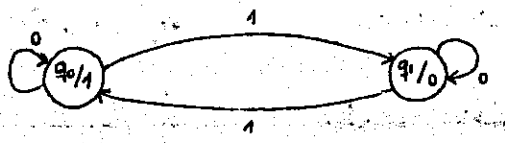


TABLA ÚNICA DE TRANSICIÓN Y SALIDA.

f/g	0	1
q_0	$q_0/1$	$q_1/0$
q_1	$q_1/0$	$q_0/1$

MOORE



La salida está asociada a cada estado.
(DETECTOR DE PARIDAD).

ENT / EST	e_1 e_2 e_i e_n	$g(q_i)$
q_0		$g(q_0)$
q_1		$g(q_1)$
q_2		$g(q_2)$
q_j		$g(q_j)$
q_p		$g(q_p)$

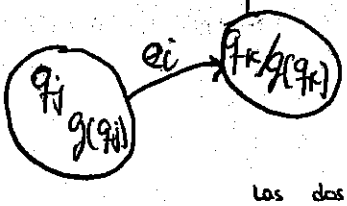


TABLA DE TRANSICIÓN → Es la misma que si es Mealy.

TABLA DE SALIDA →

Q	Σ_S
q_0	1
q_1	0

TABLA ÚNICA →

Q/Σ_S	0	1
$q_0/1$	q_0	q_1
$q_1/0$	q_1	q_0

(ENTRADAS)

Las dos máquinas representadas por estos diagramas de transición son equivalentes.

- EXTENSIÓN DE f Y g A PALABRAS

¿Cómo funcionan las MS (máquinas secuenciales) al procesar cadenas $w \in \Sigma_E^*$?

f estaba definida para símbolos de Σ_E.

Ahora definiremos f' a partir de f:

MEALY

$$f' : Q \times \Sigma_E^* \to Q$$
$$g' : Q \times \Sigma_E^* \to \Sigma_S^*$$

g': va emitiendo un símbolo por cada transición

$$\begin{cases} f'(q, \lambda) = q \quad \forall q \in Q \\ f'(q, ax) = f'(f(q,a), x) \quad a \in \Sigma_E \quad x \in \Sigma_E^* \\ g'(q, \lambda) = \lambda \quad \forall q \in Q \\ g'(q, ax) = (g(q,a)) \cdot g'(f(q,a), x) \quad \forall q \in Q \\ \qquad\qquad \forall a \in \Sigma_E, \forall x \in \Sigma_E^* \end{cases}$$

MOORE

$$f' : Q \times \Sigma_E^* \to Q$$
$$g' : Q \times \Sigma_E^* \to \Sigma_S^*$$

$$\begin{cases} f' \Rightarrow \text{Igual que la de Mealy} \\ g'(q, \lambda) = \lambda \quad \forall q \in Q \\ g'(q, ax) = g(f(q,a)) \cdot g'(f(q,a), x) \end{cases}$$

g solo tiene en cuenta el estado, y no una salida.

Emite el primer símbolo y luego ya habrá transitado a $f(q,a)$.

La longitud de la palabra de entrada siempre es igual que la longitud de la cadena de salida:

$$|g'(q, ax)| = 1 + |x| \quad \text{(para Mealy y Moore)}$$

Si $x = \lambda \rightarrow f' = f$ $(ax = a\lambda = a)$

λ transiciones: $\lambda \Rightarrow$ No recibe ninguna entrada.

Si una máquina se encuentra en un estado q y no recibe nada (λ) permanece en q. Tiene un comportamiento estable.

Existen dos tipos de máquinas.

- MÁQUINA **DETERMINISTA** — Desde cada situación solo puede hacer una siguiente operación.

- MÁQUINA **NO DETERMINISTA** — Tiene más de una posible operación y puede transitar a cualquiera de ellas.

(3/11/05)

3. EQUIVALENCIA MS. MEALY ↔ MS. MOORE.

Toda Máquina de Mealy se puede transformar en una equivalente de Moore y viceversa.

Al pasar de Mealy a Moore, el número de estados generalmente aumenta.

• MEALY → MOORE:

Sea $M = (\Sigma_E, \Sigma_S, Q, f, g)$ una MS Mealy.

$\exists\ M^* = (\Sigma_E, \Sigma_S, Q^*, f^*, g^*)$ MS. Moore tal que:

$\forall q \in Q,\ \exists q' \in Q^*,\ g'(q, x) = g'(q', x)\ \forall x \in \Sigma_E^*$

↳ Dos máquinas son equivalentes cuando tienen la misma función respuesta para cualquier entrada que se les presente.

$Q^* = Q \times \Sigma_S$ → Los estados de la máquina Moore serán los elementos que resulten del producto cartesiano $(Q \times \Sigma_S)$.

$f^*: (Q \times \Sigma_S) \times \Sigma_E \to (Q \times \Sigma_S)$ (primero conjunto de entrada de la mealy)

$f^*((q,s), a) = (f(q,a), g(q,a))$ $\quad \forall q \in Q\ \forall s \in \Sigma_S\ \forall a \in \Sigma_E$

$g^*: Q \times \Sigma_S \to \Sigma_S$ $\quad q \to ¿q'?$ $\quad q' = (q,s)$

$g^*(q,s) = s$

• MOORE → MEALY:

(Mirar las fotocopias del tema).

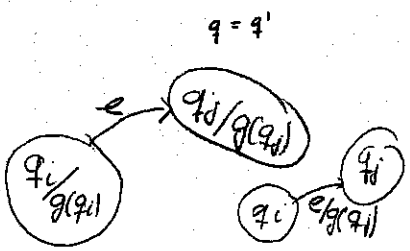
Sea $M = (\Sigma_E, \Sigma_S, Q, f, g)$ una MS. Moore.

$\exists\ M^* = (\Sigma_E, \Sigma_S, Q^*, f^*, g^*)$ una MS. Mealy tal que.

$\forall q \in Q\ \ \exists q' \in Q^*\ .\ g'(q, x) = g'(q', x)\ \ \forall x \in \Sigma_E^*$

$Q^* = Q$ $\quad g^*: Q \times \Sigma_E \to \Sigma_S$

$f^* = f$ $\quad g^*(q, a) = g(f(q,a))$



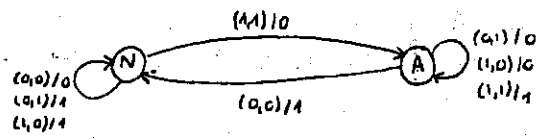
EJ. Construir una máquina secuencial que sea un SUMADOR BINARIO.

Suma x e y / $x, y \in \{0,1\}$ Posibles entradas: (0,0) (0,1) (1,0) (1,1)

Estados => Z : Que lleve o no acarreo: (N) Sin acarreo.
(A) Con acarreo.

• MEALY

	(0,0)	(0,1)	(1,0)	(1,1)
N	N/0	N/1	N/1	A/0
A	N/1	A/0	A/0	A/1



0+0 = 0 1+0 = 1
0+1 = 1 1+1 = 0 + ac.

Partiendo de esta MS Mealy vamos a obtener una MS Moore equivalente.

• MOORE

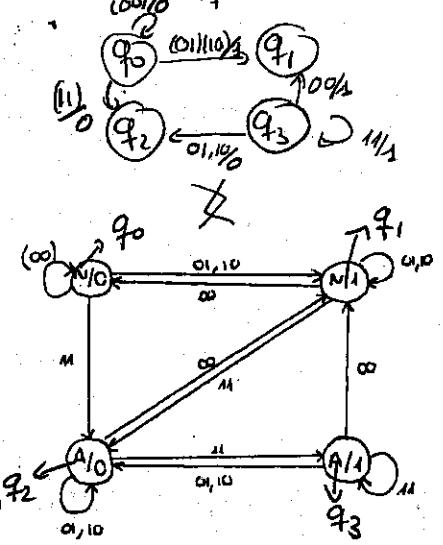
$Q = \{N, A\}$

Posibles entradas = $\{00, 01, 10, 11\}$

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

$Q^* = (Q \times \Sigma_S) = (N/0), (N/1), (A/0), (A/1)$

$\delta^* = (Q \times \Sigma_S) \times \Sigma_E \rightarrow (Q \times \Sigma_S)$



$\delta^*((N/0) \times (00)) = (N/0)$ $\quad \delta^*((N/1) \times (00)) = (N/0)$
$\delta^*((N/0) \times (01)) = (N/1)$ $\quad \delta^*((N/1) \times (01)) = (N/1)$
$\delta^*((N/0) \times (10)) = (N/1)$ $\quad \delta^*((N/1) \times (10)) = (N/1)$
$\delta^*((N/0) \times (11)) = (A/0)$ $\quad \delta^*((N/1) \times (11)) = (A/0)$

$\delta^*(A/0 \times (00)) = N/1$ $\quad \delta^*(A/1 \times (00)) = N/1$
$\delta^*(A/0 \times (01)) = A/0$ $\quad \delta^*(A/1 \times (01)) = A/0$
$\delta^*(A/0 \times (10)) = A/0$ $\quad \delta^*(A/1 \times (10)) = A/0$
$\delta^*(A/0 \times (11)) = A/1$ $\quad \delta^*(A/1 \times (11)) = A/1$

	(0,0)	(0,1)	(1,0)	(1,1)	$g(q)$
q_0	q_0	q_1	q_1	q_2	0
q_1	q_0	q_1	q_1	q_2	1
q_2	q_1	q_2	q_2	q_3	0
q_3	q_1	q_2	q_2	q_3	1

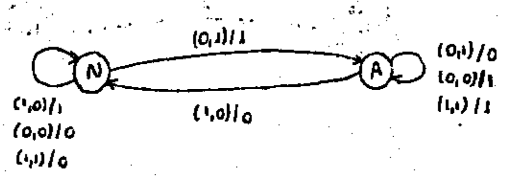
$g^* = Q \times \Sigma_S \rightarrow \Sigma_S$

$g^*(N,0) = 0$ $\quad g^*(A,0) = 0$
$g^*(N,1) = 1$ $\quad g^*(A,1) = 1$

N/0 q_0
N/1 q_1
A/0 q_2
A/1 q_3

EJ. RESTADOR BINARIO.

MEALY



0-0 = 0 1-0 = 1

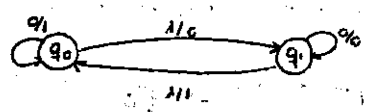
0-1 = 1 + ac 1-1 = 0

	00	01	10	11
(N 0)	N/0	A/1	N/1	N/0
(A 1)	A/1	A/0	N/0	A/1

Calcular el equivalente de Moore.

EJ. DETECTOR DE PARIDAD.

MEALY

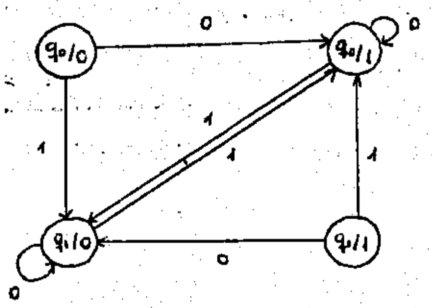


$\Sigma_E = \Sigma_S = \{0,1\}$ $q_0 \Rightarrow$ El nº de 1's leídos es par: emite un 1.

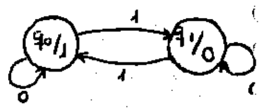
$Q = \{q_0, q_1\}$ $q_1 \Rightarrow$ " " " impar: " " 0.

$Q^* = \{q_0/_0,\ q_0/_1\ \ q_1/_0\ \ q_1/_1\}$

MOORE



Se puede simplificar →



Demostrar:

$h(qxy) = h(qx)\,h(f(qx)y) \quad \forall xy \in \Sigma_e^* \quad q \in Q$

$|x| = 0 \quad x = \lambda \quad h(qxy) = h(q\lambda y) = h(q,y) = h(q\lambda)\,h(f(q\lambda)y) = \lambda h(qy) = h(qy)$

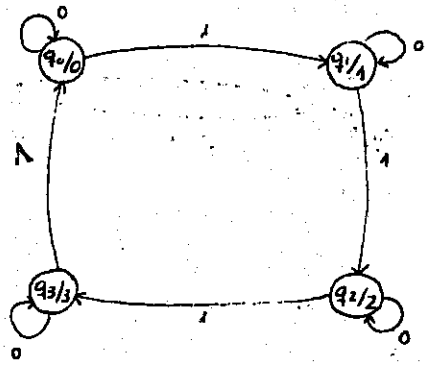
$|x| = k+1 \quad x = az \quad |z| = k$ cumple

$h(qxy) = h(qazy) \underset{Def}{=} h(qa)\,h(f(qa)zy) \underset{Ind}{=} h(qa)\,h(f(qa)z)\,h(f(f(qa)z)y) =$

$\underset{Def}{=} h(qaz)\,h[f(qaz)y] = h(qx)\,h(f(q,x)y)$

EJ $\Sigma_E = \{0,1\}$ Entrada $= w \in \Sigma_E^*$

MOORE



Último símbolo que emite en la salida:

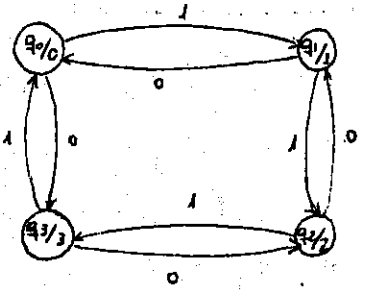
$N_1(w) \bmod 4$

↓ ↓

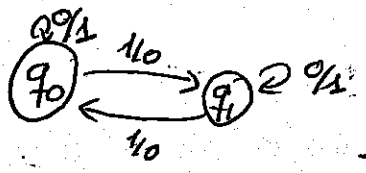
nº de 1s módulo.

EJ: Último símbolo de la salida $= (N_1(w) - N_0(w)) \bmod 4$

MOORE



EJ



	0	1
q_0	$q_0/1$	$q_1/0$
q_1	$q_1/1$	$q_0/0$

Estando en q_0 y con entrada 0 nos quedamos en q_0 y con salida 1

$X = 011101$

$f'(q_0\,011101) = f'(f(q_0)11101) = f'(q_0 11101) = f'(f(q_0)1101) =$

$= f'(q_1 1101) = f'(f(q_1)101) = f'(q_0 101) = f'(q_1 01) = f'(q_1 1) = q_0$

$g'(q_0\,011101) = g(q_0 0)\, g'(f(q_0 0)11101) = 1 g'(q_0 1110) =$

$= 1 g(q_0 1)\, g'(f(q_0 1)1101) = 10 g'(q_1 1101) = 10 g(q_1 1) g'(f(q_1 1)101) =$

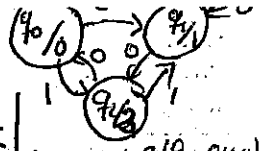
$= 100 g'(q_0 101) = 100 g(q_0 1) g'(f(q_0 1)01) = 1000 g'(q_1 01) =$

$= 1000 g(q_1 0) g'(f(q_1 0)1) = 10001 g'(q_1 1) = 100010$

$10001 g(q_1 1) g'(f(q_1 1)\lambda) =$

$= 100010$

0/0
10/1
01/



$f(q_0\ 01110) = f(f(q_0,0)1110) = f(q_0\ 1110) = f(\lambda(q_0,1)110) = f(q_2 110) =$

$= f(\delta(q_2,1)10) = f(q_1, 10) = f(f(q_1,1)0) = f(q_2 0) = q_0$

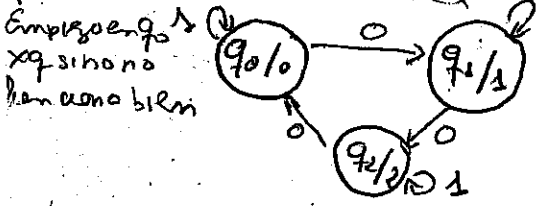
$g(q_0\ 01110) = g(f(q_0,0)) = g(f(q_0)1110) = g(q_1\ 1110) = 1\,g(q_1,1)110) = 1\,\delta(q_2)\delta(q_2 110) =$

$= 12\,g(f(q_2,1))\,g'(f(q_2,1)\,10) = 12\,g(q_1)\,g'(q_1 10) = 121(f(q_1,1))g'(q_1,0)$

$= 1212\,g(q_2)\,g'(q_2 0) = 1212\,g(f(q_2,0))\,g'(f(q_2,0),\lambda) = 12120$

$x = 01110$

Construir una maquina secuencial en el alfabeto de entrada $\{0,1\}$ y cuyas salidas sean el nº de ceros módulo 3 de la palabra de entrada



Moore

Vamos a ver si es correcto.

Lo comprobamos con la palabra 01100010

$f'(q_0\ 01100010) = f'(f(q_0,0)1100010) =$

$f'(q_1 1100010) = f'(f(q_1,1)100010) = f'(q_1 100010) =$

$f'(f(q_1,1)00010) = f'(q_1 00010) = f'(f'(q_1,0)0010) =$

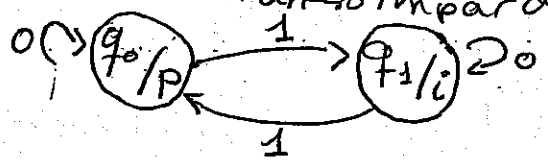
$f'(q_2 0010) = f'(f(q_2,0)010) = f'(q_0 010) =$

$= f'(f(q_0,0)10) = f'(q_1 10) = f'(f(q_1,1)0) =$

$= f'(q_1 0) = q_0$

$g(q_0\ 01100010) = 11120112$

Maquina secuencial que genera como salida una p si hasta ese instante ha recibido un nº par de unos en la entrada, y que genera una i en la salida si hasta ese instante ha recibido un número impar de unos



Moore

DEPARTAMENTO DE INTELIGENCIA ARTIFICIAL

FACULTAD DE INFORMÁTICA
UNIVERSIDAD POLITÉCNICA DE MADRID

INFORMATICA TEÓRICA

Curso 2004-2005

Prácticas Tema 3

MÁQUINAS SECUENCIALES

Práctica 3.1: Diseño de máquinas secuenciales.

1. Diseñar una máquina expendedora que distribuya dos tipos de bebidas en lata: coca-cola (c) y cerveza (cv). El precio por unidad es 100 pts. La máquina acepta monedas de 25, 100 y 200 ptas. y devuelve el cambio necesario. La máquina posee 2 botones para expulsar la bebida : B (blanco, para que salga la cerveza) y N (negro, para que salga la coca-cola).

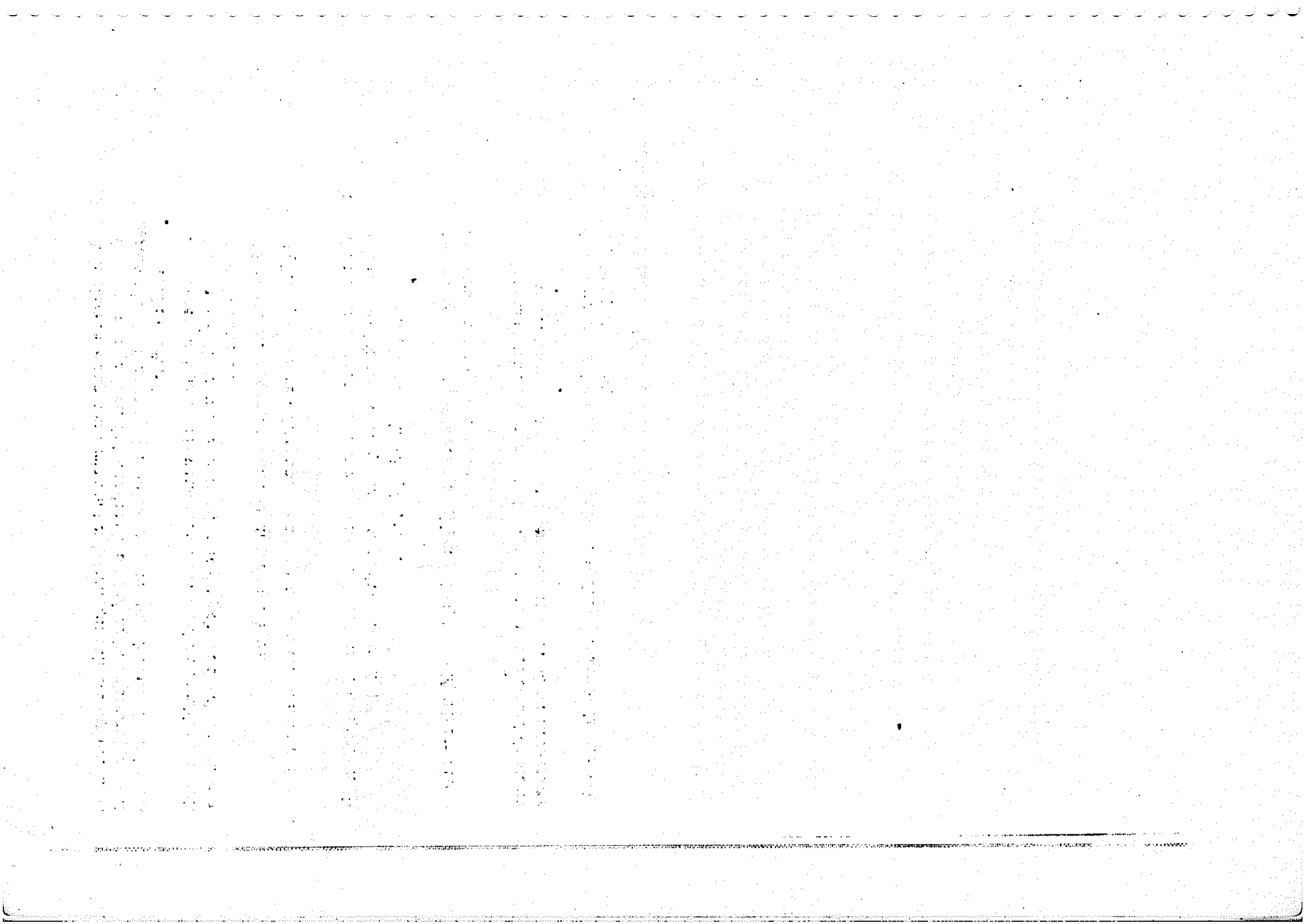
   (Nota: La máquina devuelve el cambio cuando la cantidad introducida excede de 100 ptas., antes de pulsar un botón).

2. Dados los alfabetos $\Sigma_E = \{a,b\}$ de entrada y $\Sigma_S = \{X,Y,Z\}$ de salida. Se pide:

   a) Construir una máquina secuencial de Mealy que genere la salida:

   - **X** si la entrada termina en **aba**,
   - **Y** si la entrada termina en **aab** y
   - **Z** en cualquier otro caso.

   b) Construir una máquina secuencial de Moore que realice la misma operación.



3. A un ciego se le asigna una caja con tres botones que pulsados suponen inflar, desinflar y cambiar de estado (pasar de inflado a desinflado, o viceversa) un globo conectado con dicha caja y que el ciego no puede ver. Cada uno de los botones sólo realiza una función.

   Al no poder ver el globo, se le suministra al ciego una alarma, también conectada a la caja, que sólo suena en el instante de paso del globo de desinflado a inflado, apagándose después.

   a) Representar este sistema mediante una máquina secuencial.

   b) ¿Cómo podría el ciego determinar el estado del globo en cualquier momento?

4. a) Construir, a ser posible, una máquina secuencial de Moore sobre el alfabeto $\Sigma_E = \{a,b,c\}$ de tal forma que la respuesta corresponda a la operación:

   $$r = (N_a(x) + 2\,N_b(x) - 3\,N_c(x)) \bmod 5.$$

   b) Supuesto que la respuesta fuese: $r = (N_a(x) + 2\,N_b(x) - 3\,N_c(x))$ construir, a ser posible, la correspondiente máquina.

5. Diseñar una máquina secuencial de Mealy y otra máquina secuencial de Moore que resten dos números expresados en sistema binario, siendo el minuendo mayor que el sustraendo.

   Comprobar el funcionamiento de ambas máquinas efectuando la operación 101100101 - 10110110.

Práctica 3.2: Relaciones entre los modelos Mealy y Moore

1. Conversión Moore-Mealy

Teorema: Sea $M = (\Sigma_E, \Sigma_S, Q, f, g)$ máquina secuencial de Moore, existe $M' = (\Sigma_E, \Sigma_S, Q', f', g')$ máquina secuencial de Mealy tal que

$$\forall\, q \in Q \quad \exists\, q' \in Q' \text{ que cumple } h(q, x) = h'(q', x) \quad \forall\, x \in \Sigma_E^*$$

Construimos la máquina M' de la siguiente forma:

$Q' = Q$

$f' = f$

$g' : Q \times \Sigma_E \rightarrow \Sigma_S$ definida por $g'(q, e) = \hat{g}(f(q, e))$

a) Construir la MS de Mealy equivalente a la MS de Moore obtenida en la práctica 3.1.4.

b) Demostrar la equivalencia de ambas MS por inducción sobre la longitud de x.

2. Conversión Mealy-Moore

Teorema: Sea $M = (\Sigma_E, \Sigma_S, Q, f, g)$ máquina secuencial de Mealy, existe $M' = (\Sigma_E, \Sigma_S, Q', f', g')$ máquina secuencial de Moore tal que

$$\forall\, q \in Q \quad \exists\, q' \in Q' \text{ que cumple } h(q, x) = h'(q', x) \quad \forall\, x \in \Sigma_E^*$$

Construimos la máquina M' de la siguiente forma:

$Q' = Q \times \Sigma_S$

$f': (Q \times \Sigma_S) \times \Sigma_E \rightarrow Q \times \Sigma_S \qquad f'((q, s), e) = (f(q, e), g(q, e))$

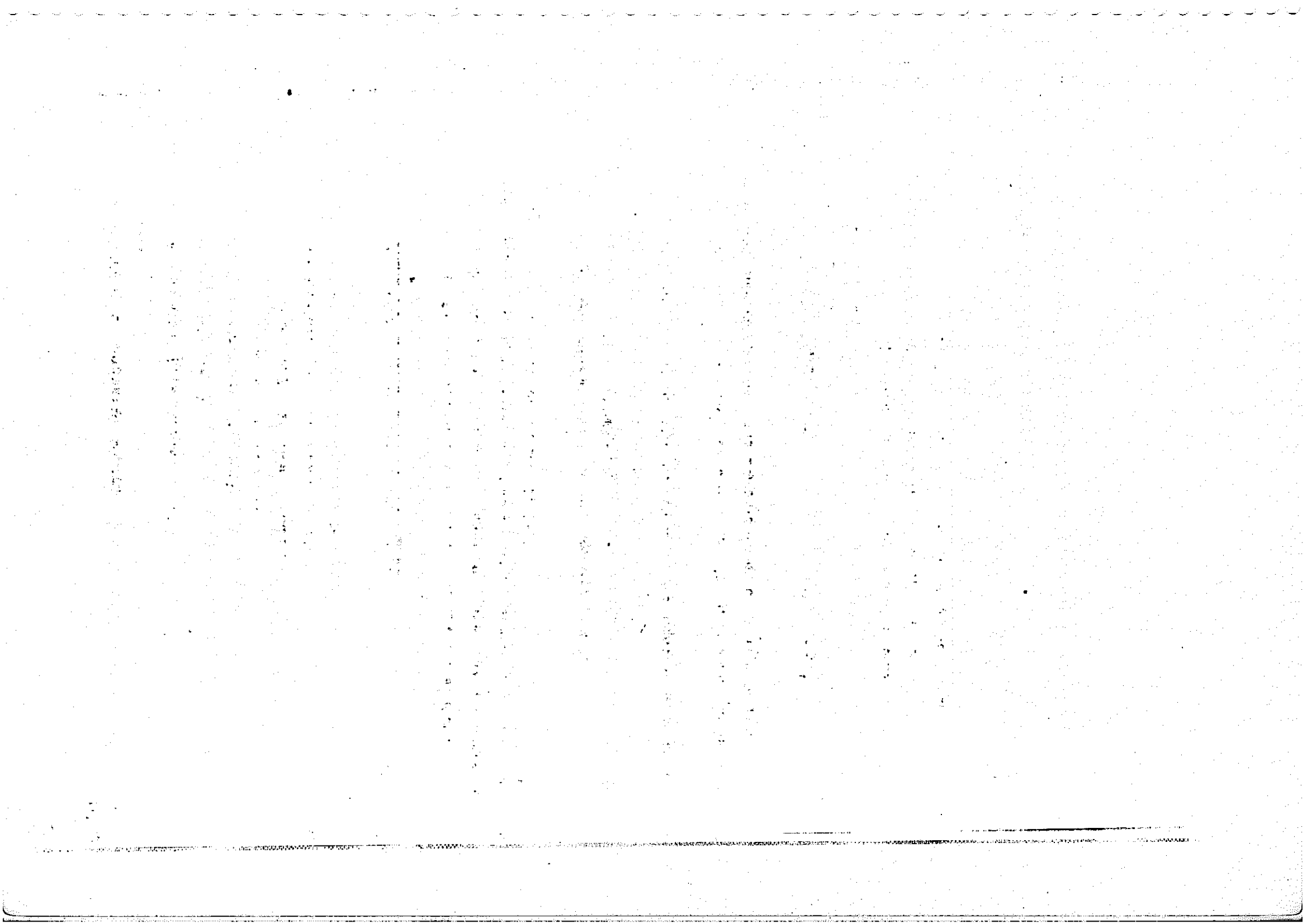
$g': Q \times \Sigma_S \rightarrow \Sigma_S \qquad g'((q, s)) = s$

a) Dada la MS de Mealy "sumador binario":

	00	01	10	11
N	N/0	N/1	N/1	A/0
A	N/1	A/0	A/0	A/1

Construir la MS de Moore equivalente.

b) Demostrar la equivalencia de ambas MS por inducción sobre la longitud de x.



(8/11/05)

TEMA 4 AUTÓMATAS FINITOS.

1.- Autómatas finitos deterministas (AFD)
2.- Minimización de AFD.
   - Equivalencia de estados.
   - Equivalencia de autómatas.
   - Construcción del mínimo equivalente
3.- Autómatas finitos no deterministas (AFND)

1. AUTÓMATAS FINITOS DETERMINISTAS (AFD).

Un AFD puede considerarse como una máquina secuencial de Moore.

En cada instante la cabeza de lectura va leyendo símbolo a símbolo la cadena $w \in \Sigma^*$ al mismo tiempo que transita entre sus estados. Sólo produce un tipo de salida: aceptación o no de la palabra ($\Sigma_s = \{0,1\}$).

LOS AFD ACEPTAN SÓLO LENGUAJES REGULARES (TIPO 3).

$L(AF) = L.R.$

L(A) está formado por todas las cadenas w que acepta el autómata A.

El conjunto de lenguajes que aceptan los AFD es el mismo que el conjunto de lenguajes aceptado por los AFND ⟶ Conjunto de los Lenguajes Regulares.

Para cada lenguaje regular existe un AF que lo acepta.

- 3 definiciones de lenguaje regular:
   1) Lenguaje generado por una Gramática Lineal.
   2) " " que puede representarse por una ER.
   3) " " para el existe un AF que lo acepta.

DEFINICIÓN: Sea A un AFD.

A se define como la quintupla:

$$A = (\Sigma, Q, q_0, \delta, F)$$

Σ ≡ Alfabeto de entrada.

Q ≡ Conjunto finito de estados.

q_0 ≡ Estado inicial : $q_0 \in Q$.

δ ≡ Función de transición

F ≡ Conjunto de estados finales o estados de aceptación : $F \subseteq Q$.

$$\begin{cases} \delta : Q \times \Sigma \to Q \\ \delta(q, a) \to p \quad q, p \in Q, a \in \Sigma \end{cases}$$ (Si se encuentra en el estado q y lee a, transita a p).

extensión a palabras

$$\begin{cases} \delta^* : Q \times \Sigma^* \to Q \\ \delta^*(q, \lambda) \to q \quad \forall q \in Q \\ \delta^*(q, ax) \to \delta^*(\delta(q,a), x) \quad a \in \Sigma, x \in \Sigma^*, q \in Q \end{cases}$$

(A partir de ahora usaremos siempre δ. El contexto nos dirá si es δ o δ^*).

¿Cuándo una palabra w es aceptada por un autómata?

↳ ¿$w \in L(A)$?

Una palabra w es aceptada por un autómata cuando al leer la cadena w, el autómata, empezando en el estado inicial q_0, acaba sus transiciones en alguno de sus estados finales.

↳ $L(A) = \{ x \in \Sigma^* / \delta^*(q_0, x) \in F \}$

Una palabra será rechazada cuando:

$$x \in \Sigma^* / \delta(q_0, x) \in Q - F$$

λ ≡ No recibe ninguna entrada.

COMPORTAMIENTO DETERMINISTA → $\delta(q, \lambda) = q \quad \forall q \in Q$

COMPORTAMIENTO NO DETERMINISTA → $\delta(q, \lambda) = q' \quad \forall q, q' \in Q$

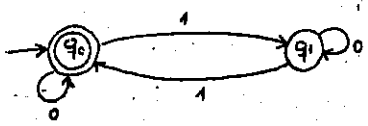
- $F = \emptyset \rightarrow L(A) = \emptyset$ (No hay estados finales).
- $F = Q \rightarrow L(A) = \Sigma^*$ (Todos los estados son finales).
- $\lambda \in L(A) \Leftrightarrow q_0 \in F$ (Aceptará la palabra λ si y sólo si el estado inicial es final).

- REPRESENTACIÓN DE AUTÓMATAS:

Estado intermedio — Estado inicial — Estado final

EJ: AFD que acepta cadenas binarias cuyo nº de unos es par.

$L(A) = \{x \in \{0,1\}^* / N_1(x) \text{ par}\}$



Es muy similar a la M.S. Moore.
2 estados. q_0 => nº de 1's leídos par.
q_1 => " " " impar.

LOS ESTADOS PERMITEN MEMORIZAR LA INFORMACIÓN QUE NECESITAMOS.

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

$Q = \{q_0, q_1\}$

$F = \{q_0\}$

$\lambda \in L(A)$ porque $q_0 \in F$

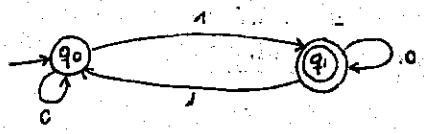
Tabla de transición:

ESTADOS	δ	0	1
→	$*q_0$	q_0	q_1
	q_1	q_1	q_0

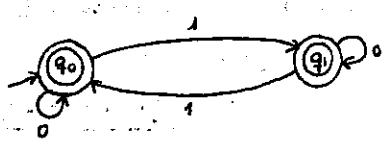
$\delta(q_0, 0) = q_0$
$\delta(q_0, 1) = q_1$
$\delta(q_1, 0) = q_1$
$\delta(q_1, 1) = q_0$

El estado inicial se marca con una →
Cada estado final se precede del símbolo *

EJ: El mismo ejemplo, pero ahora el estado final es q_1.
Ahora el lenguaje aceptado será:
$L(A_1) = \{x \in \{0,1\}^* / N_1(x) \text{ impar}\}$ $\quad \lambda \notin L(A_1)$



EJ: ¿Y si tanto q_0 como q_1 son finales?

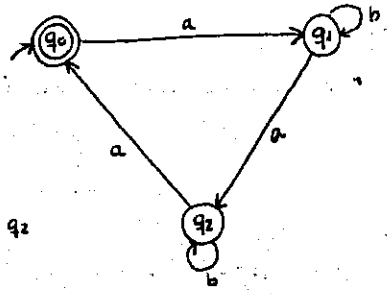


$L(A) \cup L(A_1) = (0,1)^* = \Sigma^*$

↳ Unión de las cadenas con nº de 1s P y las cadenas con nº de 1s impa

EJ: $L(A) = \{x \in \{a,b\}^* / N_a(x)$ múltiplo de 3 y un nº cualquiera de b$\}$

$\lambda \in L(A)$



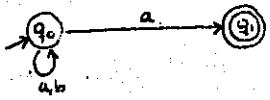
Si queremos que también las acepte si $N_a(x)$ es múltiplo de 3+1, hago final es estado q_1 también.

Si fuesen finales q_0, q_1 y q_2 $L(A)$ sería Σ^*.

Si sólo debe aceptarlas si $N_a(x)$ es múlt de 3+1, el único estado final sería q_1.

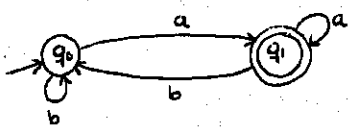
-CARACTERÍSTICA DE UN AFD.- $\forall (q, \Sigma_c)$ hay exactamente un único estado siguiente posible.

EJ: $L(A) = \{x \in \{a,b\}^* / x$ acaba en a$\} = (a+b)^*a$



$\delta(q_0, a) = \{q_0, q_1\}$ → NC DETERMINISTA

$\delta(q_1, a) = \delta(q_1, b) = \emptyset$ (No salen flechas de q_1)



$\delta(q_0, a) = q_1$

$\delta(q_0, b) = q_0$ DETERMINISTA.

$\delta(q_1, a) = q_1$

$\delta(q_1, b) = q_0$

Si q_0 fuese estado final, todos serían finales ($Q = F$) y entonces $L(A) = \Sigma^*$

(10/11/05)

- **ESTADOS ACCESIBLES**: Sea $A = (\Sigma, Q, q_0, \delta, F)$ un AFD.

Se dice que un estado $p \in Q$ es accesible desde otro estado $q \in Q$ si existe una cadena $x \in \Sigma^*$ tal que:

$$\delta(q, x) = p$$

Todo estado es accesible desde sí mismo: $\delta(q, \lambda) = q \quad \forall q \in Q$

→ Si $|Q| = n$ (hay n estados):

p accesible desde $q \iff \exists x \in \Sigma^*, |x| < n \mid \delta(q, x) = p$

• **DEM**. -

p accesible desde $q \longrightarrow \exists y \in \Sigma^* \quad \delta(q, y) = p$

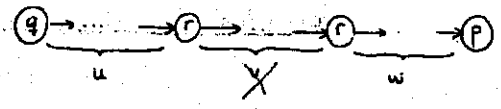
- $|y| < n$: Demostrado
- $|y| \geq n \longrightarrow$ El autómata transita por lo menos por $n+1$ estados.

Como $|Q| = n$, entonces habrá al menos 1 estado por el que transita más de una vez.

$y = uvw$

$|y| \geq n$

r el estado que se repite



$y_1 = uv \quad ; \quad (q, y_1) = p$

$|y_1| < |y| \longrightarrow$ 2 casos.

- $|y_1| < n$ Demostrado
- $|y_1| \geq n \longrightarrow$ Repetir

$|y_2| < |y_1| \longrightarrow$ 2 casos:

- $|y_2| < n$ Demostrado
- $|y_2| \geq n \longrightarrow$ Repetir hasta conseguir $|y_i| < n$

- **AFD CONEXO**: Un autómata A es conexo cuando todos sus estados son accesibles desde el estado inicial.

Los estados inaccesibles se pueden eliminar (así se obtiene el autómata conexo equivalente).

2. MINIMIZACIÓN DE AFD.

(Mirar las fotocopias del tema).

Sea A un AFD.

Existe $\hat{A}$ (AFD equivalente a A) con un número mínimo de estados. $\hat{A}$ es el **AUTÓMATA MÍNIMO** y es único salvo isomorfismo.

a) EQUIVALENCIA ENTRE ESTADOS.

* Sean p y q dos estados cualesquiera de un AFD A: $p, q \in Q$

$$p \, E \, q \equiv \forall x \in \Sigma^* \cdot \; \delta(p, x) \in F \iff \delta(q, x) \in F$$

(relación de equivalencia)

↳ Los estados p y q son **equivalentes** cuando todas las cadenas x que leídas desde el estado p hacen que se acceda a un estado final, hacen también que se llegue a un estado final si son leídas desde q (Y viceversa).

De otra forma:

$$\forall x \in \Sigma^* \quad \begin{cases} \delta(p, x) \in F \iff \delta(q, x) \in F \\ \delta(p, x) \notin F \iff \delta(q, x) \notin F \end{cases}$$

- Simétrica → Si pEq, entonces qEp ...
- Reflexiva → $pEp \; \forall p \in Q$ (Todo estado es equivalente consigo mismo.
- Transitiva → pEq & qEq', entonces pEq'.

Por lo tanto, la **equivalencia de estados (E)** es una relación de equivalencia sobre el conjunto Q y determinarán una partición de Q.

$$Q/E = P_E$$

↑ PARTICIÓN DE EQUIVALENCIA.

Dentro de una partición de equivalencia hay clases de equivalencia, y dentro de éstas se agrupan los estados equivalentes.

Como la condición $\delta(p,x) \in F \Leftrightarrow \delta(q,x) \in F$ debe cumplirse $\forall x$ y hay ∞ cadenas $x \in \Sigma^*$, no hay ningún algoritmo para calcular P_E porque no terminaría nunca.

* Sean $p, q \in Q$, $k \in \mathbb{N}$.

Se dice que p y q son <u>k-equivalentes</u> o equivalentes en <u>longitud k</u> si:

$$\left[p E_k q \equiv \forall x \in \Sigma^* \; |x| \leq k \quad \delta(p,x) \in F \Leftrightarrow \delta(q,x) \in F \right]$$

La k-equivalencia es también <u>simétrica, reflexiva y transitiva</u>. Por lo tanto, es una relación de equivalencia sobre el conjunto Q y determinará una partición de Q (el conjunto cociente):

$$\boxed{Q/E_k = P_k}$$

↑ PARTICIÓN DE K-EQUIVALENCIA.

$k = 0$ $\quad p E_0 q \equiv \forall x \in \Sigma^* \; |x| \leq 0 \; (\Leftrightarrow x = \lambda) \; \delta(p,\lambda) \in F \Leftrightarrow \delta(q,\lambda) \in F \longrightarrow p \in F \Leftrightarrow q \in F$

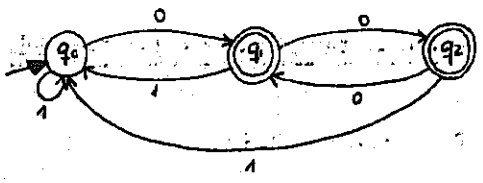
$\boxed{P_0 = \{F, Q-F\}}$ ⟶ Dos estados son <u>0-equivalentes</u> si los <u>dos son finales</u> o si los <u>dos son no finales</u>.

En P_0 hay dos clases de equivalencia: los estados finales y los no finales.

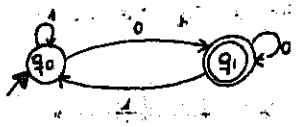
Se cumple:

$$\begin{cases} p E q \Rightarrow p E_k q \quad \forall k \\ p E_k q \Rightarrow p E_r q \quad \forall r \leq k \end{cases}$$

Ej. A_1



A_2



$L(A_1) = L(A_2) = \{x \in \{0,1\}^* / \ x \text{ termina en } 01\}$

• **PROPIEDADES**

1.- $pEq \Rightarrow \delta(p,x) E \delta(q,x) \quad \forall x \in \Sigma^*$ → Si p y q son equivalentes, se verifica que los estados siguientes a los que acceden p y q al leer cualquier x son también equivalentes.

2.- $pE_k q \not\Rightarrow \delta(p,x) E_k \delta(q,x) \quad \forall x \in \Sigma^*$ → Que dos estados sean k-equivalentes no implica que los dos siguientes lo sean

→ 3.- $pE_{k+1}q \iff pE_k q \wedge \delta(p,e) E_k \delta(q,e) \quad \forall e \in \Sigma$ → Dos estados son k+1 equiv. si son k-equivalentes y sus estados siguientes $\forall x \in \Sigma^*$ son también k-equivalentes.

$p P_0 q$
↓ Si $pE_0 q$ ∧ sus estados siguientes son también E_0.
$p P_1 q$ (S
↳ Si no se verifica, p y q pasarán a clases distint

$P_0 \rightsquigarrow P_1 \rightsquigarrow P_2 \ldots$

4.- $P_k = P_{k+1} \Rightarrow P_k = P_{k+i} \quad \forall i \geq 0$

5.- $\boxed{P_k = P_{k+1} \Rightarrow P_E = P_k}$ → ASÍ TENEMOS UN ALGORITMO PARA CALCULAR E.

6.- Sea $|Q| = n \quad \exists j \leq n-2 \ / \ P_j = P_{j+1}$ → La repetición existe siempre y produce en un número finito de casos.

- ALGORITMO PARA LA OBTENCIÓN DE LA PARTICIÓN DE EQUIVALENCIA P_E:

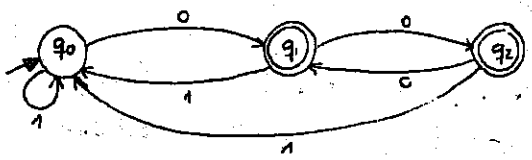
1) Obtener P_0 : $P_0 = \{F, Q-F\}$

2) Obtener P_{k+1} a partir de P_k (aplicando la propiedad 3)

3) Si $P_{k+1} = P_k \Longrightarrow P_E = P_k$ FIN

4) Si $P_{k+1} \neq P_k \Longrightarrow$ volver al paso 2.

EJ:



$P_0 = \{[q_0], [q_1, q_2]\}$

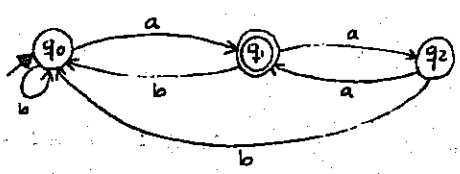
¿Son $q_1 E_1 q_2$? → $q_1 E_0 q_2$.

$$\begin{cases} e=0 & \begin{cases} \delta(q_1,0) = q_2 \\ \delta(q_2,0) = q_1 \end{cases} \Big\} q_1 E_0 q_2 \\ e=1 & \begin{cases} \delta(q_1,1) = q_0 \\ \delta(q_2,1) = q_0 \end{cases} \Big\} q_0 E_0 q_0 \end{cases}$$

$P_1 = \{[q_0], [q_1, q_2]\}$

$P_0 = P_1 = P_E$ //

EJ:



$P_0 = \{[q_0, q_2], [q_1]\}$

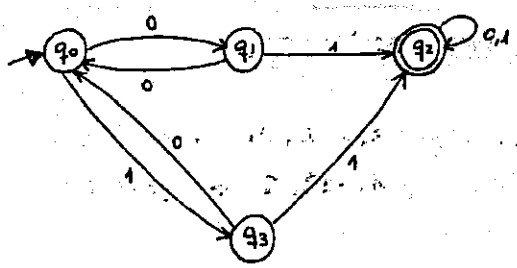
¿ $q_0 E_1 q_2$? $\quad e=a \quad \begin{cases} \delta(q_0,a) = q_1 \\ \delta(q_2,a) = q_1 \end{cases} \Big\} q_1 E_0 q_1$

$\qquad\qquad e=b \quad \begin{cases} \delta(q_0,b) = q_0 \\ \delta(q_2,b) = q_0 \end{cases} \Big\} q_0 E_0 q_0$

$P_1 = \{[q_0, q_2], [q_1]\}$

$P_0 = P_1 = P_E$ //

EJ:



$\{q_0 E_1 q_1\}$

$P_0 = \{[q_0, q_1, q_3], [q_2]\}$

$\delta(q_0, 0) = q_1$	$q_0 E_0 q_1$	$\delta(q_0, 1) = q_3$	$q_2 \not E_0 q_3$	
$\delta(q_1, 0) = q_0$		$\delta(q_1, 1) = q_2$	$q_2 E_0 q_2$	→ Sólo se cumple para los estados q_1 y q_3
$\delta(q_3, 0) = q_0$		$\delta(q_3, 1) = q_2$		

$\{q_1 E_1 q_3\}$

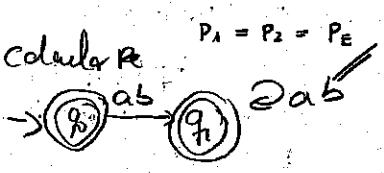
$P_1 = \{[q_0], [q_1, q_3], [q_2]\}$

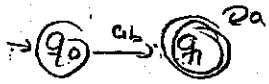
$\delta(q_1, 0) = q_0$	$q_0 E_1 q_0$	$\delta(q_1, 1) = q_2$	$q_2 E_1 q_2$
$\delta(q_3, 0) = q_0$		$\delta(q_3, 1) = q_2$	

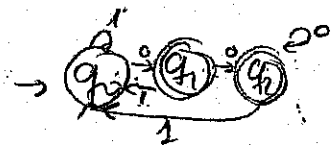
$P_2 = \{[q_0], [q_1, q_3], [q_2]\}$

$P_1 = P_2 = P_E$

- calcula R







ii) **AUTÓMATAS FINITOS EQUIVALENTES**

Sean dos AFD $\begin{cases} A_1 = (\Sigma, Q_1, q_{01}, f_1, F_1) \\ A_2 = (\Sigma, Q_2, q_{02}, f_2, F_2) \end{cases}$

a) $q_1 \in Q_1$ y $q_2 \in Q_2$ son equivalentes si:

$$\left[q_1 E q_2 \equiv \forall x \in \Sigma^* \quad f_1(q_1, x) \in F_1 \iff f_2(q_2, x) \in F_2 \right]$$

b) A_1 equivalente a A_2 si:

$$\left[A_1 E A_2 \equiv L(A_1) = L(A_2) \iff \forall x \in \Sigma^* \; f_1(q_{01}, x) \in F_1 \iff f_2(q_{02}, x) \in F_2 \right.$$
$$\left. \iff q_{01} E q_{02} \right.$$

c) El **AUTÓMATA SUMA** $A_1 \oplus A_2$ se define como la unión de la tabla de transición del primer autómata y la tabla de transición del segundo.

Sean A_1 y A_2 dos AFD tales que $Q_1 \cap Q_2 = \emptyset$

↳ No pueden tener estados en común.

$A_1 = (\Sigma_1, Q_1, q_{01}, f_1, F_1)$
$A_2 = (\Sigma_2, Q_2, q_{02}, f_2, F_2)$

$$\left[A_1 \oplus A_2 = (\Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2, q_0, f, F_1 \cup F_2) \right]$$ → Tendrá 2 partes no conexas

donde

$q_0 = q_{01}$ ó q_{02}

$$f: \begin{cases} f_1(q, e) \; \forall q \in Q_1, \; \forall e \in \Sigma_1 \\ f_2(q, e) \; \forall q \in Q_2, \; \forall e \in \Sigma_2 \end{cases}$$

La tabla de transición del autómata suma es el resultado de colocar juntas las dos tablas.

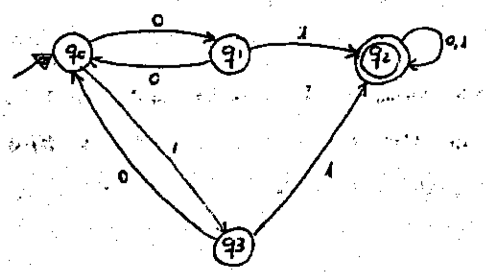
* Para saber si dos autómatas A_1 y A_2 son equivalentes, se construye la partición P_E (ir calculando P_k hasta que $P_k = P_{k+1}$) del autómata suma y se mira si los estados iniciales q_{01} y q_{02} están en la misma clase de equivalencia.

$$\underbrace{A_1 \oplus A_2 / E}_{P_E} \qquad \boxed{A_1 \, E \, A_2 \iff q_{01} \, E \, q_{02} \text{ en } A_1 \oplus A_2}$$

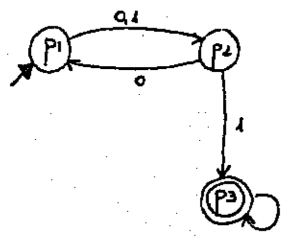
EN LA PRÁCTICA NO SE USA DEMASIADO ESTA FORMA. ES MÁS SENCILLO DETERMINAR EL LENGUAJE DE CADA UNO DE ELLOS Y COMPROBAR QUE $L(A_1) = L(A_2)$

EJ:

A_1



A_2



¿ $A_1 \, E \, A_2$? (Hacerlo mediante el autómata suma).

$A_1 \oplus A_2 / E \longrightarrow$ ¿ $q_0 \, E \, p_1$? $\qquad F = \{q_2, p_3\}$

$P_0 = \{[q_2, p_3], [q_0, q_1, q_3, p_1, p_2]\}$

$\delta(q_0, 0) =$ ___

$P_1 = \{[q_2, p_3], [q_1, q_3, p_2], [q_0, p_1]\}$

$q_0 \not{E}_1 q_3$ $\qquad P_2 = \{[q_2, p_3], [q_1, q_3, p_2], [q_0, p_1]\}$

$q_0 \not{E}_1 q_1$

$q_0 \not{E}_1 p_1$ → Mal $\qquad P_2 = P_1 = P_E$

$p_1 \not{E}_1 p_2$

$q_1 \not{E}_1 q_3$ $\qquad A_1 \, E \, A_2 \iff q_0 \, E \, p_1$ $\qquad$ Por lo tanto, A_1 y A_2 sí son equivalentes.

$p_2 \, E_1 \, q_1$

$p_2 \, E_1 \, q_3$

iii) AUTÓMATAS FINITOS ISOMORFOS.

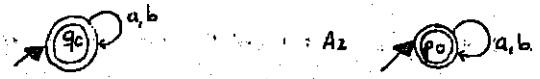
Dos autómatas A_1, A_2 son ISOMORFOS si comparten el mismo alfabeto y existe una aplicación $i: Q_1 \longrightarrow Q_2$ biyectiva definida:

$$\begin{cases} \bullet\ i(q_{01}) = q_{02} \\ \bullet\ i[\delta_1(q,e)] = \delta_2[i(q), e] & \forall q \in Q_1 \quad \forall e \in \Sigma \\ \bullet\ q \in F_1 \iff i(q) \in F_2 & \forall q \in Q_1 \end{cases}$$

Tanto los estados iniciales como los estados finales se corresponden entre sí.

$$A_1 \simeq A_2 \Longrightarrow A_1 \mathcal{E} A_2$$
$$A_1 \simeq A_2 \not\Longleftarrow A_1 \mathcal{E} A_2$$

Dos autómatas A_1 y A_2 son isomorfos cuando la única diferencia entre ellos es el nombre de los estados. Renombrando uno de ellos puedo obtener el otro. Tienen que tener el mismo nº de estados.



iv) AUTÓMATA MÍNIMO

• DEF. 1 - AUTÓMATA COCIENTE.

Dado un AFD $A = (\Sigma, Q, q_0, \delta, F)$.

A partir de éste se construye otro autómata:

$$\hat{A} = (\Sigma, Q/E, [q_0], \hat{\delta}, \hat{F}) \longrightarrow \text{AUTÓMATA COCIENTE.}$$

$$\hat{\delta}: Q/E \times \Sigma \longrightarrow Q/E$$
$$\hat{\delta}([q], e) \longrightarrow [\delta(q,e)]$$

$$\hat{F} = \{[q] \,/\, q \in F\}$$

1) $\hat{\delta}$ está bien definida: δ' está definida a partir de representantes de una clase de equivalencia. La imagen mediante $\hat{\delta}$ no depende del representante elegido.

$$[q] = [q'] \longrightarrow [\delta(q,e)] = [\delta(q',e)] \quad \forall q, q' \in Q \quad \forall e \in \Sigma$$

2) $\hat{F}$ está bien definido: En cada clase de equivalencia del conjunto cociente Q/E todos los estados son finales o todos los estados son no finales.

↳ Un estado final y uno no final nunca pueden ser equivalentes.

Cada una de las clases de P_E va a ser un estado del autómata cociente.

· TEOREMA:

Sea A un AFD.

Existe $\hat{A}$, equivalente a A, con un nº mínimo de estados, único salvo isomorfismo.

El autómata mínimo que estábamos buscando es $\hat{A}$.

DEM: · $\hat{A} \in A$ $\quad L(\hat{A}) = L(A)$

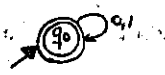
$x \in L(\hat{A}) \iff x \in L(A)$

· $\hat{A}$ es mínimo. $\quad A' \in A \longrightarrow |Q'| \geq |Q/E|$ $\quad \varphi : Q' \longrightarrow Q/E$ ↳ suprayectiva

· $\hat{A}$ es único salvo isomorfismo.

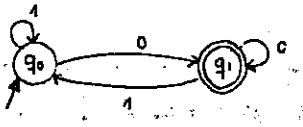
$$\left.\begin{array}{l} A' \in A \\ |Q'| = |Q/E| \end{array}\right\} A' \simeq \hat{A} \quad \varphi : Q' \longrightarrow Q/E \text{ biyectiva.}$$

(17/11/05)

EJ: 

$P_0 = \{[q_0]\}$ $\quad P_0 = P_E$

P_E sólo tiene una clase, por lo que $\hat{A}$ tendrá un único estado.

EJ: 

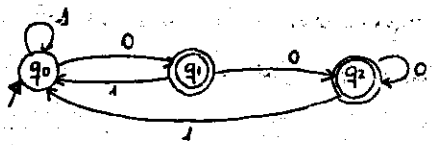
$P_0 = \{[q_0], [q_1]\}$ → CADA CLASE TIENE UN SÓLO ESTADO.

NO PODRÁN ESTAR EN LA MISMA CLASE NUNCA.

POR LO TANTO, P_1 TIENE QUE SER IGUAL A P_0.

$P_1 = P_0 = P_E$

P_E tiene 2 clases de equivalencia. $\hat{A}$ tendrá 2 estados. EL AUTÓMATA DADO YA ERA MÍNIMO.

EJ: 

$P_0 = \{[q_0], [q_1, q_2]\}$ → ¿QUÉ PUEDE OCURRIR?

- Las clases no se pueden mezclar a medida que vamos calculando, por lo que $[q_0]$ permanecerá tal cual.
- Lo único que puede pasar es:

$[q_1, q_2]$ → $[q_1], [q_2]$ o $[q_1, q_2]$ ¿Son 1-equivalentes?

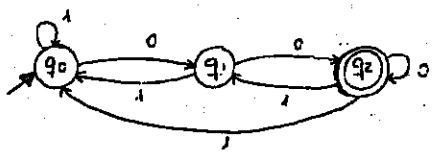
q_1, q_2 $\quad e=0$ $\delta(q_1, 0) = q_2$, $\delta(q_2, 0) = q_2$ } Equivalente consigo mismo $\quad e=1$ $\delta(q_1, 1) = q_0$, $\delta(q_2, 1) = q_0$

$P_0 = P_1 = P_E$ → $\hat{A}$ tendrá dos estados.

q_1 y q_2 son equivalentes → Hay que quitar uno.

* Aquellas clases que contienen estados finales dan lugar a estados finales de

EJ: ¿Y si q_1 no fuera final?



$P_0 = \{[q_0, q_1], [q_2]\}$

↳ Permanecerá igual

↳ Puede pasar → $[q_0, q_1]$ ¿Son 1-equivalentes?

→ $[q_0], [q_1]$

$\delta(q_0, 0) = q_1$
$\delta(q_1, 0) = q_2$
$\delta(q_0, 1) = q_0$
$\delta(q_2, 1) = q_0$

} no son 0-eq. ⟶ q_0 y q_1 no son 1-eq.

$P_1 = \{[q_0], [q_1], [q_2]\}$ ⟶ CADA CLASE TIENE UN SOLO ESTADO.

$P_2 = P_1$ OBLIGATORIAMENTE.

NO PUEDE HABER REFINAMIENTO ADICIONAL.

P_E tiene 3 clases de equivalencia. EL AUTÓMATA DADO YA ERA MÍNIMO.

(28/11/05)

3. AUTÓMATAS FINITOS NO DETERMINISTAS (AFND)

$$A = (\Sigma, Q, q_0, f, F)$$

$$f: Q \times (\Sigma \cup \{\lambda\}) \to P(Q) = 2^Q \qquad f: Q \times (\Sigma \cup \{\lambda\}) \to 2^Q$$

↳ Conjunto potencia de Q.
Conjunto de todos los subconjuntos que se pueden formar con Q.

La diferencia entre un AFD y un AFND está únicamente en la función f.

AFD: $f: Q \times \Sigma \to Q$ → Para cada transición hay un único estado siguiente.

AFND: Pueden darse varios posibilidades
- → Puede no haber estado siguiente.
- → Puede haber un " " "
- → Puede haber varios " "

AFD: No hay λ-transiciones; cuando no recibe ninguna entrada permanece en el estado.

AFND: Puede haber λ-transiciones; cuando no recibe ninguna entrada puede realizar una transición, elegida entre un conjunto de posibles estados siguientes.

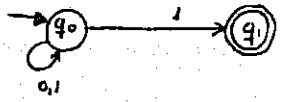
Los AFD son un caso particular de los AFND: cuando no hay λ-transiciones y sólo hay un único estado siguiente.

Al igual que los AFD son un subconjunto de los AFND, el conjunto de lenguajes aceptados por los AFD es un subconjunto de los lenguajes aceptados por los AFND. Ambos aceptan el mismo tipo de lenguajes LENGUAJES REGULARES.

$$L(AFND, \Sigma) = L(AFD, \Sigma) = L_3$$

La potencia de cómputo de los AFND no es mayor que la de los AFD: no existe ningún lenguaje regular que sea aceptado por un AFND y no lo sea por un AFD.

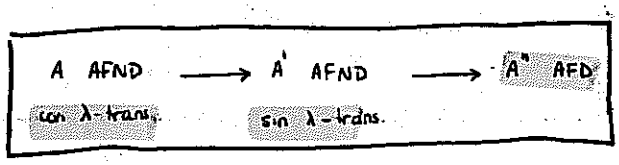
Ej:



$$f(q_0, 1) = \{q_0, q_1\}$$
$$f(q_1, 0) = f(q_1, 1) = \emptyset$$

¿1 ∈ L(A)?
¿01 ∈ L(A)?
} Sí.

* Partiendo de un AFND con λ-transiciones, vamos a obtener un autómata intermedio (también AFND pero sin λ-transiciones) y por último obtendremos un AFD.



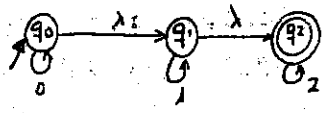
$$\left[L(A) = L(A') = L(A^*) \right]$$

AFD: $\lambda \in L \iff q_0$ es final.

AFND: $\lambda \in L \iff q_0$ es final o $\exists$ una secuencia de λ-transiciones que haga que partiendo del estado inicial llegue a un estado final.

EJ.

$\Sigma = \{0, 1, 2\}$



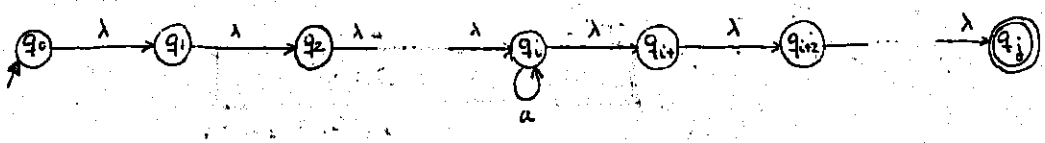
$L(A) = \emptyset^* 1^* 2^*$

(*)

$e = 0 \longrightarrow$ Puede quedarse en q_0

Puede leer 0λ y acabar en q_1

" " $0\lambda\lambda$ " " q_2



$\lambda a = a = a\lambda$

$\lambda\lambda\lambda\lambda \dots \lambda = \lambda$

$\lambda\lambda\lambda\dots\lambda a \lambda\lambda \dots \lambda = a$

Lenguaje aceptado por este autómata:

$L(A) = \{a, \lambda\}$

(*) Diseñar un AFD para este lenguaje con menos de 3 estados.

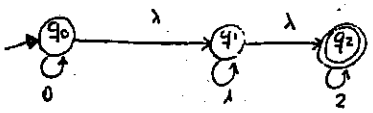
1) Eliminar las λ-transiciones
2) Obtener el AFD equivalente.
3) Minimización.

- λ-CLAUSURA DE UN ESTADO: $\Omega(q)$

La λ-clausura de un estado q es el conjunto formado por ese estado q y todos los estados accesibles desde el estado q mediante una secuencia de n λ-transiciones ($n > 0$).

$$\left[q \in Q,\ \Omega(q) = \{ q \wedge q' \text{ tales que } q \xrightarrow{\lambda^n} q',\ \forall n > 0 \} \right] \quad \Omega(\phi) = \phi$$

EJ.



$\Omega(q_0) = \{ q_0, q_1, q_2 \}$
$\Omega(q_1) = \{ q_1, q_2 \}$
$\Omega(q_2) = \{ q_2 \}$

① PARTIENDO DE UN AFND A CON λ-TRANSICIONES, OBTENER AFND A' SIN λ-TRANS.

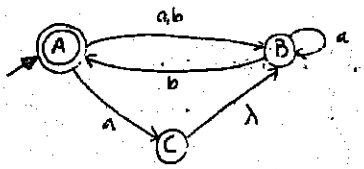
$A = (\Sigma, Q, q_0, \delta, F)$
$A' = (\Sigma, Q, q_0, \delta', F')$

$$F' = \{ q \,/\, \Omega(q) \wedge \neq \emptyset \}$$

$\delta' : Q \times \Sigma \longrightarrow 2^Q$
$\delta'(q, e) = \cup \Omega\, \delta(q_i, e) \qquad q_i \in \Omega(q)$

EJ:

A



$\Omega(A) = \{A\}$
$\Omega(B) = \{B\}$
$\Omega(C) = \{C, B\}$

δ'	a	b
A	B, C	B
B	B	A
C	B	A

⚠ De C para a B al leer λa
De C para a A al leer λb

$$\delta'(A,a) = \bigcup \Omega\, \delta(q_i,a) \;/\; q_i \in \Omega(A) = \bigcup \Omega\, \underbrace{\delta(A,a)}_{C} = \bigcup \Omega(C) = \Omega(C) = \{C,$$

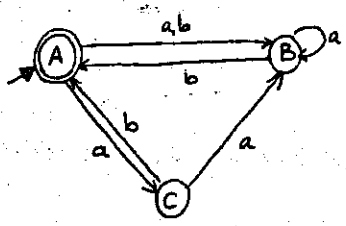
$$\delta'(A,b) = \bigcup \Omega\, \delta(q_i,b) \;/\; q_i \in \Omega(A) = \Omega\, \underbrace{\delta(A,b)}_{B} = \Omega(B) = \{B\}$$

$$\delta'(B,a) = \bigcup \Omega\, \delta(q_i,a) \;/\; q_i \in \Omega(B) = \Omega\, \underbrace{\delta(B,a)}_{B} = \Omega(B) = \{B\}$$

$$\delta'(C,a) = \bigcup \Omega\, \delta(q_i,a) \;/\; q_i \in \Omega(C) = \Omega\, \underbrace{\delta(C,a)}_{\emptyset} \cup \Omega\, \underbrace{\delta(B,a)}_{B} = \{B\}$$

$$\delta'(C,b) = \Omega\, \underbrace{\delta(C,b)}_{\emptyset} \cup \Omega\, \underbrace{\delta(B,b)}_{A} = A$$

A'



② PARTIENDO DEL AFND A" SIN λ-TRANSICIONES, OBTENER EL AFD A'

$$A' = (\Sigma, 2^{\Phi}, \{q_0\}, \delta', F')$$

$$\delta' : 2^{\Phi} \times \Sigma \longrightarrow 2^{\Phi} \qquad ; C = \{q_1, q_2, \dots, q_i\} \quad C \in 2^{\Phi}$$

$$\delta'(C,e) = C' \;/\; C' = \bigcup \delta(q,e) \quad \forall q \in C$$

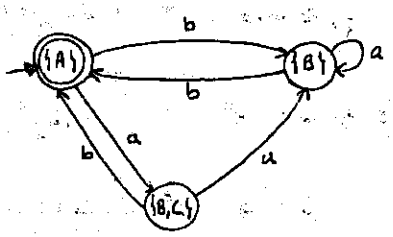
$$F' = \{C \;/\; \exists q \in C \wedge q \in F\}$$

δ'	a	b
→ {A}	{B,C}	{B}
{B}	{B}	{A}
← {B,C}	{B}	{A}

* QUEREMOS QUE A' SEA CONEXO, POR LO QUE PONEMOS EN LA PARTE IZQUIERDA DE LA TABLA LOS ESTADOS QUE NOS VAN SALIENDO.
Hago la 1ª fila (estado inicial). Luego el B porque es accesible, etc...

A"

AFD.

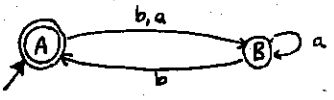


Minimización:

$P_0 = \{[A], [B, BC]\}$

$P_1 = \{[A], [B, BC]\}$

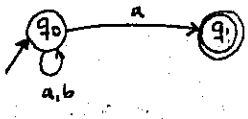
$P_0 = P_1 = P_E$ → Tiene dos clases de equivalencia sobre un estado.



AUTÓMATA MÍNIMO, EQUIVALENTE AL DE PARTIDA.

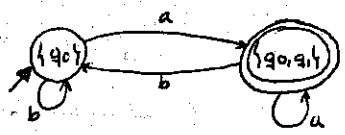
$L(A''\ \text{mínimo}) = [(b+a)a^*b]^*$

EJ:



A'

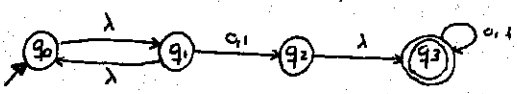
δ''	a	b
→ $\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0\}$



A*

A" es mínimo.

EJ:



A

$\Omega(q_0) = \{q_0, q_1\}$

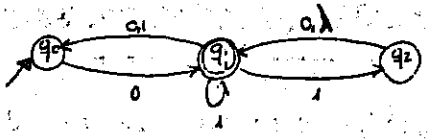
$\Omega(q_1) = \{q_0, q_1\}$

$\Omega(q_2) = \{q_2, q_3\}$

$\Omega(q_3) = \{q_3\}$

δ'	0	1
q_0	q_2	q_2
q_1	q_2	q_2
q_2	q_3	q_3
q_3	q_3	q_3

EJ:



A

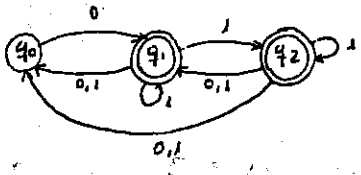
$\Omega(q_0) = \{q_0\}$

$\Omega(q_1) = \{q_1\}$

$\Omega(q_2) = \{q_2, q_1\}$

δ'	0	1
q_0	q_1	$\emptyset$
finales → q_1	q_0	q_0, q_1, q_2
finales ↘ q_2	q_1, q_0	q_0, q_1, q_2

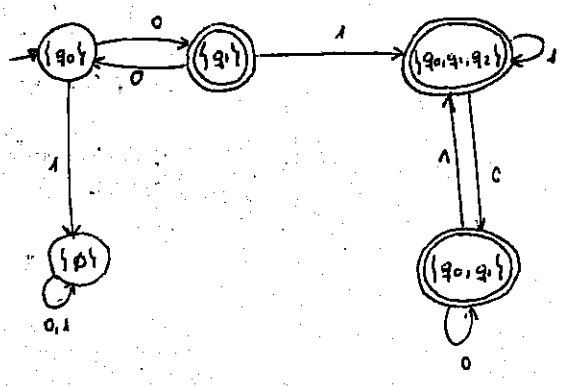
A'



$\delta'(q_2, 1) = \cup\, \Omega\delta(q_i, 1) = \Omega\delta(q_2, 1) \cup \Omega\delta(q_1, 1)$
$= \Omega(\phi) \cup \Omega(q_0, q_1, q_2) = q_0, q_1, q_2$

δ''	0	1
→ $\{q_0\}$	$\{q_1\}$	$\emptyset$
$\{q_1\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$
Finales Ⓐ $\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
Finales Ⓐ $\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
$\{\phi\}$	$\{\phi\}$	$\{\phi\}$

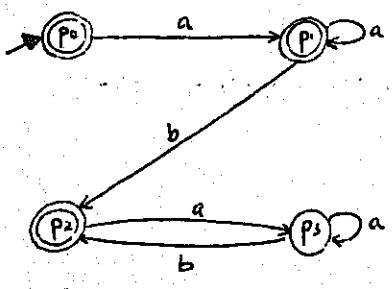
A''



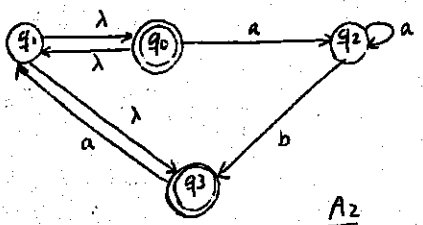
Ⓐ SON FINALES LOS QUE CONTIENEN ALGÚN ESTADO FINAL

PE → ver si A'' es mínimo.

EJ:



A_1



A_2

- Determinar si A_1 y A_2 son mínimos.
- ¿$A_1 \in A_2$? Construyendo el autómata suma → $\hat{A}_1 \simeq \hat{A}_2 \Rightarrow A_1 \in A_2$